




**Flexsim Simulation Software**  
**User Guide**

**Version 4.0**

*flexibility to the power of* 

Flexsim Simulation Software Version 4.0

**Release Date: March 7, 2007**

**Flexsim Software Products, Inc.**

1577 North Technology Way

Orem, Utah 84097

801 224-6914

**[support@flexsim.com](mailto:support@flexsim.com)**

**[www.flexsim.com](http://www.flexsim.com)**

# FLEXSIM SOFTWARE LICENSE AGREEMENT

This license is a legal agreement between you, the “end user,” and Flexsim Software Products, Inc., a Utah corporation (“Flexsim”). Use of the accompanying software indicates your acceptance of these terms. As used in this Agreement, the capitalized term “Software” means the Flexsim Simulation Software included on the CD or disk media provided with this Agreement. The term “Software” does not include any software that is covered by a separate license offered or granted by a person other than Flexsim. IF YOU DO NOT AGREE TO THESE TERMS AND CONDITIONS, PROMPTLY DISCONTINUE THE INSTALLATION AND DOWNLOAD PROCESS AND EITHER DESTROY OR RETURN, INTACT, THE FLEXSIM PACKAGE CONTAINING THE CD OR DISK MEDIA, TOGETHER WITH THE OTHER COMPONENTS OF THE PRODUCT TO THE PLACE OF PURCHASE.

1. Ownership. The Software and any accompanying documentation are owned by Flexsim or its licensors and are protected under U.S. copyright laws and international treaty provisions. Ownership of the Software, and all copies, modifications, and merged portions thereof shall at all times remain with Flexsim or its licensors.

2. Grant of License.

a) The Software and accompanying documentation are being licensed to you, which means you have the right to use the Software only in accordance with this Agreement.

b) If you have paid for a single user license, Flexsim grants you the right to use one copy of the Software on a single computer. The Software is considered in use on a computer when it is loaded into temporary memory or installed into permanent memory. If the Software is installed on a server or a computer that can be accessed by more than one PC or workstation, a single user license allows the Software to be accessed only by a single PC or workstation connected to the server or computer at any given time.

c) If you have paid for a multiple user license, Flexsim grants you the right to use the number of copies of the Software for which you have paid and to load the Software onto that number of computers or permit that number of PCs or workstations to access the Software on a server or other computer.

d) If you have paid for a site license, Flexsim grants you the right to use any number of copies of the Software on any number of computers at the designated site.

3. Copies. You are authorized to make a single copy of the Software solely for backup purposes. The backup copy of the Software must include our copyright notice. You may not make any other copies of the Software, except for copies that you are authorized to use under the applicable license.

4. Scope of Use. You may not sublicense or lease the Software or any of the accompanying documentation to any other person. You may use the Software only for your own internal business purposes. You may use the Software in connection with services you provide to other persons or entities, but you may not use the Software to process data for any other person or entity.

5. Nonpermitted Uses. Without the express permission of Flexsim, you may not (a) use, copy, modify, alter, or transfer, electronically or otherwise, the Software or any of the accompanying documentation except as expressly permitted in this Agreement, or (b) translate, reverse program, disassemble, decompile or otherwise reverse engineer the Software.

6. Transfer of Software. You may transfer the Software to another person or entity, provided that the transferee accepts wholly the terms and conditions of this license. Any transfer must include all updates and all accessible prior versions of the Software, and all copies of the Software must be removed from your computer (or computers) and delivered to the transferee or destroyed

7. Term. This license is effective from your date of purchase and shall remain in force until terminated. You may terminate the license and this License Agreement at any time by destroying the Software and the accompanying documentation, together with all copies in any form.

8. Export Controls. Certain uses of the Software by you may be subject to restrictions under U.S. regulations relating to exports and ultimate end uses of computer software. You agree to fully comply with all applicable U.S. laws and regulations, including but not limited to the Export Administration Act of 1979 as amended from time to time and any regulations promulgated thereunder.

9. U.S. Government Restricted Rights. If you are acquiring the Software on behalf of any unit or agency of the United States Government, the following provision applies: It is acknowledged that the Software and the documentation were developed at private expense and that no part is in the public domain and that the Software and documentation are provided with restricted rights. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of the Commercial Computer Software-Restricted Rights at 48 CFR 52.227-19, as applicable.

10. Limited Warranty.

a) Flexsim warrants to you, the original end user, that the Software will perform substantially in accordance with the accompanying documentation. Flexsim does not warrant that the Software is error-free or that the Software will operate without interruption. This Limited Warranty extends for ninety (90) days from the date the Software is paid for or first used, whichever occurs first.

b) This Limited Warranty does not apply to any Software that has been altered, damaged, abused, misapplied, or used other than in accordance with this license and any instructions included on the Software and the accompanying documentation.

c) Flexsim's entire liability and your exclusive remedy under this Limited Warranty shall be the repair or replacement of any Software that fails to conform to this Limited Warranty or, at Flexsim's option, return of the license fees paid for the Software. Flexsim shall have no liability under this Limited Warranty unless the Software is returned to Flexsim or its authorized representative within the warranty period. Any replacement Software will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer.

d) THIS LIMITED WARRANTY IS IN LIEU OF AND EXCLUDES ALL OTHER WARRANTIES NOT EXPRESSLY SET FORTH HEREIN, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR WARRANTIES ARISING FROM USAGE OF TRADE OR COURSE OF DEALING.



e) THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS, YOU MAY HAVE OTHERS WHICH VARY FROM STATE TO STATE.

11. Limitation of Liability. Except for a return of the license fees under the circumstances provided under the Limited Warranty, NEITHER FLEXSIM NOR ITS LICENSORS SHALL IN ANY EVENT BE LIABLE FOR ANY MONEY DAMAGES WHATSOEVER ARISING OUT OF OR RELATED TO THE USE OF OR INABILITY TO USE THE SOFTWARE, INCLUDING DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER SUCH LIABILITY IS BASED ON CONTRACT, TORT, WARRANTY, OR ANY OTHER LEGAL OR EQUITABLE GROUNDS. IN NO EVENT SHALL FLEXSIM'S LIABILITY RELATED TO ANY OF THE SOFTWARE EXCEED THE LICENSE FEES ACTUALLY PAID BY YOU FOR THE SOFTWARE. Without limiting the generality of the foregoing, Flexsim shall not be liable for losses arising out of any business interruption, any loss of business profits, or loss of business information caused by or in any way related to the use of, or inability to use, the Software, even if Flexsim has been advised of the possibility of such losses. Because some states do not allow the exclusion or limitation of liability for consequential or incidental damages, some parts of the above limitation may not apply to you.

12. Entire Agreement. This Agreement constitutes the entire agreement between the parties pertaining to its subject matter, and it supersedes any and all written or oral agreements previously existing between the parties with respect to such subject matter.

13. Amendment and Modification. No modification or amendment of this Agreement shall be binding unless executed in writing by both parties. No waiver shall be effective unless it is in writing and signed by the party against whom enforcement is sought.

14. Assignment. This Agreement may not be assigned by either party without the prior written consent of the other.

15. Choice of Law/Forum. This Agreement shall be governed by and construed in accordance with Utah law, without regard to its rules regarding conflicts of law. Each of the parties consents to the jurisdiction of the courts located in the state of Utah with respect to all matters relating to this Agreement and agrees that all litigation relating to this Agreement shall take place in courts located in the state of Utah.







# Table Of Contents

Welcome to Flexsim Help .....	1
What's New in Flexsim.....	3
4.0 .....	3
Getting Started .....	5
Getting Started with Flexsim and Simulation.....	5
Model Description.....	9
Building the Model.....	10
Detailing the Model.....	15
Experimenting with the Model .....	25
Tutorials.....	37
Tutorials.....	错误!未定义书签。
Lesson 1.....	38
Lesson 1 .....	错误!未定义书签。
Flexsim Software Concept Learning .....	39
Model 1 Description .....	41
Step-By-Step Model Construction .....	42
Lesson 2.....	54
Lesson 2 .....	54
Flexsim Software Concept Learning .....	55
Model 2 Description .....	66
Step-By-Step Model Construction .....	67
Lesson 2 Extra Mile.....	78
Lesson 2 Extra Mile .....	78
Step-By-Step Model Construction .....	79
Lesson 3.....	94
Lesson 3 .....	94
Flexsim Software Concept Learning .....	95
Model 3 Description .....	103
Step-By-Step Model Construction .....	104
Fluid Objects Tutorial .....	116
Fluid Object Lesson.....	116
Flexsim Software Concept Learning .....	117
Fluid Lesson Description .....	119
Step-By-Step Model Construction .....	121
Flexsim Concepts.....	131
Modeling Views .....	141
Orthographic/Perspective View Window .....	141
Planar View .....	148
Tree Window .....	149
Library Icon Grid.....	151
Custom User Libraries.....	151
Keyboard Interaction .....	157
Menus and Settings .....	160
View Window Settings for Ortho/Perspective View .....	160
View Window Settings for Planar View .....	164
Views Toolbar .....	166
Edit Selected Objects Toolbar.....	167
Ortho/Perspective Popup Menu .....	171
Light Source Editor .....	173

Model View Toolbars.....	174
Mode Toolbar.....	174
Travel Networks Toolbar .....	176
Model Layouts Toolbar .....	177
Edit Highlighted Object Toolbar.....	178
Find Objects Toolbar .....	179
Groups Toolbar .....	180
Main Menu and Toolbar .....	181
File Menu .....	181
Edit Menu .....	184
View Menu.....	185
Build Menu.....	187
Execute Menu .....	188
Stats Menu .....	189
Tools Menu.....	191
Window Menu.....	193
Help Menu .....	194
Flexsim Toolbar .....	196
Simulation Run Panel.....	198
Buttons .....	198
General Windows.....	201
Attribute Hints .....	201
Command Hints.....	202
Completion Hints .....	203
Find Replace .....	204
Preferences Window .....	205
Fonts and Colors .....	205
Sample Models.....	207
Reports and Statistics .....	208
Trace Debugger .....	213
Tree Browse Dialog.....	214
Database Table View .....	215
Table Editor .....	216
Object Windows .....	217
Object Parameters and Properties Windows.....	217
Parameters Pages .....	219
ASRSvehicle Tab Page .....	219
BasicFR Advanced Page.....	220
BasicTE Page .....	224
Collision Tab Page.....	228
Combiner Tab Page.....	231
Conveyor Tab Page.....	232
Layout Tab Page .....	236
Crane Tab Page .....	238
Dispatcher Tab Page.....	240
Elevator Tab Page .....	241
FixedResource Page .....	242
Flow Tab Page.....	243
Flowitem Page .....	246
FlowNode Tab Page .....	247
MergeSort Tab Page .....	248

MultiProcessor Tab Page .....	250
NetworkNode Connection Tab Page.....	252
NetworkNode Tab Page .....	254
Operators Tab Page .....	255
Photo Eyes Tab Page.....	257
Processor Tab Page.....	258
ProcessTimes Tab Page .....	259
Queue Tab Page .....	260
Rack Tab Page .....	262
Rack Size Table Tab Page.....	264
Recorder Parameters .....	266
Reservoir Tab Page.....	273
Robot Tab Page.....	274
Separator Tab Page .....	275
Sink Page.....	276
Source Tab Page.....	277
TaskExecutor Tab Page .....	279
Traffic Control Network Node Page .....	281
Traffic Control Page.....	282
Transporter Tab Pages.....	285
Object Trigger Functions .....	286
FluidBlender Tab Page .....	288
FluidBlenderRecipe Tab Page .....	290
FluidGenerator Tab Page .....	291
FluidLevelDisplay Tab Page.....	293
FluidMixer Tab Page.....	295
FluidMixerSteps Tab Page .....	297
FluidPipe Tab Page .....	299
FluidPipeLayout Tab Page .....	301
FluidProcessor Tab Page.....	303
FluidSplitter Tab Page .....	305
FluidSplitterPercents Tab Page.....	307
FluidTank Tab Page .....	308
FluidTankMarks Tab Page .....	310
FluidTerminator Tab Page .....	311
FluidTicker Tab Page.....	312
FluidToItem Tab Page .....	313
Initial Product GUI.....	315
ItemToFluid Tab Page .....	316
Ticker .....	318
Properties Pages.....	320
Visual Properties Page .....	320
General Properties Page .....	323
Labels Tab Page.....	325
Statistics Properties Page .....	327
Flexsim Object Library .....	333
Flexsim Object Library .....	333
ASRSvehicle .....	335
BasicFR.....	337
BasicTE.....	338
Combiner .....	339

Conveyor .....	342
Crane.....	346
Dispatcher .....	347
Elevator .....	348
FixedResource .....	349
FlowNode .....	354
MergeSort.....	356
MultiProcessor.....	359
NetworkNode.....	361
Operator .....	369
Processor .....	371
Queue.....	373
Rack.....	375
Recorder.....	377
Reservoir .....	378
Robot.....	379
Separator .....	380
Sink.....	382
Source .....	383
TaskExecutor .....	385
TrafficControl.....	390
Transporter .....	394
VisualTool.....	395
Fluid Objects Library .....	410
FluidBlender .....	412
FluidGenerator .....	414
FluidMixer .....	415
FluidPipe.....	417
FluidProcessor .....	419
FluidSplitter.....	420
FluidTank.....	422
FluidTerminator .....	424
FluidToItem.....	425
ItemToFluid.....	426
Modeling Tools .....	427
Modeling Tools .....	427
AVI Maker.....	428
Excel Interface.....	430
Flowitem Bin.....	431
Global Tables .....	432
Global Time Tables .....	434
Global User Events .....	437
Global Task Sequences .....	439
Global Variables .....	442
Graphical User Interfaces .....	444
Import Media Files.....	460
Model Startup Code .....	461
MTBF/MTTR.....	462
Multiple Table Excel Import.....	466
OptQuest Optimizer .....	470
Presentation Builder .....	475



Script Editor .....	478
Simulation Experiment Control .....	479
Single Table Export .....	485
Single Table Import .....	486
Sky Box Editor .....	487
Table Configurator .....	489
User Commands .....	491
Microsoft Visio™ Import .....	493
Watch List.....	515
Pick Lists .....	517
Pick Lists .....	517
Triggers .....	520
Load/Unload Trigger (On Load/On Unload) .....	520
Entry/Exit Trigger (On Entry/On Exit) .....	521
Message Trigger (On Message).....	522
Process Finish Trigger (On Process Finish) .....	523
Creation Trigger (On Creation).....	524
Collision Trigger (Handle Collision) .....	525
Node Entry Trigger (On Continue/On Arrival) .....	526
OnCover OnUncover Trigger Picklist .....	527
OnChange Trigger .....	528
OnDraw Trigger (Custom Draw Code).....	529
OnEmpty or OnFull Trigger .....	531
Reset Trigger (On Reset).....	532
Down/Up Trigger (On Break Down/On Repair).....	533
Breakdown/Repair Trigger (On Break Down/On Repair) .....	534
OnResourceAvailable .....	535
Time Pick Lists .....	536
Time Picklists.....	536
Load/Unload Time .....	537
Minimum Staytime (Minimum Dwell Time).....	538
Setup Time.....	539
Time Picklist (Inter-Arrivaltime Usage).....	540
Cycle Time (Process Time).....	541
Fixed Resources .....	542
Down Dispatcher (Pick Operator).....	542
Flow Rate.....	543
Place in Bay .....	544
Place in Level.....	545
Process Dispatcher (Pick Operator).....	546
Receive From Port (Pull From Port).....	547
Pull Requirement .....	548
Rise/Fall Through Mark Triggers .....	549
Send Requirement.....	550
Send To Port.....	551
Split Quantity (Split/Unpack Quantity) .....	552
Transport Dispatcher (Request Transport From).....	553
Item Speed (Speed) .....	554
Mobile Resources.....	555
Break To ("Break To" Requirement).....	555
Pass To.....	556

Queue Strategy .....	557
Experimentation .....	558
End of Experiment .....	558
End of Run (End of Replication) .....	559
End of Scenario .....	560
End of Warmup (End of Warmup Period) .....	561
Pass To.....	562
Start of Experiment .....	563
Start of Run (Start of Replication) .....	564
Start of Scenario .....	565
Visualization .....	566
Table X Val (X Value) .....	566
Table Y Val (Y Value) .....	567
Text (Text Display) .....	568
Text Code (Text Display) .....	569
Task Sequences .....	571
Flexsim Task Sequences.....	571
Custom Built Task Sequences.....	575
Querying Information on Task Sequences .....	577
Task Sequence Preempting.....	579
Coordinated Task Sequences.....	581
Task Type Quick Reference .....	585
Task Types .....	587
Coordinated Task Types .....	606
Charting and Reporting.....	609
Charting and Reporting Overview .....	609
Flexsim Chart .....	611
Summary Report .....	614
State Report .....	615
Object Comparison Chart .....	617
Time Plot .....	622
Financial Reports .....	626
Object Gantt Chart .....	631
Single Object Chart .....	635
Flowitem Gantt Chart .....	641
Database Tables .....	644
Interacting with Graphs .....	645
Flexsim Coding .....	651
Writing Logic in Flexsim .....	651
Basic Modeling Functions and Logic Statements.....	657
Advanced Functions .....	661
3D Media .....	663
Importing 3D Media .....	663
Preparing a 3D File .....	664
Importing AutoCAD Drawings .....	668
Using Frames .....	670
Level Of Detail (LOD) .....	671
Transparency .....	673
Miscellaneous Concepts.....	675
Introduction to Flexsim Tree Structure.....	675
When To Compile Flexsim.....	678

State List.....	679
Kinematics.....	681
Creating Custom Libraries .....	689
Dropscrip and Droppath .....	689
Automatic Install .....	694
Icon Grid Width and Height .....	696
View Attributes Reference .....	697
View Window Classes .....	697
General Attributes .....	698
Index.....	717



# Welcome to Flexsim Help

Thank you for choosing Flexsim. This user manual is meant to help you get to know Flexsim better. It is also a reference guide for experienced users with questions about different facets of Flexsim.

Below is a list of help sections included in this manual. You can also go to What's New to see what's been added to recent releases.

## Help Sections

- Getting Started with Flexsim
- Tutorials
- Modeling Views
- Main Menu and Toolbar
- General Windows
- Object Windows
- Flexsim Object Library
- Modeling Tools
- Pick Lists
- Task Sequences
- Charting and Reporting
- Miscellaneous Concepts



# What's New in Flexsim

The following is a list of features and fixes that have been included in the latest Flexsim releases.

## 4.0

The 4.0 release is a major release. Many new features have been included with this new version.

- Users may choose whether or not to compile their models
- Visual Studio C++ and the need to compile are no longer requirements. (version 3.x models will still need to be compiled, however)
- New Flexscript interpreter and error reporting
- Flexscript is the default language for model-building
- Flexscript allows local variables (int, double, string, treenode)
- Flexscript allows arrays of local variables
- Flexscript has new C++-like statements (for, switch, while)
- Code fields can be toggled Flexscript/C++/DLL
- New Library View
- New menus
- Tools menu added – replaces Toolbox
- Global Task Sequences available in Tools menu
- Global Variables available in Tools menu – replaces Global Object Pointers
- Fluid objects added to the Standard Library
- Forward/Backwards buttons added to Parameters and Properties windows
- Pick-lists have fewer options, and remaining options are more robust
- Pick-lists options have been rewritten to be easier to read and use
- Trigger pick-list options can easily be combined with other options
- New template-creation system – replaces PARAMSTART/PROSESTART

## Flexsim User Guide

- Templates no longer open in a window separate from the Parameters window
- New code edit control – uses Scintilla editor
- Pop-up parameter hints appear when a command is typed
- Code Edit window has been redesigned to be more user-friendly
- User-written code can be added to User Libraries
- Event-handling speed improvement
- Graphic display speed improvement
- Events, Task Sequences and Kinematics display detailed information in the tree view
- Global Preferences dialog allows more customization
- Smaller software installation
- New charting/reporting application which includes Gantt charts and costing!
- 3D view windows display axes and origin of model space.
- 3D views display object information in the status bar
- New version of ExpertFit included
- New 2D/3D media and sample models available on-line in the Flexsim Gallery
- New updated Users Manual includes more tutorials and explains GUI development
- Command documentation is on-line and can be contributed to by users
- Command documentation can also be downloaded and used locally



# Getting Started

## Getting Started with Flexsim and Simulation

### What is Flexsim?

Flexsim is a powerful analysis tool that helps engineers and planners make intelligent decisions in the design and operation of a system. With Flexsim, you can build a 3-dimensional computer model of a real-life system, then study that system in either a shorter time frame or for less cost than with the actual system.

As a “what-if” analysis tool, Flexsim provides quantitative feedback on a number of proposed solutions to help you quickly narrow in on the optimum solution. With Flexsim’s realistic graphical animation and extensive performance reports, you can identify problems and evaluate alternative solutions in a short amount of time. By using Flexsim to model a system before it is built, or to test operating policies before they are actually implemented, you will avoid many of the pitfalls that are often encountered in the startup of a new system. Improvements that previously took you months or years of trial-and-error experimentation to achieve can now be attained in a matter of days and hours using Flexsim.

### Modeling

In technical terms, Flexsim is classified as a discrete-event simulation software program. This means that it is used to model systems which change state at discrete points in time as a result of specific events. Common states might be classifications such as idle, busy, blocked or down, and some examples of events would be the arrival of customer orders, product movement, and machine breakdowns. The items being processed in a discrete-event simulation model are often physical products, but they might also be customers, paperwork, drawings, tasks, phone calls, electronic messages, etc. These items proceed through a series of processing, queuing and transportation steps in what is termed a process flow. Each step of the process may require one or more resources such as a machine, a conveyor, an operator, a vehicle or a tool of some sort. Some of these resources are stationary and some are mobile, some resources are dedicated to a specific task and others must be shared across multiple tasks.

Flexsim is a versatile tool that has been used to model a variety of systems across a number of different industries. Flexsim is successfully used by small and large companies alike. Roughly half of all Fortune 500 companies are Flexsim clients; including such noted names as General Mills, Daimler Chrysler, Northrop Grumman, Discover Card, DHL, Bechtel, Bose, Michelin, FedEx, Seagate Technologies, Pratt & Whitney, TRW and NASA.

There are three basic problems which can all be solved with Flexsim:

1. Service problems – the need to process customers and their requests at the highest level of satisfaction for the lowest possible cost.
2. Manufacturing problems – the need to make the right product at the right time for the lowest possible cost.

3. Logistic problems – the need to get the right product to the right place at the right time for the lowest possible cost.

### Examples of How Flexsim is Used

To give you ideas for possible projects, Flexsim has successfully been used to:

- improve equipment utilization
- reduce waiting time and queue sizes
- allocate resources efficiently
- eliminate stock-out problems
- minimize negative effects of breakdowns
- minimize negative effects of rejects and waste
- study alternative investment ideas
- determine part throughput times
- study cost reduction plans
- establish optimum batch sizes and part sequencing
- resolve material handling issues
- study effect of setup times and tool changeovers
- optimize prioritization and dispatching logic for goods and services
- train operators in overall system behavior and job related performance
- demonstrate new tool design and capabilities
- manage day-to-day operational decision making

Flexsim has been used successfully in both system design studies and in the managing of systems on a day-to-day operational basis. Flexsim has also been used for training and education purposes. A Flexsim training model can provide insight into the complex dependencies and dynamics of a real-life system. It can help operators and management not only learn how a system operates, but learn what happens when alternative procedures are implemented. Flexsim has been used to build interactive models which can be manipulated while the model is running in order to help teach and demonstrate the cause and effects inherent in system management.

### Visualization

Flexsim is a highly visible technology that can be used by forward-thinking marketers to elevate their company's image and to declare to the outside world that their company takes pride in how it operates.

It is surprising how effective an animated simulation model can be for getting management's attention and influencing their way of thinking. The animation displayed during a simulation provides a superb visual aid for demonstrating how the final system will perform.

### Getting Started Tutorial

In this getting started section, you will become familiar with the advantages of using simulation in your company. You will also get an initial introduction to the Flexsim environment. We will build a simple simulation model, then experiment with the model to see what effect different scenarios may have on the behavior of the model.

## Flexsim Terminology

Before we start, it will be helpful to understand some of the basic terminology of the Flexsim software, and how this terminology applies to general simulation concepts.

### Flexsim objects

Flexsim objects simulate different types of resources in the simulation. An example is the Queue object, which acts as a storage or buffer area. The Queue can represent a line of people, a queue of idle processes on a CPU, a storage area on the floor of a factory, or a queue of waiting calls at a customer service center. Another example of a Flexsim object is the Processor object, which simulates a delay or processing time. This object can represent a machine in a factory, a bank teller servicing a customer, a mail employee sorting packages, etc.

Flexsim objects are found in the Object Library grid panel. The grid is arranged by group. By default the most frequently used objects are shown.

### Flowitems

Flowitems are the objects that move through your model. Flowitems can represent parts, pallets, assemblies, paper, containers, people, telephone calls, orders, or anything that moves through the process you are simulating. Flowitems can have processes performed on them and can be carried through the model by material handling resources. In Flexsim, flowitems are generated by a Source object. Once flowitems have passed through the model, they are sent to a Sink object.

### Itemtype

The itemtype is a label that is placed on the flowitem that could represent a barcode number, product type, or part number. Flexsim is set up to use the itemtype as a reference in routing flowitems.

### Ports

Every Flexsim object has an unlimited number of ports through which they communicate with other objects. There are three types of ports: input, output, and central.

Input and output ports are used in the routing of flowitems. For example, a mail sorter places packages on one of several conveyors depending on the destination of the package. To simulate this in Flexsim, you would connect the output ports of a Processor object to the input ports of several Conveyor objects, meaning once the Processor (or mail sorter) has finished processing the flowitem (or package), it sends it to a conveyor.

Central ports are used to create references from one object to another. A common use for central ports is for referencing mobile objects such as operators, fork lifts, and cranes from fixed resources such as machines, queues, or conveyors. Central ports will not be used in this tutorial.

Ports are created and connected by clicking on one object and dragging to a second object while holding down different letters on the keyboard. If the letter "A" is held down while clicking-and-dragging, an output port will be created on the first object and an input port will be created on the second object. These two new ports will then

be automatically connected. Holding down the “S” key will create a central port on both objects and connect the two new ports. Connections are broken and ports deleted by holding down the “Q” for input and output ports and the “W” key for central ports. The following table shows the keyboard letters used to make and break the two types of port connections.

	<b>Output - Input</b>	<b>Center</b>
<b>Disconnect</b>	Q	W
<b>Connect</b>	A	S

### **Model views**

Flexsim uses a three-dimensional modeling environment. The default model view for building models is called an orthographic view. You can also view the model in a more realistic perspective view. It is generally easier to build the model's layout in the orthographic view, whereas the perspective view is more for presentation purposes. However, you may use any view option to build or run the model. You may open as many view windows as you want in Flexsim. Just remember that as more view windows are opened, the demand on computer resources increases.

## Model Description

### Description

In this model we will look at the process of manufacturing three types of products in a factory. In our simulation model, we will associate an itemtype value with each of the three product types. These three types all arrive intermittently from another part of the factory. There are also three machines in our model. Each machine can process a specific product type. Once products are finished at their respective machines, all three types of products must be tested at a single shared testing station for correctness. If they have been manufactured correctly, they are sent on to another part of the facility, leaving our simulation model. If they were manufactured incorrectly, they must return to the start of the simulation model to be re-processed by their respective machines. The goal of the simulation is to find where the bottle neck is. Is the testing machine causing the three other machines to back up, or is it being starved because the three machines can't keep up with it? Is the amount of buffer space between the two machines important?

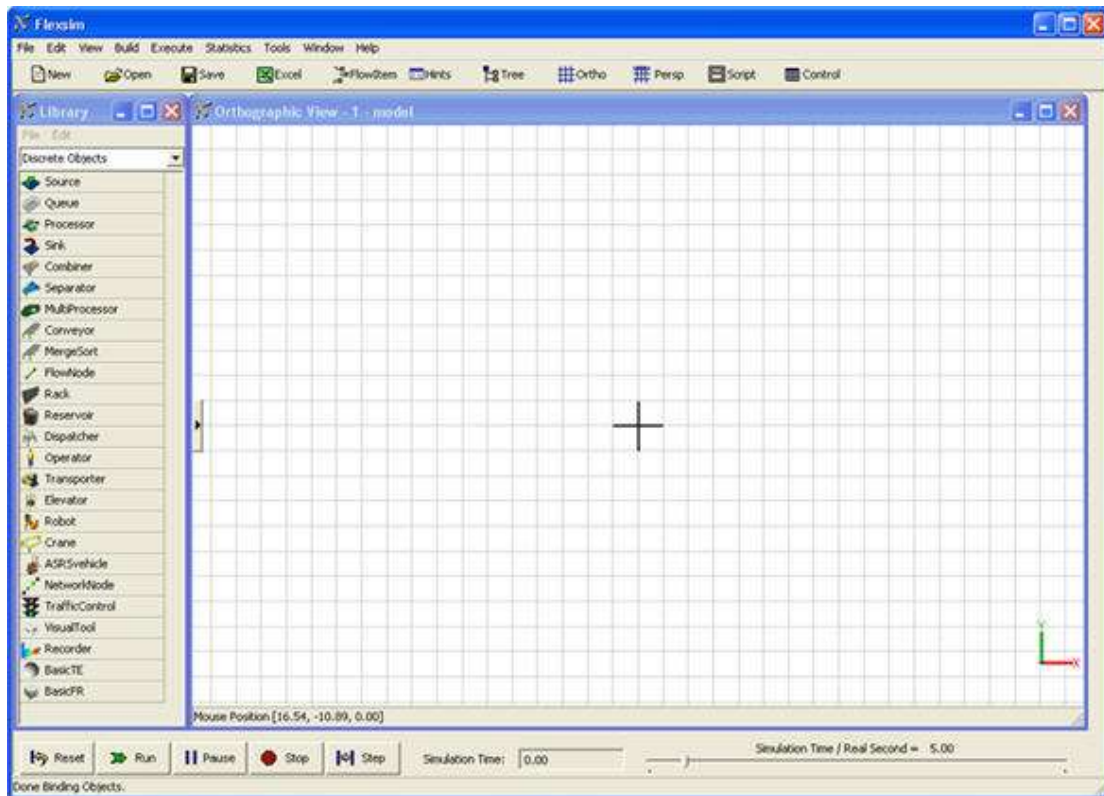
### Applying the Model to Different Industries

While we are using the manufacturing industry for this example, the same type of simulation model can be applied to other industries. Take a copy shop for example. A copy shop has three main services: black and white copies, color copies, and binding. During business hours, there are three employees working. One employee handles black and white copy jobs, another handles color copy jobs, and the third handles binding jobs. There is also a cashier to ring up finished orders. Each customer that enters the copy shop gives a job to the employee that specializes in his type of job. When each job is finished, the cashier takes the finished product or service, gives it to the customer and rings him up for the proper amount. However, sometimes the customer is not satisfied with the job that was done. In such cases, the job must be given back to the respective employee to be done again. This scenario represents the same simulation model as the one described above for the manufacturing industry. Here, though, you may be more concerned with the number of waiting people in the copy shop, as slow service can be very costly to a copy shop's business.

Here's another example of the same simulation model applied to the transportation industry. Commercial shipping trucks traveling over a bridge from Canada into America must go through a customs facility before being allowed to enter the country. Each truck driver must first get the proper paperwork necessary, and then pass through a final inspection of the truck. There are three general weight categories of trucks. Each category has a different type of paperwork to fill out and must apply at a different department of the customs facility. Once paperwork is finished, all weight categories of trucks must go through the same inspection process. If they fail the inspection, then they must go through more paperwork, etc. Again, this situation contains the exact same simulation elements as the manufacturing example, only applied to the transportation industry. Here, you may be interested in how far the trucks back up across the bridge. If they back up for miles and thus block traffic into the neighboring Canadian city, then you may need to change how the facility operates.

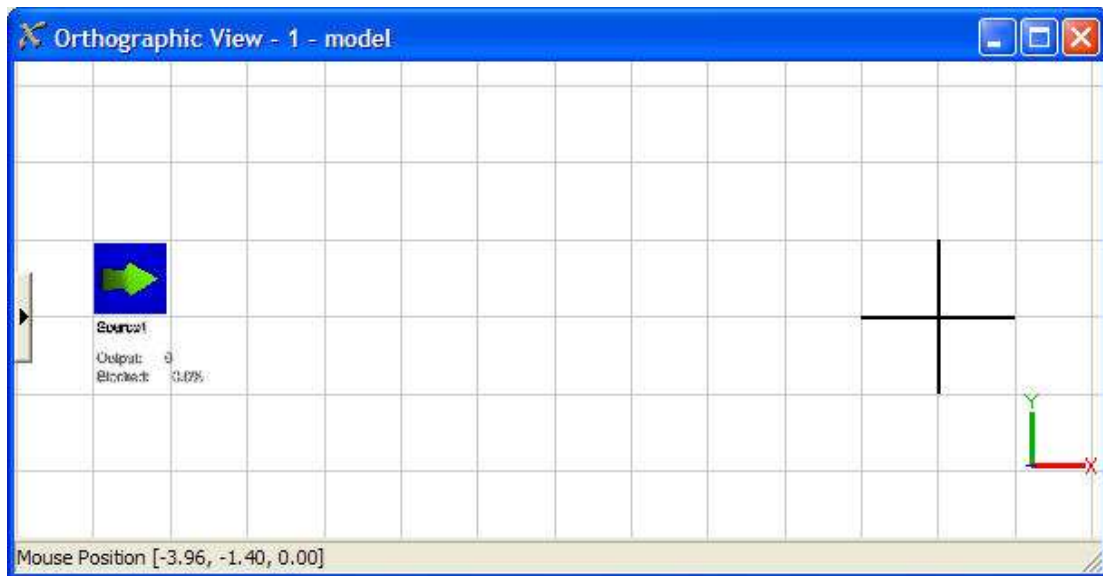
## Building the Model

To verify that the Flexsim software has been installed correctly, open the application by double clicking on the Flexsim icon on your desktop. Once the software loads you should see the Flexsim menu and toolbars, Object Library, and Orthographic Model View windows.



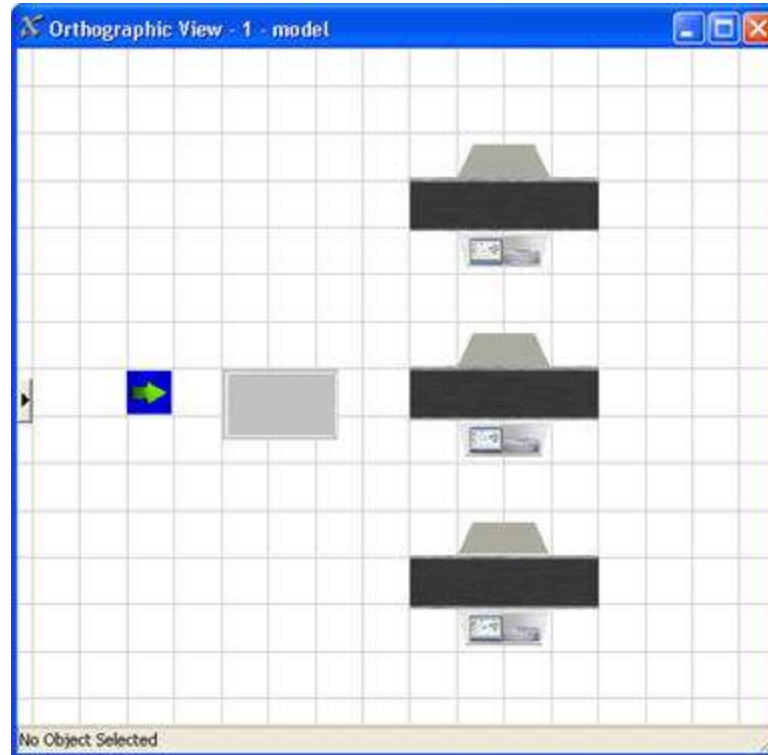
**Step 1: Create an object in the model.**

Drag and drop a Source from the Object Library on the left into the Model View window. To do this, click and hold on the object in the Object Library, then drag it to the position you want to drop it in the model, and release the mouse button. This will create a source object in the model, as shown below. As objects are created, they will be given default names such as Source#, where # is the number of objects created since the Flexsim application was opened. You may rename the objects in your model during the editing process to be defined later.

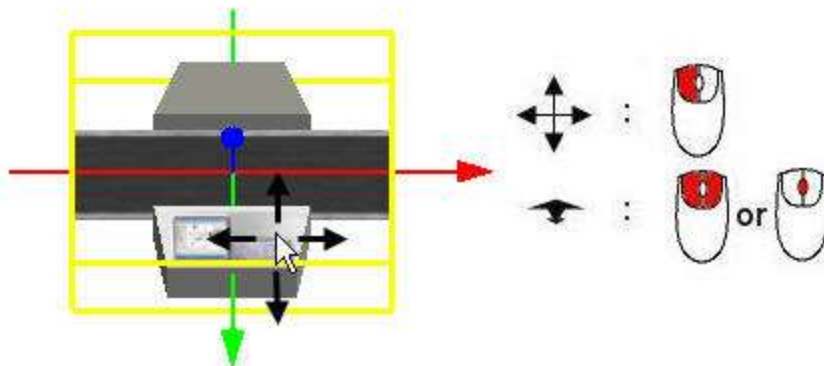


## Step 2: Create more objects in the model.

Drag a Queue object from the library and drop it to the right of the source object.  
 Drag three Processor objects from the library and drop them to the right of the queue, as shown below.

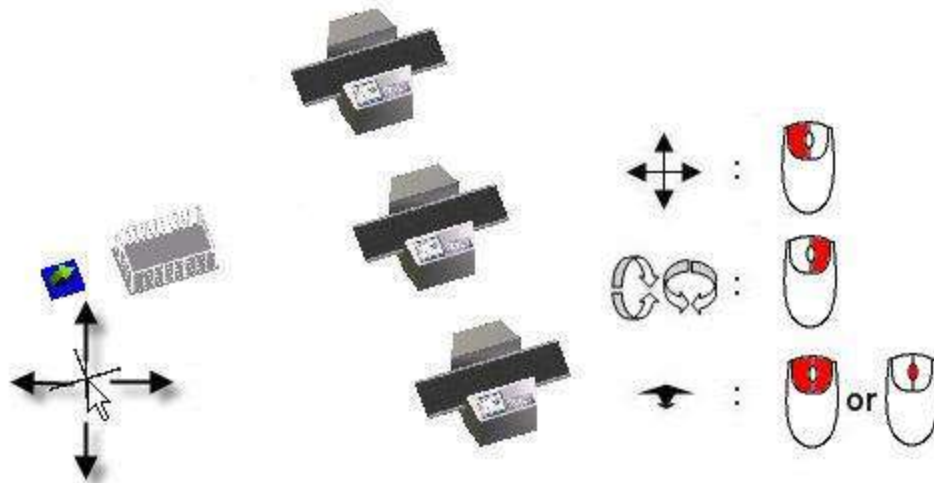


**Moving Objects** - To move an object around in the model, click on it with the left mouse button, and drag it to the position you want. You can also move the object up and down in the z direction using the mouse wheel, or by holding both the left and right mouse buttons down on the object, and dragging the mouse. To edit the object's size and rotation, select the main menu option Edit > Resize and Rotate Objects. You should see three colored arrows along each axis of the object. To resize the object, left-click on the axis you want to resize on, and drag the mouse up or down. To edit the object's rotation, right-click on the arrow corresponding to the axis you want to rotate around, and dragging the mouse up or down.



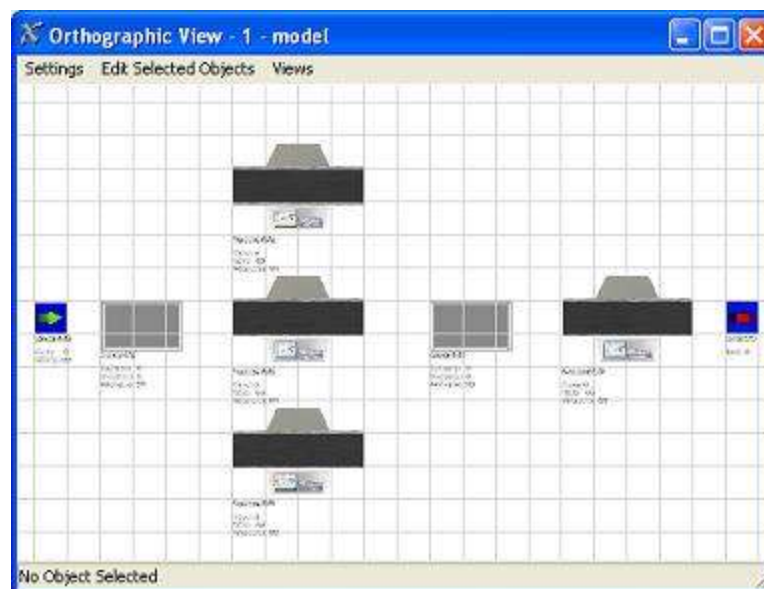


**Moving the View** - To move the model view point, click in an empty area of the view with the left mouse button, and drag the mouse around. To rotate the model view point, click in a blank area with the right mouse button and drag the mouse around. To zoom out or in, use the mouse wheel or hold both left and right mouse buttons down and drag the mouse.



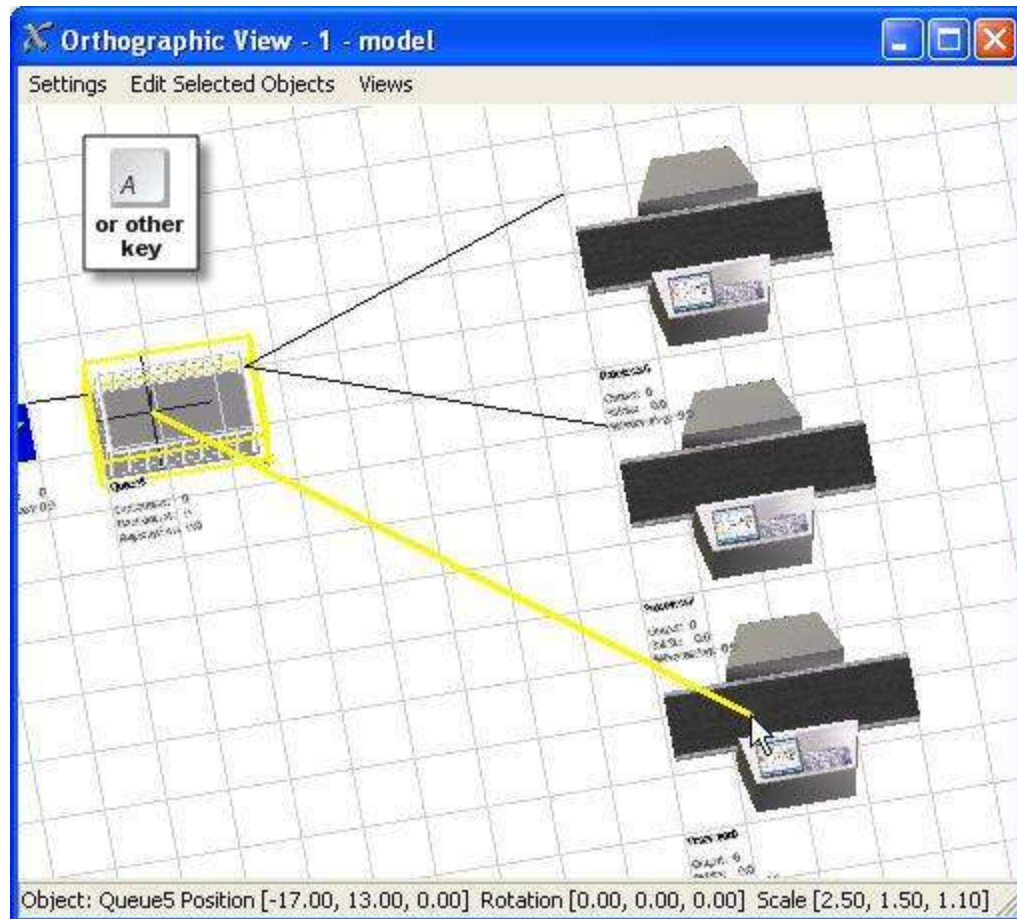
### Step 3: Finish creating objects in the model.

Drag and drop one more Queue, one more Processor, and a Sink object into the model.

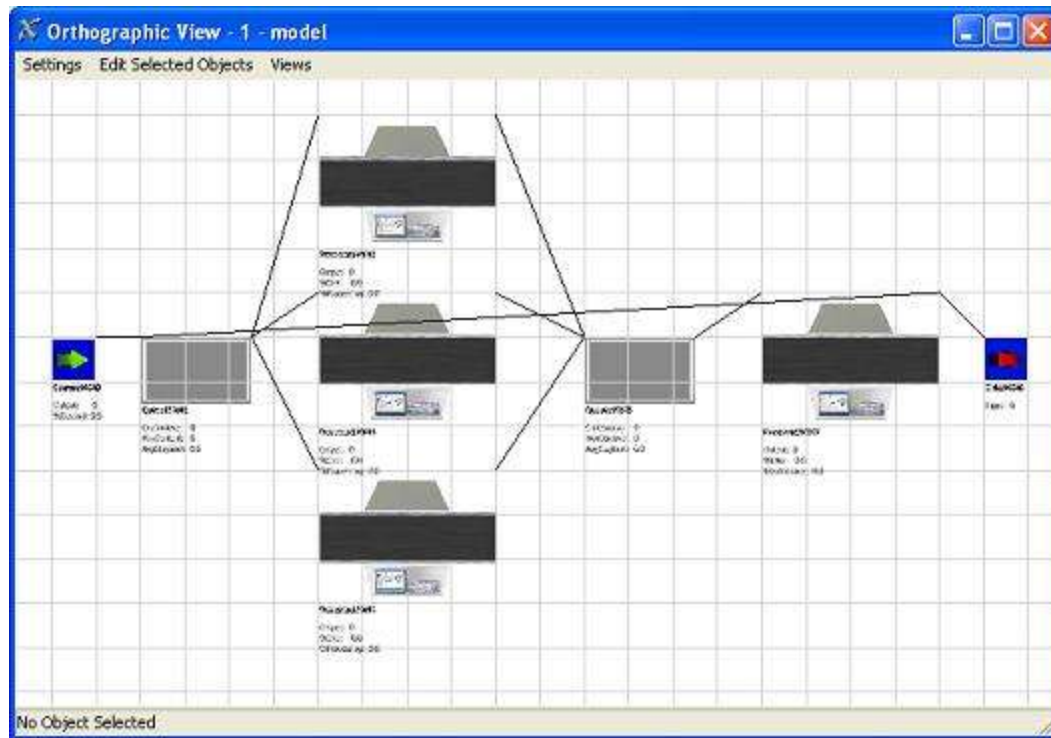


### Step 4: Connect the ports

The next step is to connect the ports for the routing of flowitems. To connect an object's output ports to the input ports of another object, press and hold the "A" key on the keyboard, then click and hold the left mouse button on the first object, then drag the mouse to the next object and release the mouse button. You should see a yellow line as you drag and a black connection line when you release.



First, connect the source to the first queue. Then connect the queue to each of the three processors. Then connect each of the three processors to the second queue. Then connect the second queue to the testing processor. Then connect the testing processor to both the sink and to the first queue at the top of the model. Connect the testing processor to the sink first, and then to the first queue. The model connections should now look like the figure below.



The next step is to change the parameters of the different objects so they will behave as specified in the model description. We will start with the source and work our way to the sink.

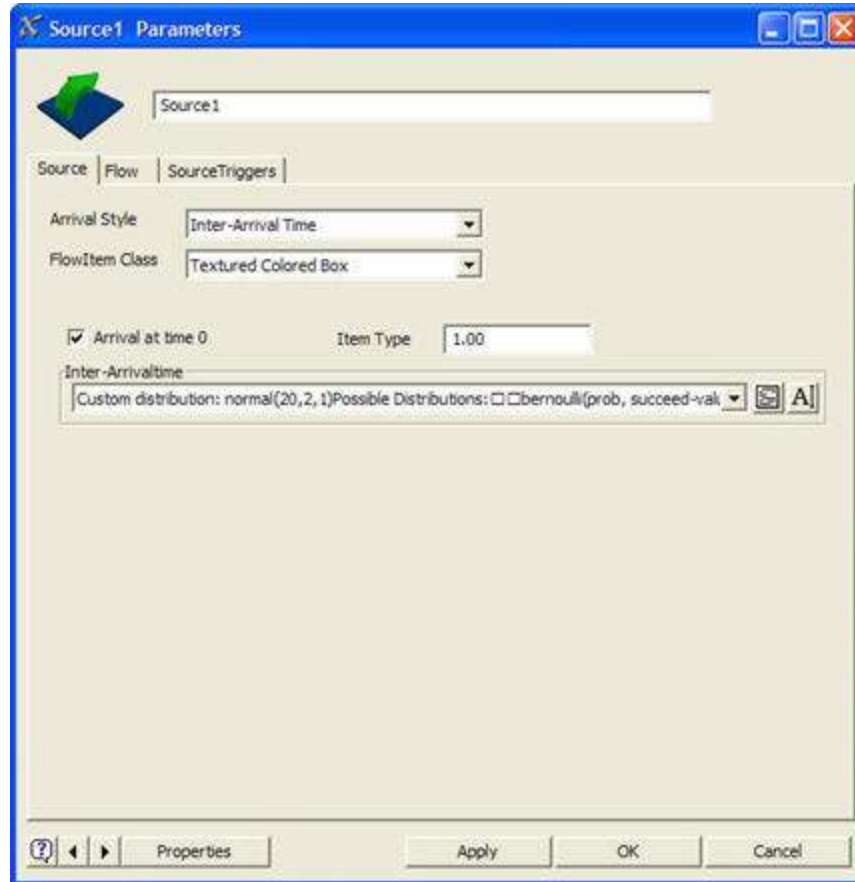
## Detailing the Model


Each object has its own parameters window through which data and logic are added to the model. Double-clicking on an object accesses the object's window.

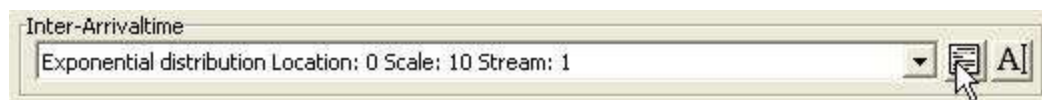
For this model, we want three different product types to enter the system. To do this, each flowitem's itemtype (see Flexsim Terminology for a description of itemtype) will be assigned an integer value between one and three using a uniform distribution. This will be accomplished using the source's exit trigger.

**Step 5: Assign the arrival rate to the source**

Double click on the source to bring up its parameters window.



All Flexsim objects have a number of pages (tabs) that present variables and information that the modeler can change based on the requirements of the model. In this model we need to change the Inter-Arrival time and the itemtype to generate 3 types of products. In this model, products arrive every 5 seconds, exponentially distributed. The source by default uses an exponentially distributed inter-arrival time, but we will change the mean of that distribution. Statistical distributions like the exponential distribution are used throughout simulation in order to model the variations that occur in real-life systems. Flexsim provides a tool called ExpertFit to help you determine which statistical distribution best matches your actual data. A detailed explanation of distributions and how to use them will be discussed later in the documentation. In the Source tab, under Inter-Arrivaltime, click on the template  button.



A pop-up window will open, explaining the option, and letting you edit parameters for the option. All text shown in blue can be changed.



Exponential distribution

Location: 0


Scale: 10

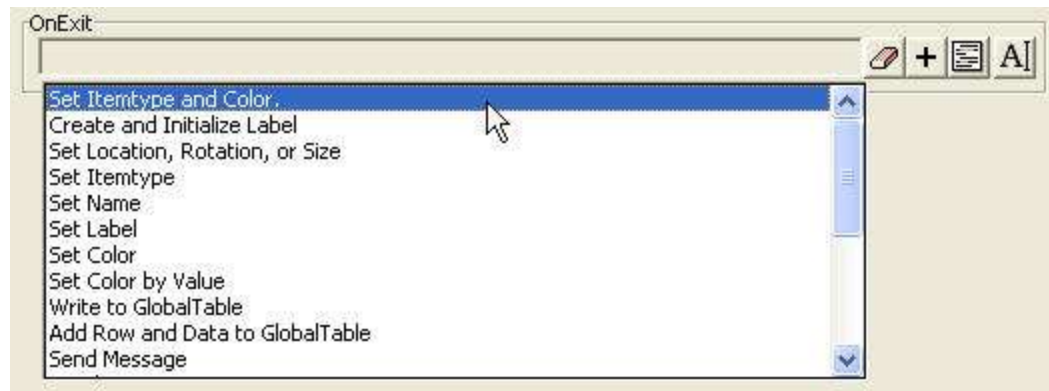
Stream: 1

Using the template you change the values to adjust the distribution or you can even insert an expression. For this model, change the scale value from 10 to 5. For an exponential distribution the scale value is the mean value. Select the OK button to return to the parameters page.

### Step 6: Assign the flowitem an itemtype and color

The next thing we need to do is assign an itemtype number to the flowitems as they enter the system. This value is uniformly distributed between 1 and 3, meaning the chance that the entering product is type 1 is just as likely as it is type 2, which is just as likely as it is type 3. The most elegant way to do this would be to change the itemtype from the OnExit trigger of the source.

Select the Source Triggers tab. Press the  button for the OnExit trigger. Select the option in the pick list titled "Set Itemtype and Color".



After selecting the option to change the flowitem itemtype and color, the following pop-up template should appear.



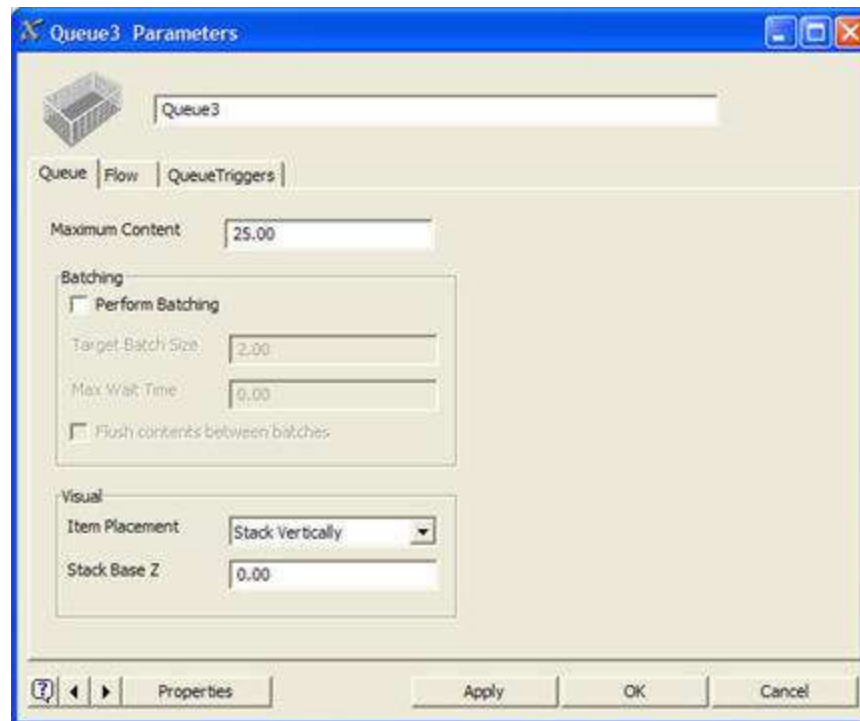
The duniform distribution is similar to a uniform distribution except that instead of returning any real number between the given parameters, only whole numbers will be returned.


We are finished editing the source's parameters because the default brown text entries are exactly what we want. Click the OK button on this window to accept and close it.

### **Step 7: Set the Queue capacity**

The next step is to detail the first Queue. There are 2 things we need to configure here. First we want to set the capacity of the Queue. Second, we want the Queue's routing to send itemtype 1 to processor 1, itemtype 2 to processor 2 and so on.

Double click on the first Queue. The Queue parameters window should appear.

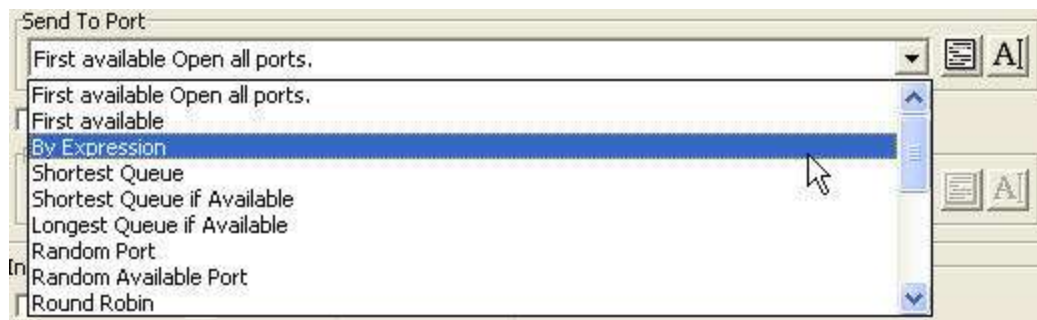


Change the Maximum Content field to 10000. This will give us a practically infinite queue size. Select the  button.

### Step 8: Assign routing for the Queue

Select the Flow tab to set the flow options for the Queue.

In the “Output” panel, under the “Send To Port” drop down pick list, select the option “By Expression”



Since we have assigned an itemtype number equal to 1, 2, or 3, we can now use the itemtype to specify the port number through which flowitems will pass. Processor 1 should be connected to port 1, processor 2 should be connected to port 2 and processor 3 should be connected to port 3.

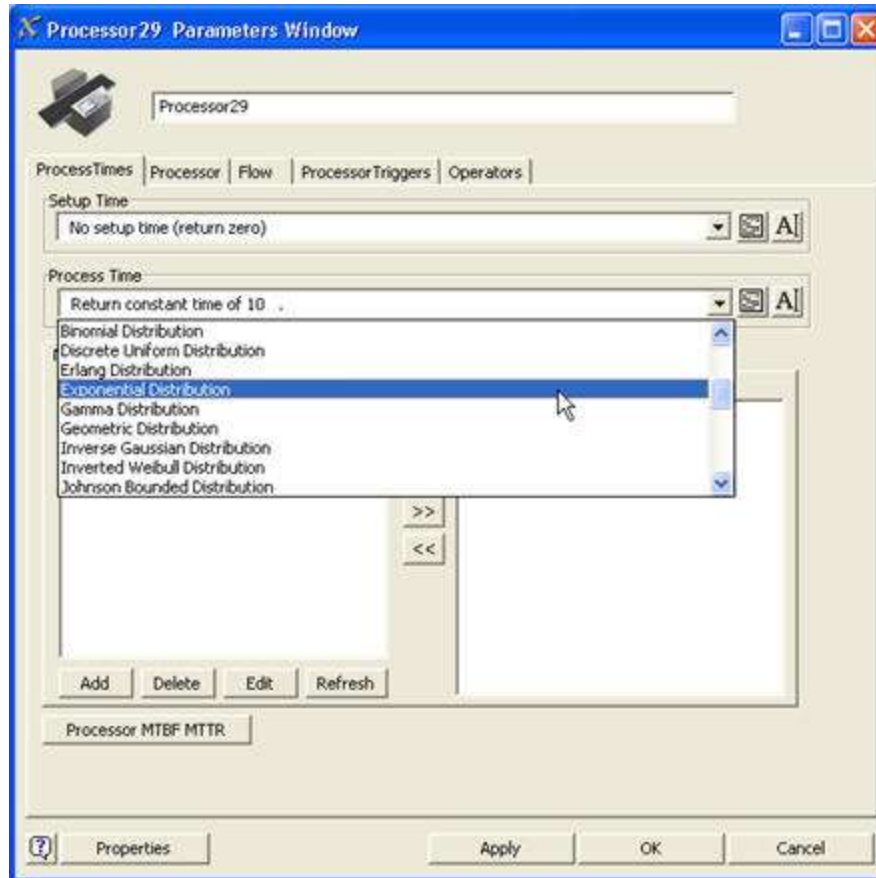


Once you have selected the “By Expression” option, notice that the default expression is "getitemtype(item)", which is exactly what we want to use. Select the OK button to close the parameters window for the queue.

## Step 9: Assign operation times to the Processors

The next step is to set the processing times for the three processors.

Double click on the first processor. The processor’s parameters window should appear.



In the “Process Time” pick list, select the option for “Exponential Distribution”. By default the scale value is set to 10 seconds. We will leave this as it is. Thus, in our model, each product will be processed for 10 seconds, exponentially distributed.

Exponential distribution

Location: 0

Scale: 10

Stream: 1



Click the OK button to close the template window. At this point this is the only change we will make to the processor. We will explore some of the other options in later lessons. Click the OK button to close the processor's parameters window.


Repeat this process for the other 2 processors.

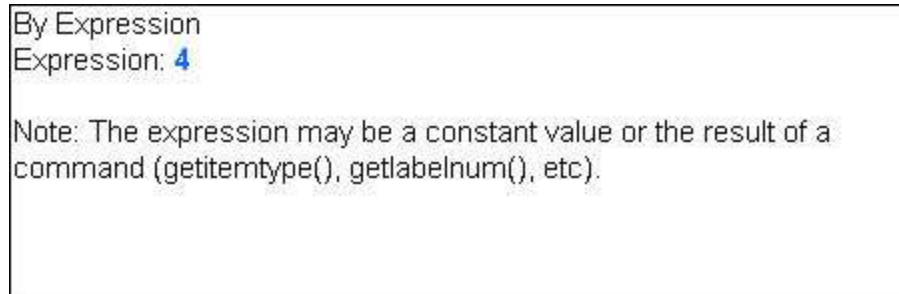
### Step 10: Detail the Second Queue

Now open the second queue's parameters window by double clicking on it. Here we want to simulate a practically infinite capacity for this queue just like we did for the first queue. Enter 10000 in the Maximum Content field. Then press the OK button to close the window.

### Step 11: Configure the Testing Station Process Time

Now we need to specify the process time and routing logic of the testing station. Double click on the testing station to open its parameters window. In the

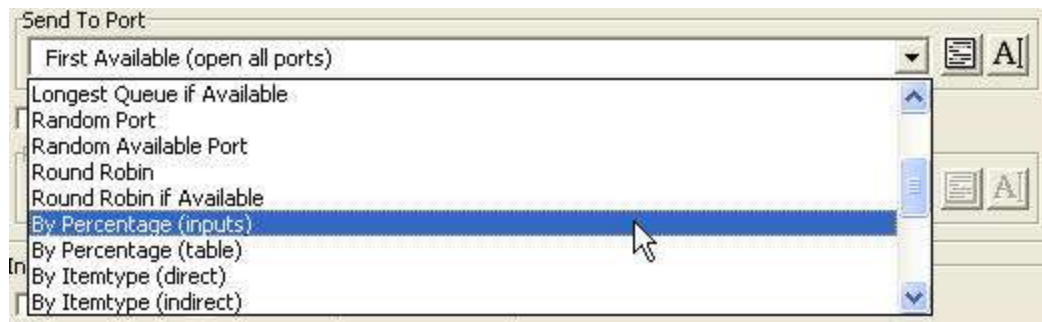
ProcessTimes tab, under Process Time, press the  button. This will again open a template pop-up explaining the current option for the process time. Change the constant time from 10 to 4. Thus, in our model, it will take a constant time of 4 seconds (no variance) to test whether a product has been manufactured correctly.



### Step 12: Configure the Testing Station Routing

Now we need to configure the testing station to send bad products back to the beginning of the model, and to send good products to the sink. When you created this object's connections, you should have first connected it to the sink, then connected it back to the first queue. This ordering will have made the first output port of testing station connect to the sink, and the second output port connect to the queue. Now we want to route to the appropriate port number based on a certain percentage.

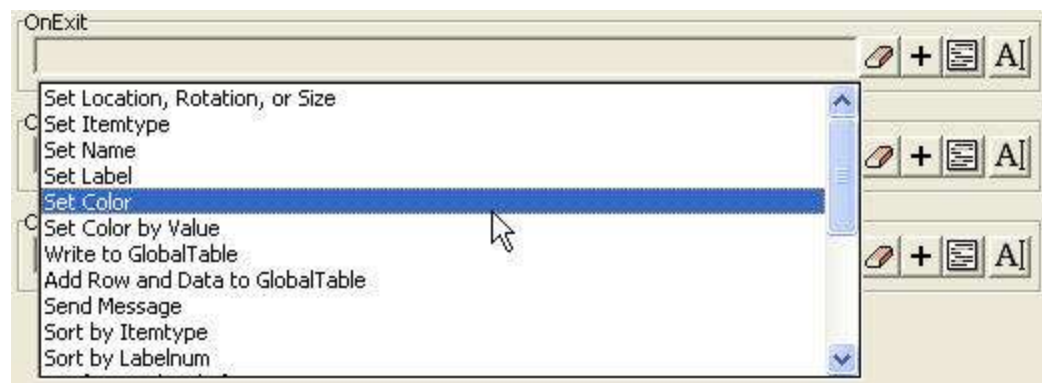
Click on the Flow tab of the testing station. In the Output panel, under the Send to Port drop down pick list, select the option: "By Percentage (inputs)".



This will again open a window explaining the chosen routing strategy. Enter 80 percent for port 1, and 20 percent for port 2. This means that 80 percent of the products, or the correctly manufactured products, will be sent out of output port 1, or to the Sink, and 20 percent, or the incorrectly manufactured products, will be sent to port 2, or back to the first queue.

By Percentage (inputs)	
Percent	Port
80	1
20	2
0	3
0	4

One more thing we might want to do is visually distinguish items that have already been through the testing station and have been sent back to the first queue. Click on the ProcessorTrigger tab of the testing station's parameters window. In the OnExit trigger, press **+** and select the option: "Set Color".



Enter colorblack as the color to change the flowitem to.


Set Color  
Color: **colorblack**  
Object: **item**

Available colors are: coloraqua, colorblack, colorblue, colorbrown, colorgray, colorgreen, colorlightblue, colorlime, colororange, colorpink, colorpurple, colorrandom, colorred, colorsilver, colorteal, colorwhite, coloryellow


Press the OK button of the testing station's parameters window to close it.

## Compiling and Running the Model

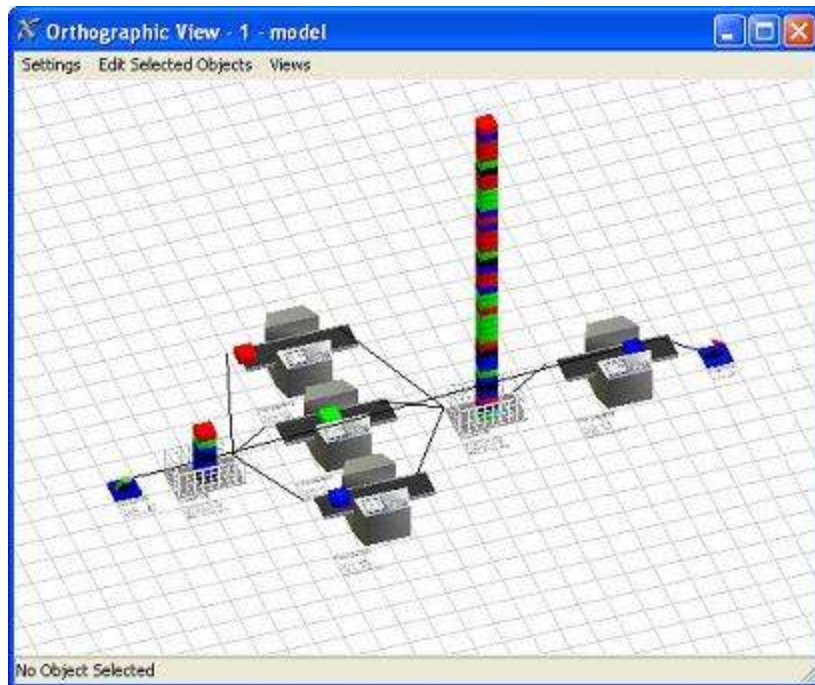
### STEP 13: Reset the Model


On the lower left of the main window click on the  button. Resetting the model ensures that all system variables are set back to their starting values, and that any flowitems present in the model are cleared out.

### STEP 14: Run the Model

Select the  button at the bottom of the main window.

The model should now start to run. Flowitems should move from the first queue, into one of the three processors, then to the second queue, into the testing station, and from there to the sink, with some being re-routed back to the first queue. Re-routed items will be colored black.



To stop the model, press the  Stop button at any time. Later you will learn how to run a model for a specified time, and for a specified number of iterations. Running a model more than once is important when statistical distributions have been used in the model definition.

To speed the model up or slow it down, move the run speed slide bar at the bottom of the window to the right or left.



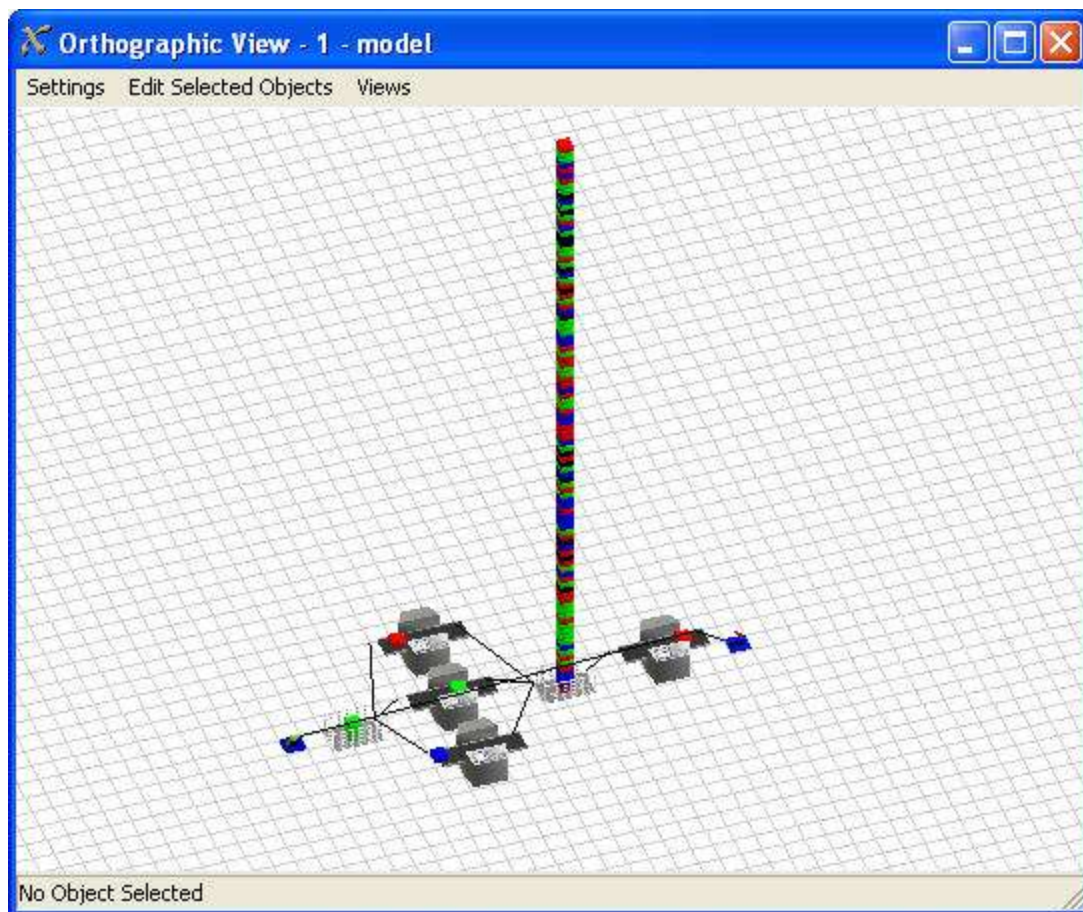
Moving the slide bar changes how fast the simulation time proceeds relative to real time. It has no effect on model results.

We have now completed building the model. Let's look at some of the statistics the model generates.

## Experimenting with the Model

### Finding the Bottleneck

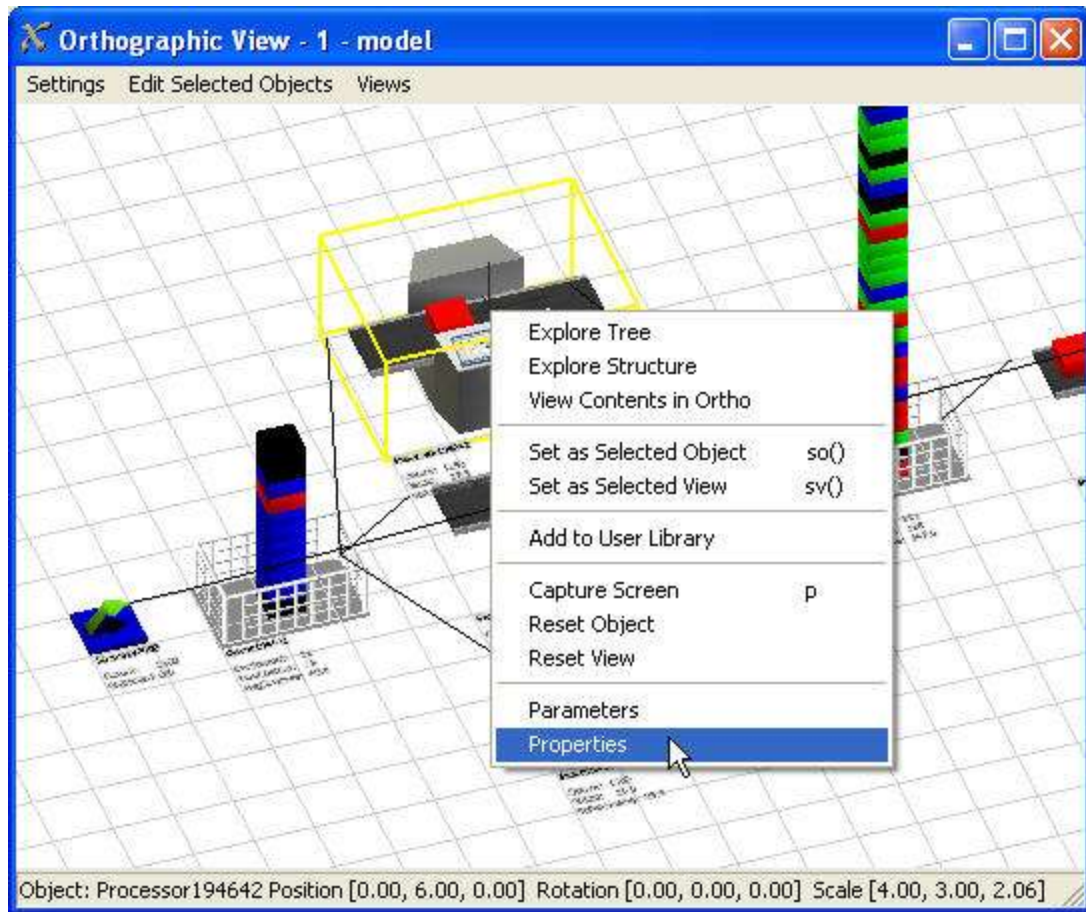
In the model description, we said that we wanted to know where the bottleneck was in the system. There are several ways to determine this. First, you can simply examine the visual size of each queue. If one queue in the model consistently has many products backed up in it, then that is a good indication that the processing station(s) that it feeds are causing a bottleneck in the system. In running this model, you'll notice that the second queue very often has a lot of products waiting to be processed, whereas the first queue's content is usually 20 or less, as shown below.



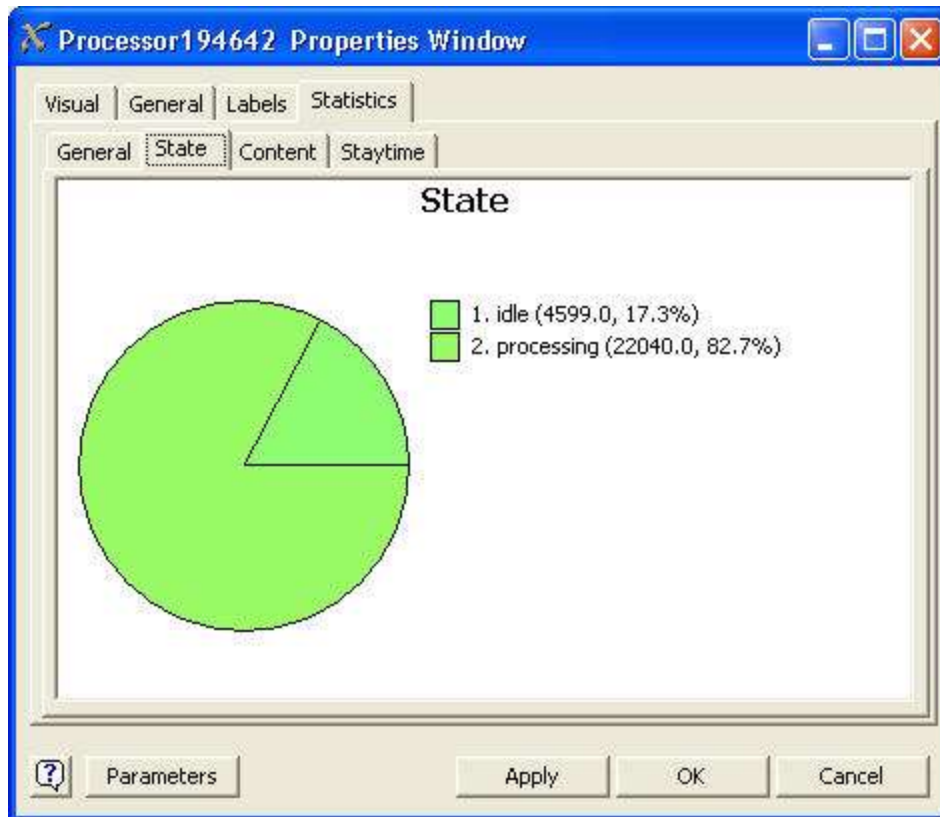
Another way of finding the location of a bottleneck is by examining the state statistics of each of the processors. If the three upstream processors are always busy, while the testing station is often idle, then the bottleneck is likely to be at the three upstream processors. On the other hand, if the testing station is always busy, while the upstream processors are often idle, then the bottleneck is probably at the testing station.



Run the model for at least 50000 seconds. Then stop the model and open the properties window of the first of the three processors by right clicking on it and selecting Properties.

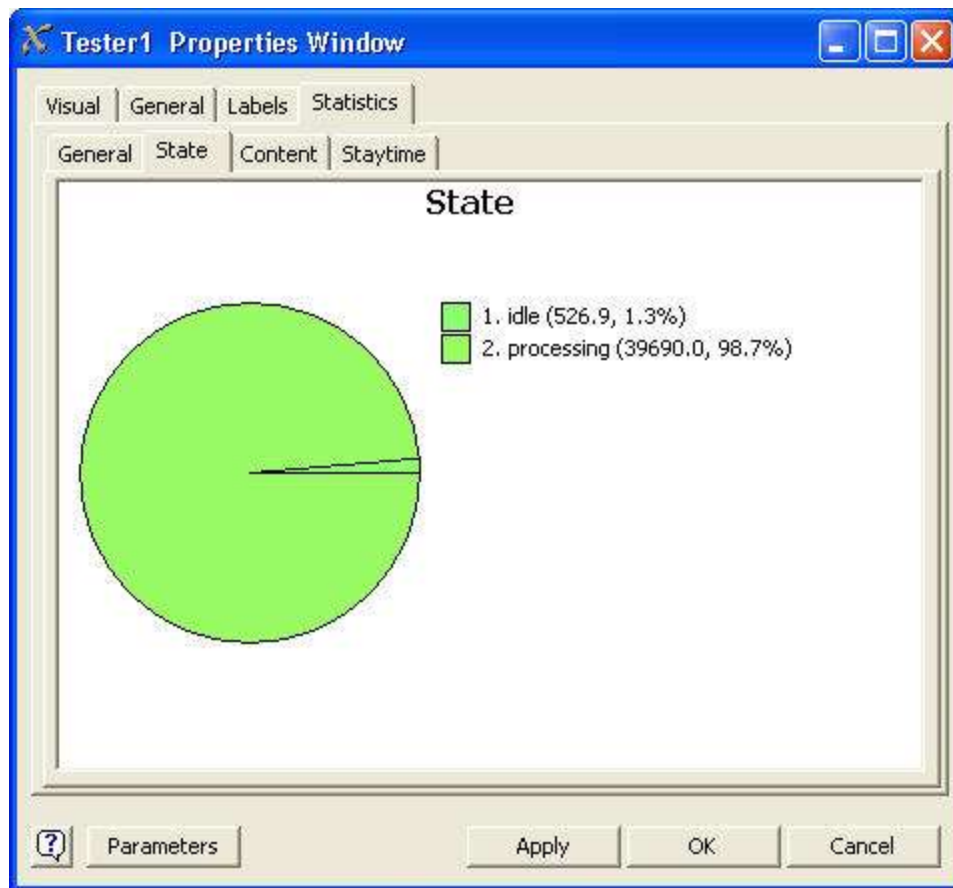


Select the Statistics tab page. Then select the State tab page. This will show a pie chart of the object's time and percentage spent in certain states.



The pie chart above shows that the processing station was idle for 17.3% of the simulation time, and that it was processing for 82.7% of the simulation time. Close this window, and then right click on each of the other two processors and go to their properties window again. They should have similar results.

Now right click on the testing station, and open its properties window. The testing station's state pie chart is shown below.



Notice that the tester was busy 98.7% of the simulation run. By these differing state diagrams, it is apparent that the bottleneck is the testing station, and not the three processing stations.

Now that we have figured out where the bottleneck is, the question is, what should we do about it? This depends on several factors of cost versus gain, as well as on the future goals of the facility. In the future, will the facility need to be able to push more products through at a faster rate? In our model, the source produces one product every five seconds on average, while the tester sends a product to the sink about once every 5 seconds on average. This average of 5 seconds for the tester can be calculated using the tester's 4 second cycle-time and its 80/20 send-to strategy. Thus, over time, our model's total capacity levels off. If the factory started pushing more products through this part of the facility, this equates to a higher arrival rate (a lower inter-arrival time) for the source. If we then make no changes to the tester, our model would continually accumulate more and more parts, and the content of the queues would continue to increase until there was no room left. To fix this, we would have to add a second tester station, since it is the model's bottleneck.

Another situation in which we might want to add another testing station is if the queue size of the tester's queue is very important to us. If it is costly to allow the



tester queue size to accumulate to high amounts, then it would be smart to put a second tester station in, to make sure that the queue size, as well as each product's wait time in that queue, doesn't get too high. Let's look at the queue statistics.

Right click on the tester queue, and select Properties. Click on the Statistics tab, and view the General page. It should look something like the page below.

**Queue194645 Properties Window**

Visual | General | Labels | Statistics

General | State | Content | Staytime

**Content**

Current:	102.000
Minimum:	0.000
Maximum:	162.000
Average:	60.197

Chart...

**Staytime**

Minimum:	0.000
Maximum:	646.212
Average:	241.503

Chart...

**State**

Current:	8.000
----------	-------

Chart...

**Throughput**

Input:	30036.000
Output:	29934.000

**Settings**

Limit Content History Size to:	100.000	points
Staytime Histo Lower Bound:	0.000	
Staytime Histo Upper Bound:	100.000	
Staytime Histo Divisions:	20.000	
<input type="checkbox"/> Display Confidence Interval	95.000	percent

? Parameters Apply OK Cancel

Continue to run the model, and you'll notice these values change as the simulation runs. Look at the average content and average staytime values. Staytime refers to the amount of time flowitems resided in the queue. Early on in the simulation, the queue's average content is usually low, but as the simulation continues, it may reach high values like 200 or 300. If an average queue size of 200 or 300 is unacceptable, then it may be necessary to add a second tester.

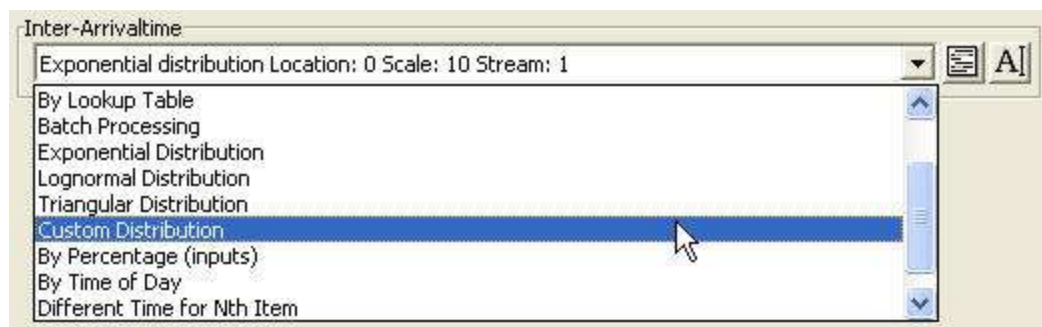
## Randomness

Let's do some more testing before we actually decide to add another tester. Since on average one product arrives from the source every 5 seconds, and on average one product goes to the sink every 5 seconds, why should the queue accumulate at all? Products are leaving just as fast as they arrive, so there shouldn't be any accumulation in the system.

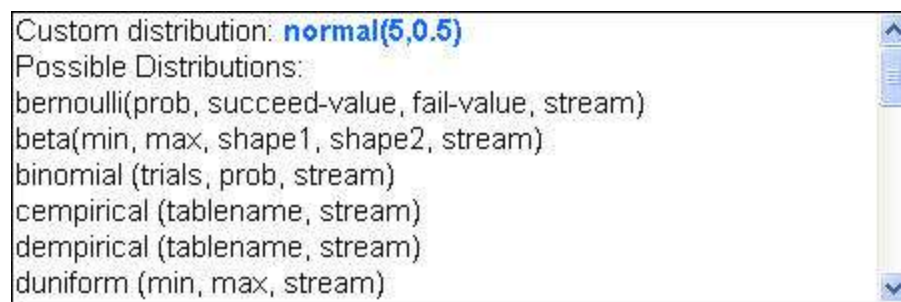
The reason the queue accumulates is because of randomness in the system. Yes, on average a product arrives every 5 seconds, but this arrival rate is according to an exponential distribution. For an exponential distribution with a mean of 5, most of the time products will actually arrive at a faster rate than every 5 seconds. But every once in a while there will be a long drought where no products arrive at all. In the end it evens out to an average of 5 seconds, but usually products arrive faster, and thus will accumulate in the tester's queue, since the tester is the bottleneck.

What if, in our facility, products actually arrive at a more predictable rate, instead of by the somewhat unpredictable exponential distribution? Will the queue size generally stay at a lower level? Let's test it.

Double click on the Source to open its parameters window. Under the inter-arrival time drop down pick list, select the "Custom Distribution" option.



In the blue text, enter the following expression: `normal(5,0.5)`. This specifies a normal distribution with mean of 5 and standard deviation of 0.5.




Press OK in the parameters window. Reset and run the model again.

If you do not still have the tester queue's properties window available, open it again by right clicking on the queue and selecting Properties. Continue to run the model. You will notice here that the queue's average content and average staytime don't go up as high. Usually they won't go much higher than 50 or 60 now, whereas before they would sometimes get up to 200 or 300. This is a significant improvement, and it was effected by only changing the type of randomness in the model.

### Higher Throughput

Now what if the facility does indeed need to increase the throughput rate of this system by 15%. This equates to a change of the mean inter-arrival time of the source from 5 seconds to 4.25 seconds. Since the tester was already at 100% utilization, we will obviously need to add a second tester to the system. Let's make this change.

Double click on the source again to bring up its parameters window. Click on the

template  button for the Inter-Arrivaltime. Change the mean of the normal distribution from 5 to 4.25. Click OK in the template window. Click OK in the source's parameters window.

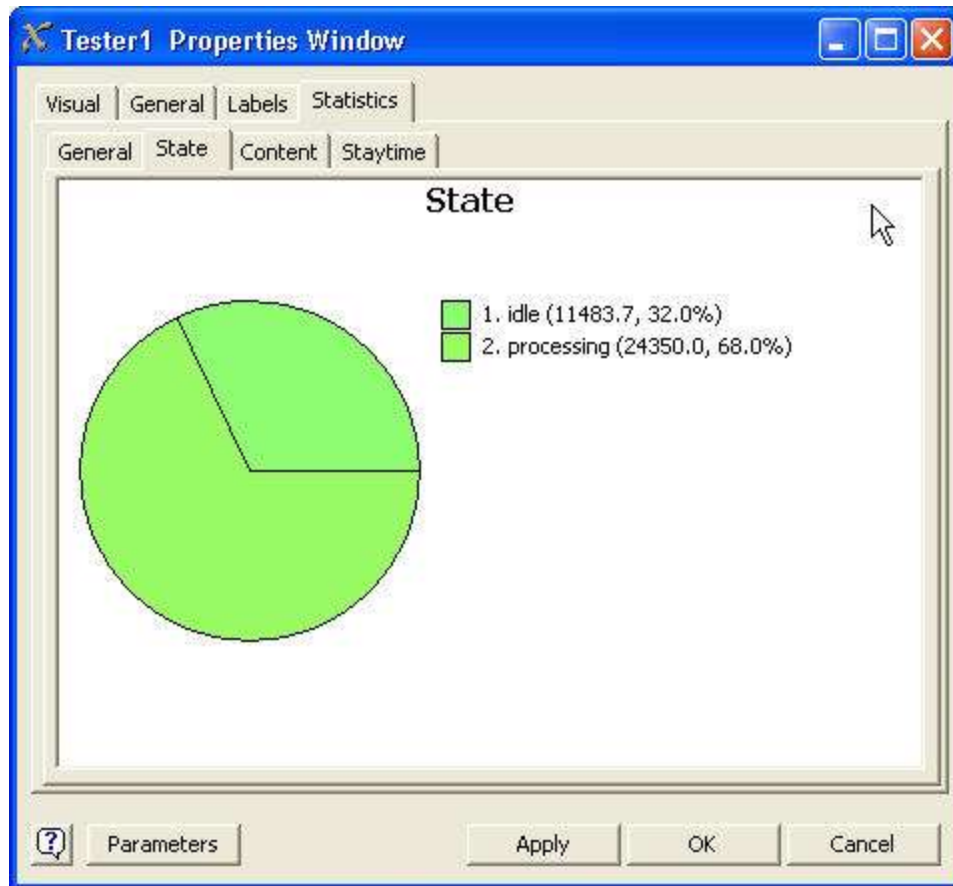
Now we will create a second tester. Drag another Processor object into the model, below the first tester. Double click on it and set its Process Time to a constant of 4 seconds, just like the original tester. Then connect the tester queue's output ports to the new tester using the 'A' drag connection, and then connect the new tester's output ports first to the sink, and then to the upstream queue.

In the new tester's flow page, select the By Percentage (inputs) option in the send to port drop-down box. Enter 80% to port 1 and 20% to port 2.

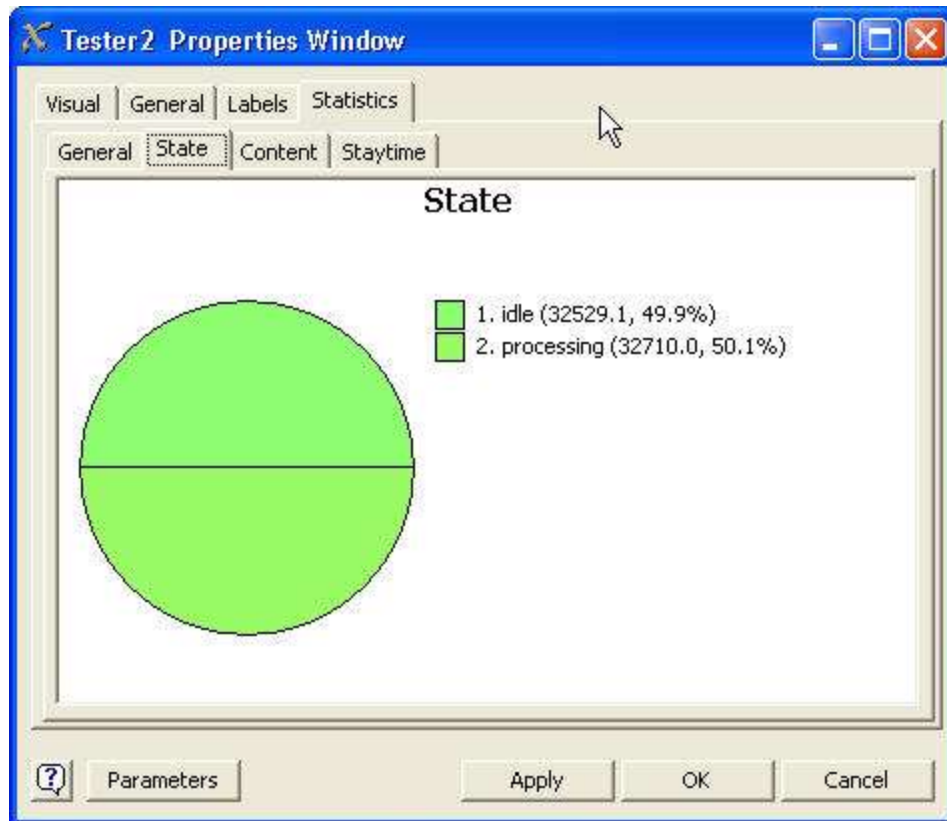
Now that we have finished making the changes, we can reset and run the model again.

### Evaluating the New Configuration

Now run the model for at least 50,000 seconds. Notice first that the tester queue is now almost always empty, whereas the queue for the 3 processors backs up quite often. Let's look at the original tester's pie chart now. Right click on the original tester and select Properties. Go to the Statistics tab, and then to the State tab. This should show that the original tester is now busy only about 68% of the time, now that there is a second tester.

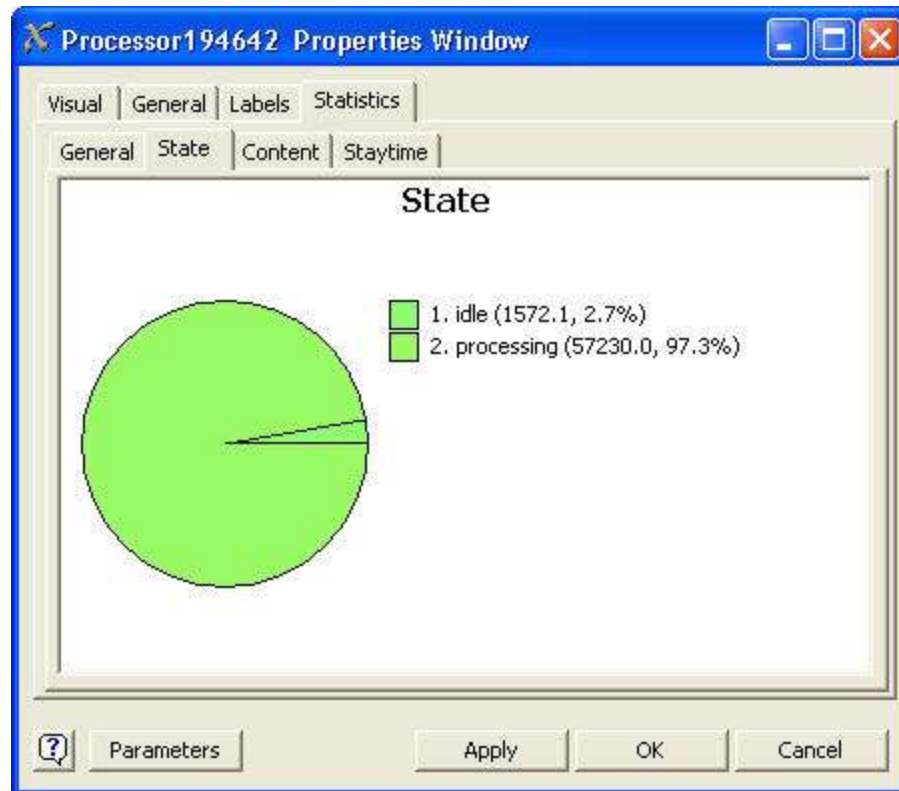


Click OK to close the Properties window. Now click on the new tester and go to its state pie chart. It is busy only 50% of the time, less than the original tester.



The reason that these two are different is because the tester queue sends to the first available tester. Whenever both testers are available, a product will always go to the original tester, since it is the first available. Products only go to the second tester if the original tester is already busy. Thus the original tester gets higher utilization than the second tester.

Click OK in the second tester's properties window. Now go to one of the three upstream processors, right click on it and select Properties. Again, go to its state pie chart. Notice that the processors' utilization has now gone up to almost 100%.



We have effectively moved the system bottleneck from the tester to the three upstream processors. Also, by increasing throughput by 15%, and consequently adding another tester, we have significantly decreased the utilization of each tester. Whether this is a good decision depends much on the cost it would take to add a second tester. Now, since the bottleneck is in the 3 processors, in order to further increase throughput, and thus increase the utilization of each tester, we would need to add more processors. Again, there is a cost/benefit analysis to this decision.

## Results

By creating a model that simulates our system, we have clearly determined what effect certain decisions will have on the system. We can now use the information we have gathered from the simulation to make better informed decisions for the future of the factory.

With this simple model, many of the same conclusions could have been made through mathematical models and formulas. However, real systems are often much more complex than the model we have just built, and are outside the scope of mathematical modeling. By using Flexsim simulation, we can model these real-life complexities, and examine the results just as we have done in this model.

Flexsim also gives your simulations much more visual appeal. It is much easier to convince a management team of the wisdom in a decision if the management team

can see the effects of that decision in a virtual reality 3D world. This world is created automatically as you build your Flexsim models.

### **What Next?**

Now that you have become familiar with Flexsim and the use of simulation, we suggest that you go through the other tutorials included in Flexsim Help.





# Tutorials

This basic tutorial will take you through the steps of setting up a process flow, building a model, inputting data, viewing the animation, and analyzing output. Each lesson will build upon the previous one. It is therefore important to thoroughly understand each lesson before moving on to the next one. You should plan on at least 45 minutes to complete each lesson. Lesson 2 will also include an “Extra Mile” section at the end that will add additional value to your model. The following lessons are contained in this tutorial:

**Lesson 1:** Building a simple model that will process 3 different flowitem types. Each itemtype will have a specific routing. Objects used in this model will be the Source, Queue, Processor, Conveyor, and Sink. The basic statistics of model performance will be introduced, and the parameter options for each object will be explained.

**Lesson 2:** Using the model from lesson 1, you will add Operators and Transporters to the process. Object properties will be introduced, and additional statistical analysis concepts will be discussed.

**Lesson 2 Extra Mile:** After you have completed lesson 2 you will be shown how to add 3D charts and graphs to the model using the Recorder object. 3D visual text will also be added using the VisualTool object to give annotation to the model.

**Lesson 3:** Using the model from lesson 2, you will add rack storage and network paths. Advanced statistics and model logic will be added, as will global tables used for reading and writing data.

Each lesson will have the following format:

1. Introduction
2. What you will learn
3. Approximate completion time
4. Model description
5. Model data
6. Flexsim software concept learning
7. Step-by-step model construction

Feel free to contact our technical support group if you have any questions while working on this tutorial. Flexsim technical support can be reached at 801-224-6914, or e-mail your question to [support@flexsim.com](mailto:support@flexsim.com). We hope you enjoy learning how Flexsim can help you optimize your flow process.

## Lesson 1

### Introduction

Lesson 1 introduces the basic concepts of diagramming and building a simple model. Building a diagram of the process is a great way to start every model that you will build in Flexsim. If you can not build a diagram, flowchart, or at least see a picture in your mind of how the process works, you will have a difficult time building the model in Flexsim.

**Note:** if you have already gone through the Getting Started tutorial, many of the concepts you learn in this lesson will not be new. However, subsequent lessons build upon this lesson, so it is probably a good idea to go through it anyway.

### What You Will Learn

- How to build a simple layout
- How to connect ports for routing flowitems
- How to detail and enter data into Flexsim objects
- How to navigate in the animation views
- How to view simple statistics on each Flexsim object

### New Objects

In this lesson you will be introduced to the Source, Queue, Processor, Conveyor, and Sink objects.

### Approximate Time to Complete this Lesson

This lesson should take about 30-45 minutes to complete.

## Flexsim Software Concept Learning

### Flexsim Terminology

Before you start this model it will be helpful to understand some of the basic terminology of the software.

**Flexsim objects:** Flexsim objects simulate different types of resources in the simulation. An example is the Queue object, which acts as a storage or buffer area. The Queue can represent a line of people, a queue of idle processes on a CPU, a storage area on the floor of a factory, or a queue of waiting calls at a customer service center. Another example of a Flexsim object is the Processor object, which simulates a delay or processing time. This object can represent a machine in a factory, a bank teller servicing a customer, a mail employee sorting packages, an epoxy curing time, etc.

Flexsim objects are found in the Object Library grid panel.

**Flowitems:** Flowitems are the objects that move through your model. Flowitems can represent parts, pallets, assemblies, paper, containers, telephone calls, orders, or anything that moves through the process you are simulating. Flowitems can have processes performed on them and can be carried through the model by material handling resources. In Flexsim, flowitems are generated by a Source object. Once flowitems have passed through the model, they are sent to a Sink object.

**Itemtype:** The itemtype is a label that is placed on the flowitem that could represent a barcode number, product type, or part number. Flexsim is set up to use the itemtype as a reference in routing flowitems.

**Ports:** Every Flexsim object has an unlimited number of ports through which they communicate with other objects. There are three types of ports: input, output, and central.

Input and output ports are used in the routing of flowitems. For example, a mail sorter places packages on one of several conveyors depending on the destination of the package. To simulate this in Flexsim, you would connect the output ports of a Processor object to the input ports of several Conveyor objects, meaning once the Processor (or mail sorter) has finished processing the flowitem (or package), it sends it to a specific conveyor through one of its output ports.

Central ports are used to create references from one object to another. A common use for central ports is for referencing mobile objects such as operators, fork lifts, and cranes from fixed resources such as machines, queues, or conveyors.

Ports are created and connected by clicking on one object and dragging to a second object while holding down different letters on the keyboard. If the letter "A" is held down while clicking-and-dragging, an output port will be created on the first object and an input port will be created on the second object. These two new ports will then be automatically connected. Holding down the "S" key will create a central port on both objects and connect the two new ports. Connections are broken and ports

deleted by holding down the “Q” for input and output ports and the “W” key for central ports. The following table shows the keyboard letters used to make and break the two types of port connections. Lesson 1 of the tutorial demonstrates how to properly make port connections.

	<b>Output - Input</b>	<b>Center</b>
<b>Disconnect</b>	Q	W
<b>Connect</b>	A	S

**Model views:** Flexsim uses a three-dimensional modeling environment. The default model view for building models is called an orthographic view. You can also view the model in a more realistic perspective view. It is generally easier to build the model's layout in the orthographic view, whereas the perspective view is more for presentation purposes. However, you may use any view option to build or run the model. You may open as many view windows as you want in Flexsim. Just remember that as more view windows are opened the demand on computer resources increases.

---

## Model 1 Description

In our first model we will look at the process of testing three products coming off a manufacturing line. There are three different flowitem itemtypes that will arrive based on a normal distribution. Itemtypes will be uniformly distributed between itemtypes 1, 2, and 3. As flowitems arrive they will be placed in a queue and wait to be tested. Three testers will be available for testing. One tester will be used for itemtype 1, another for itemtype 2, and the third for itemtype 3. Once the flowitem is tested it will be placed on a conveyor. At the end of the conveyor the flowitem will be sent to a sink where it will exit the model. Figure 1-1 shows a diagram of the process.

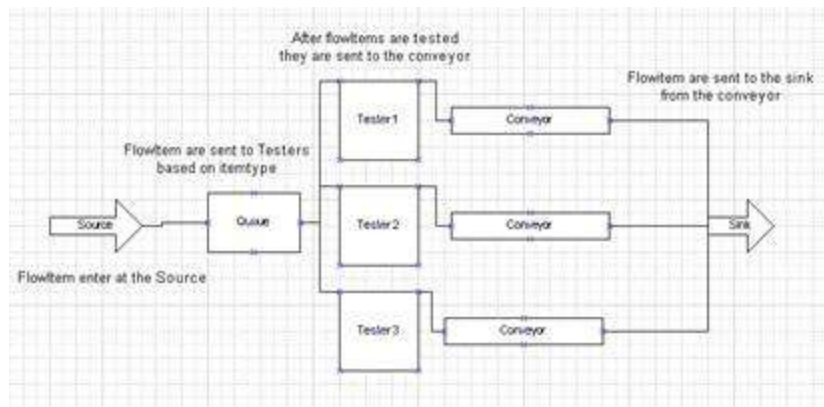


Figure 1-1. Model 1 Diagram

## Model 1 Data

**Source arrival rate:** normal(20,2) seconds

**Queue maximum size:** 25 flowItems

**Testing time:** exponential(0,30) seconds

**Conveyor speed:** 1 meter per second

**Flowitem routing:** Itemtype 1 to Tester 1, Itemtype 2 to Tester 2, Itemtype 3 to Tester 3.

## Step-By-Step Model Construction

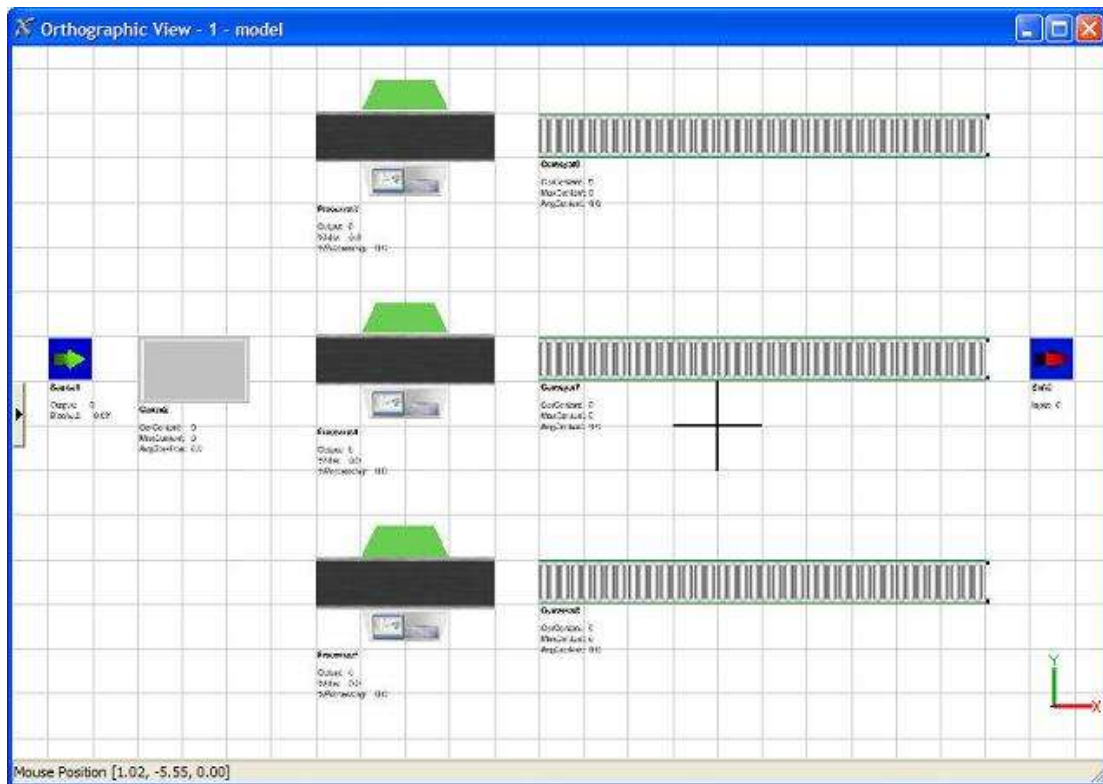
### Building Your First Model

Open the application by double clicking on the Flexsim icon on your desktop. Once the software loads, you should see the Flexsim menu and toolbars, Library, and Orthographic Model View windows.

**Step 1: Drag and drop a Source from the Library into the Orthographic View window.**

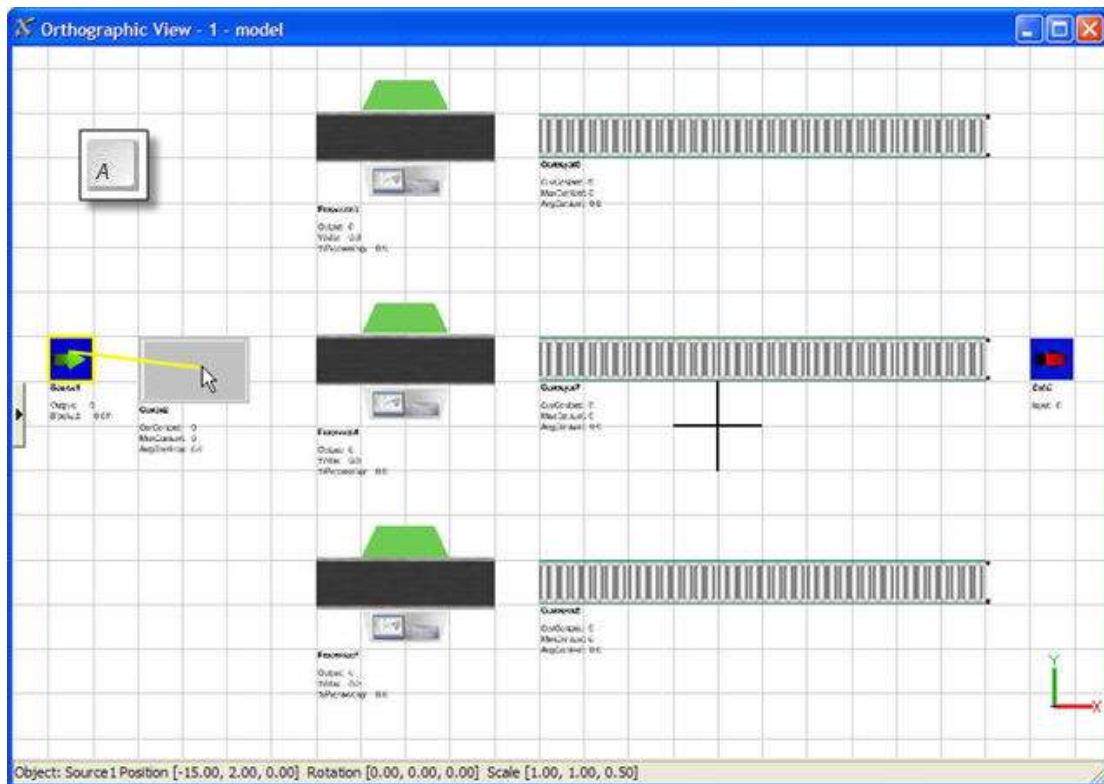


**Step 2: Drag and drop the remaining objects into the Orthographic View window.**

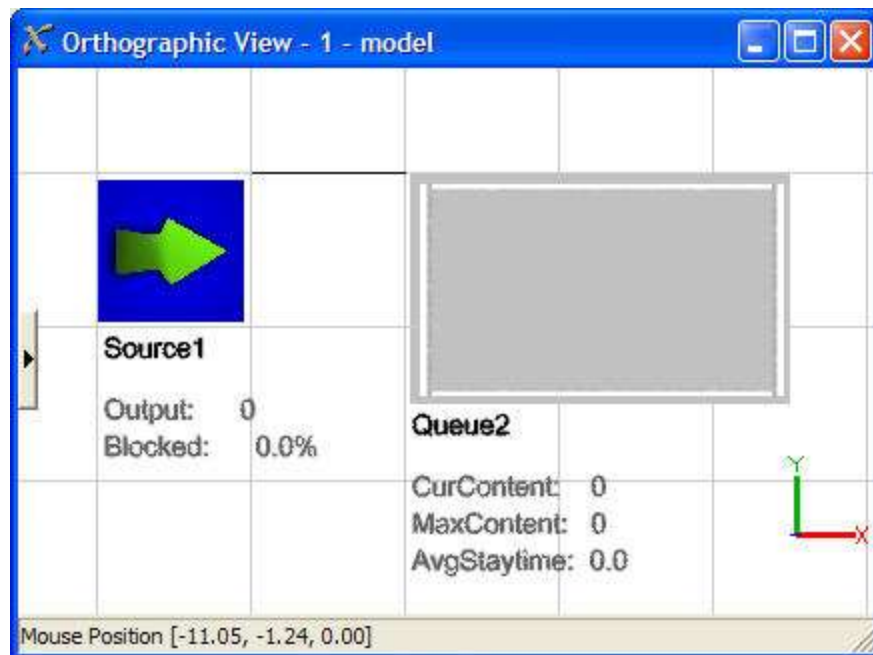


### Step 3: Connect the ports

The next step is to connect the ports for the routing of the flowitems. This is accomplished by pressing and holding the "A" key on the keyboard, then click and hold the left mouse button on the source, then drag the mouse to the queue, and then release the mouse button. You should see a yellow line as you drag.

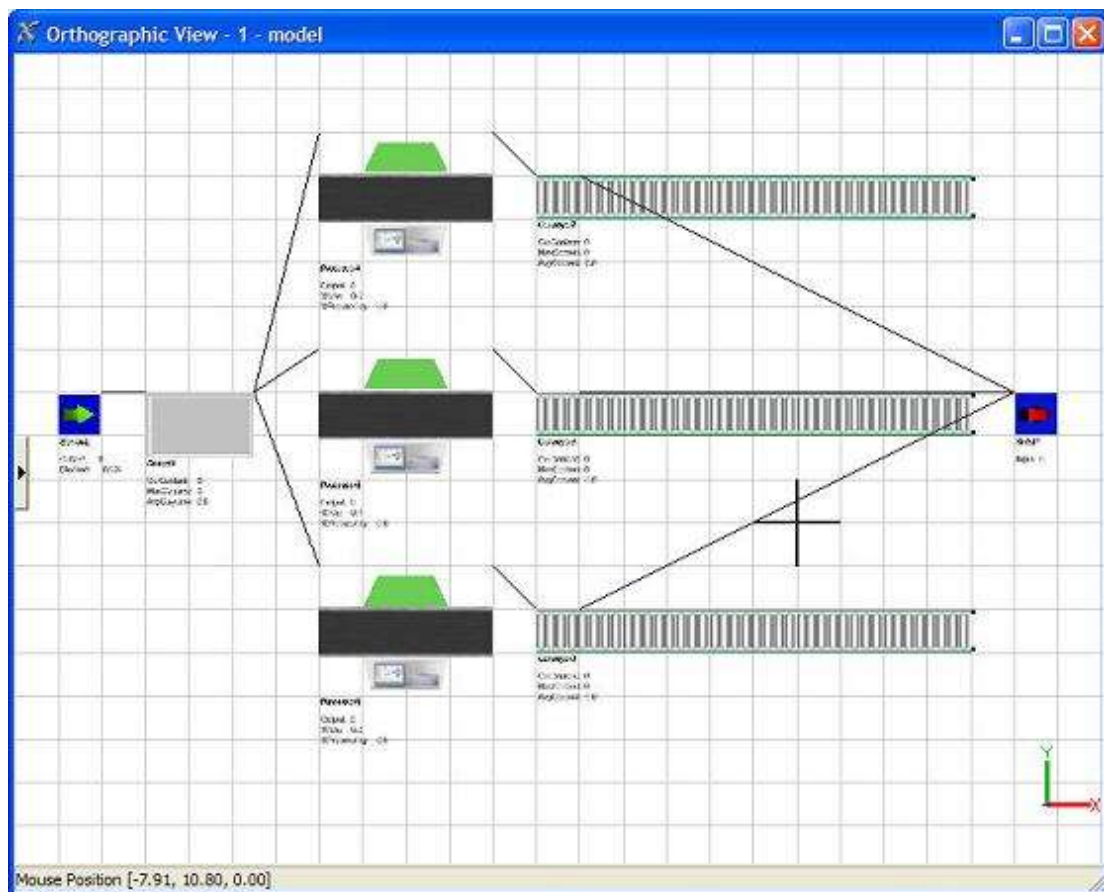


A black connection should appear once you've released the mouse button.



Finish the connections by connecting the queue to each of the processors, each processor to the corresponding conveyor, and each conveyor to the sink. When you are finished, the layout should look like the following picture.





The next step will be to change the parameters of the different objects so they will behave the way you want. We will start with the source and work our way to the sink.

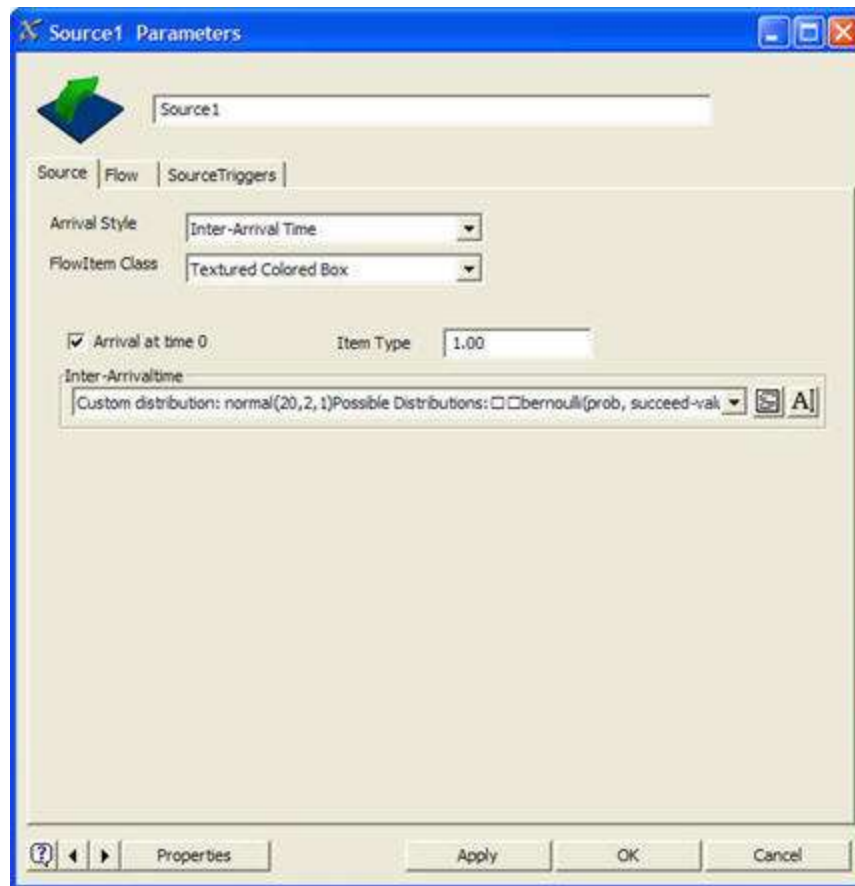
### Detailing the Model

Each object has its own graphical user interface (GUI) through which data and logic are added to the model. Double-clicking on an object accesses that object's GUI called the parameters window.

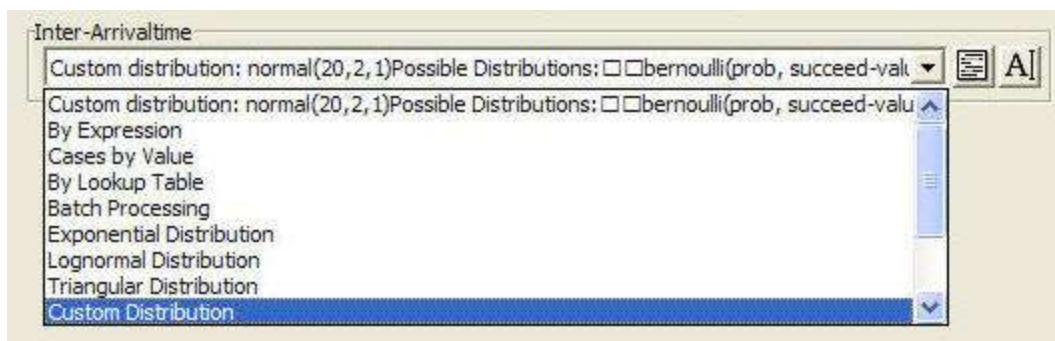
For this model, we want three different product types to enter the system. To do this, each flowitem's itemtype will be assigned an integer value between one and three using a uniform distribution. This will be accomplished using the Source's exit trigger (OnExit).

#### Step 4: Assigning the arrival rate

Double click on the source to bring up its parameters window



All Flexsim objects have a number of pages or tabs that present variables and information that the modeler can change based on the requirements of the model. In this model we need to change the Inter-Arrival time and the itemtype to generate 3 types of items. To set the Inter-Arrival time to normal(10,2) as stated in the model description, select the down arrow on the Inter-Arrival time pick list and select the "Custom Distribution" option.




Once you have selected the "Custom Distribution" option, you will see this window:

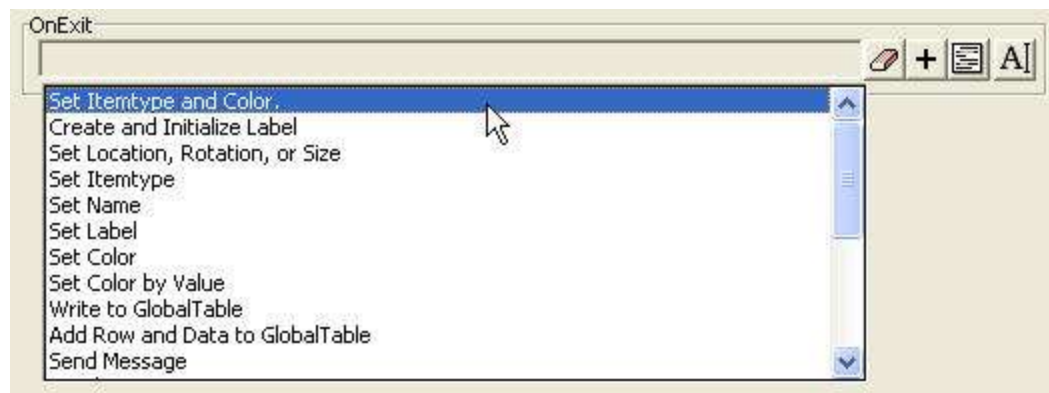


Using the template, you can change the blue text to adjust the distribution. For this model, change the blue text to: `normal(20,2)`. This specifies an inter-arrival time with mean of 20 seconds and standard deviation of 2 seconds. Now click off of the pop-up window to hide it.

The next thing we need to do is assign an itemtype number to the flowitems as they enter the system that is uniformly distributed between 1 and 3. The most elegant way to do this would be to change the itemtype on the OnExit trigger of the Source.

### Step 5: Set Itemtype and Color

Select the Source Triggers tab. In the OnExit trigger, click on the  button and select the option to change the flowitem's itemtype and color named "Set Itemtype and Color."



After selecting the option to change the flowitem itemtype and color, a pop-up window will appear:

Set Itemtype and Color  
Flowitem: **item**  
Itemtype: **duniform(1,3)**


The duniform distribution is similar to a uniform distribution except that instead of returning any real number between the given parameters, only discrete integer values will be returned.

Click the OK button for this window and for the source's parameters window.

The next step will be to detail the queue. Since the queue is a place to hold flowitems until they can be processed at the processor, there are 2 things we need to do. First, we need to set the capacity of the queue to hold 25 flowitems. Second, set the flow options to send itemtype 1 to processor 1, itemtype 2 to processor 2 and so on.

#### Step 6: Setting the Queue capacity

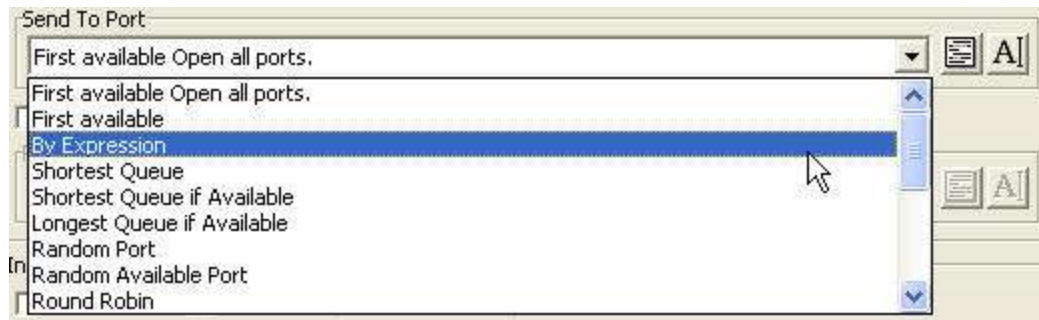
Double click on the queue. The queue parameters window should appear.

Change the Maximum Content field to 25. Select the  button.

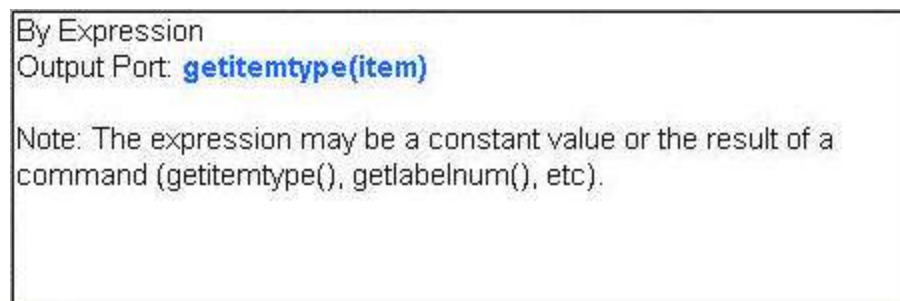
### Step 7: Assign the Flow options for the Queue

Select the Flow tab in the Parameters Window to set the flow options for the queue.

On the "Send To Port" pick list select the option "By Expression"



The following pop-up window should appear.



Since we have assigned an itemtype number equal to 1, 2, or 3, we can now use the itemtype to specify the port number through which flowitems will pass. Notice that the default output port is: `getitemtype(item)`. Leave this as it is. Processor 1 should be connected to port 1, processor 2 should be connected to port 2 and so on.

Once you have selected the "By Expression" option select the OK button to close the parameters window for the queue.

The next step is to set the processor times.

### Step 8: Assigning operation times to the Processors

Double click on processor number 1. The processor's parameters window should appear (Figure 1-14).

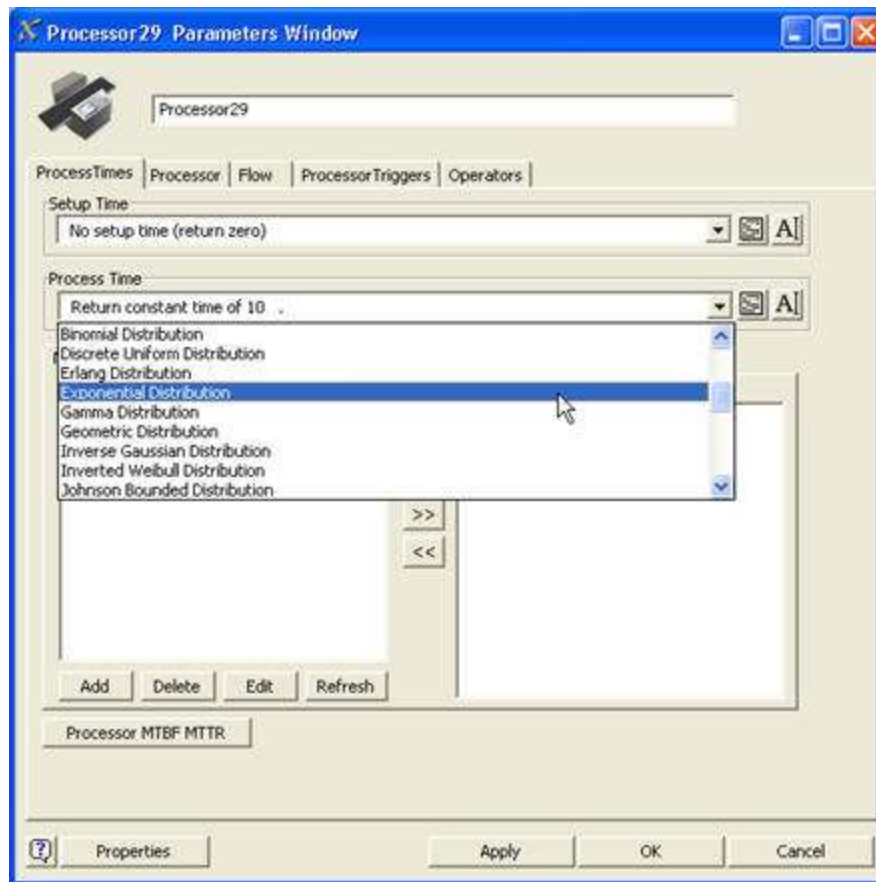


Figure 1-14

In the "Process Time" pick list, select the option for "Exponential Distribution". By default the distribution's scale value is set to 10 seconds so we will need to change this in the template pop-up window (see figure 1-15).

Exponential distribution

Location: 0

Scale: 30

Stream: 1

Figure 1-15


Change the scale value to 30. The scale value of an exponential distribution happens to be its mean. Click the OK button to close the window. This is the only change we will make to the processor at this time. We will explore some of the other options in later lessons. Click the OK button to close the processor parameters window.

Repeat this for the other 2 processors.

The default speed for the conveyors is already set to 1 length unit per time unit so here is no need to modify the conveyors at this time.

Now we are ready to run the model.

### Step 9: Reset the model

Always click the  button at the bottom of the main window to reset system and model parameters to their initial state before running a model.

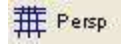
### Step 10: Run the model

Now click on the  button to actually start the model running..

You should see flowitems entering the queue and then moving to the processors. From the processors, flowitems should move to the conveyors and then to the sink. You can change how fast the model is running by sliding the run speed slide bar found at the bottom of the main window.

### Step 11: Model Navigation

The model is currently being viewed in an orthographic view window. Let's view the model in a perspective view. Close the orthographic window by selecting the X in the upper right corner of the window. Open the perspective view by selecting the



button on the toolbar (Figure 1-17).

## Mouse Navigation

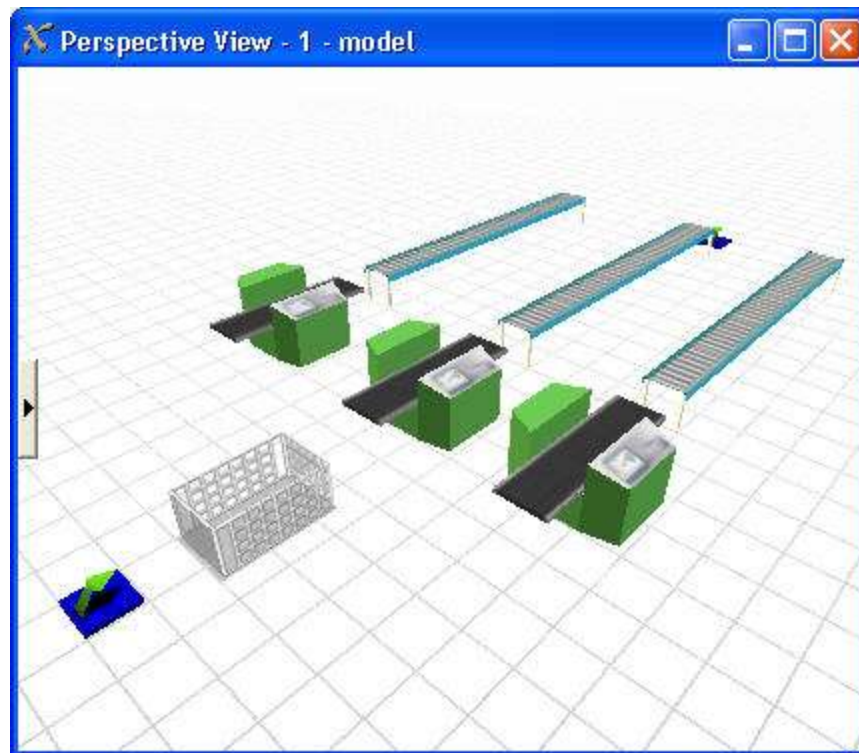
**Left mouse button:** Moves the model in the X-Y plane. If you click on an object it will move the object in the X-Y plane.

**Right mouse button:** Moves the X,Y,Z rotations. If you click on an object you can rotate the object.

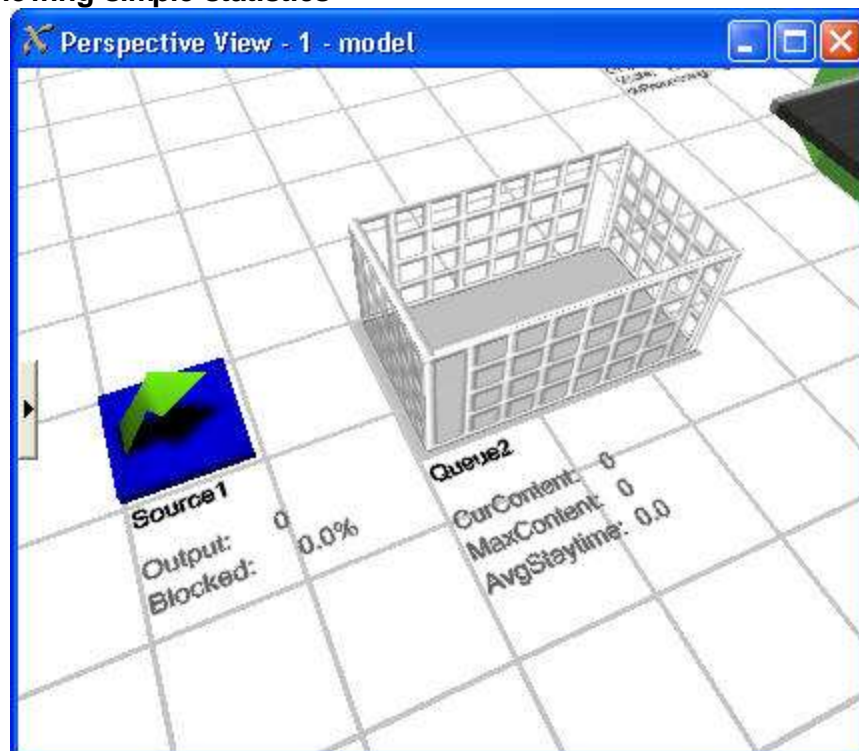
**Left and Right mouse buttons (or mouse wheel):** Zooms the animation in and out by rolling the mouse forward or backward. If an object is currently highlighted, you will change its Z height. If you have a wheel on your mouse you can roll the wheel instead of the left and right click.

**F7 Key:** The F7 key will toggle the view to "fly-through" mode. When in fly-through mode you can move the mouse cursor above the window center line to fly forward, below the center line to fly backward, left of the center line to rotate left, and right of the center line to rotate right. To exit fly-through mode, press F7. This navigation style may take a little practice to master, but give it a try. If you lose your model, you can stop the fly-through with F7, and then single right click in the window and choose the pop up menu option to Reset View.





### Step 13: Viewing simple statistics



To view simple statistics for each object, open the window's tool panel by clicking on the arrow button on the left side of the window. Then check the box "Show Names". Then collapse the tool panel by pressing on the arrow button again.



**Step 14: Save Model**

Save your model by pressing the Save button on the main toolbar.

You have now completed Lesson 1. Spend some time reviewing the steps and viewing the model as it runs. Congratulations!

To continue the tutorial, go to Lesson 2.

## Lesson 2

### Lesson 2

#### Introduction

Lesson 2 introduces the concept of adding operators and transporters to a model, and explores object properties and parameters in greater detail. Lesson 2 also introduces additional graphical statistical output options. Make sure you have completed lesson 1 before starting lesson 2 as lesson 2 will use the model from lesson 1 as a starting point.

#### What You Will Learn

- How to access object properties and parameters
- How to add a team of operators to the model
- How to add a fork truck transporter to the model
- How to select an object for statistics
- How to turn on stats collecting
- How to view object statistics while the model is running

#### New Objects

In this lesson you will be introduced to the Dispatcher, Operator, and Transporter objects.

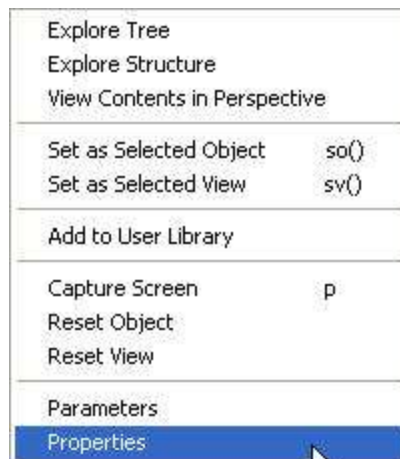
#### Approximate Time to Complete this Lesson

This lesson should take about 30-45 minutes to complete.

## Flexsim Software Concept Learning

### Object Properties and Parameters

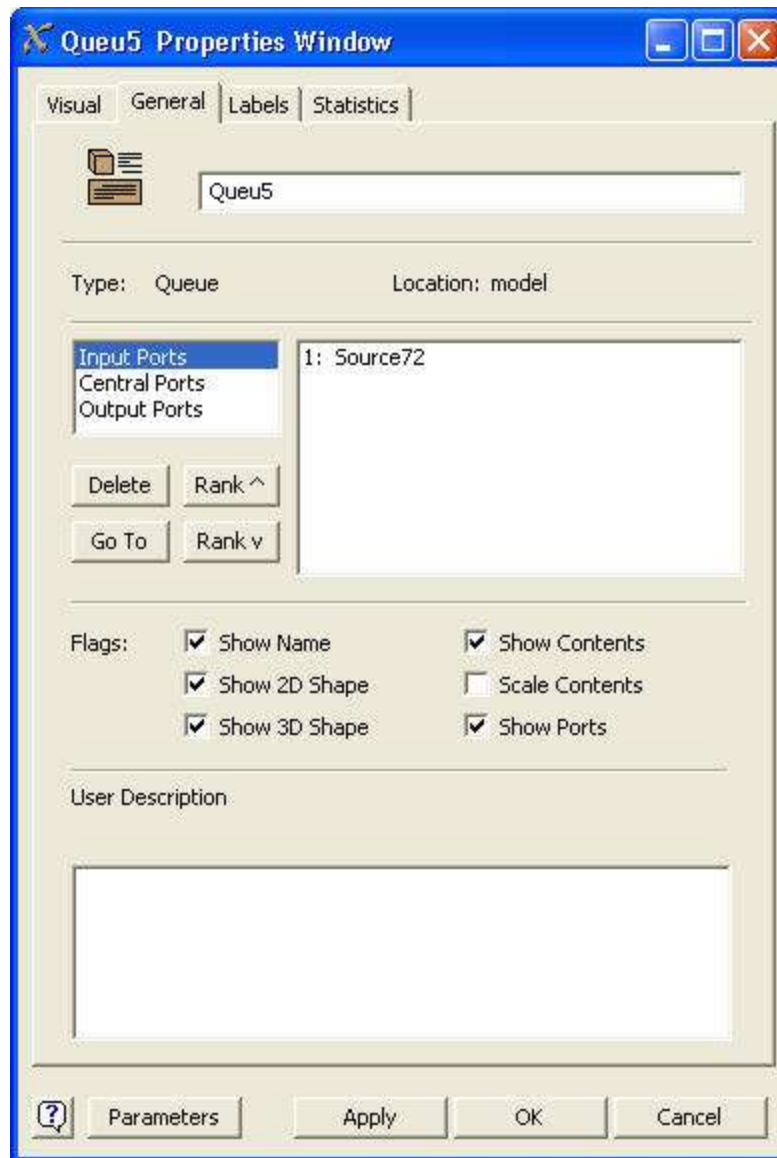
It is now time to introduce the object Properties and Parameters Windows in a more organized manner. Every Flexsim object has both a properties window and a parameters window. As a modeler you need to completely understand the difference between object properties and object parameters. To access properties you right click on an object in the model view window and select Properties.



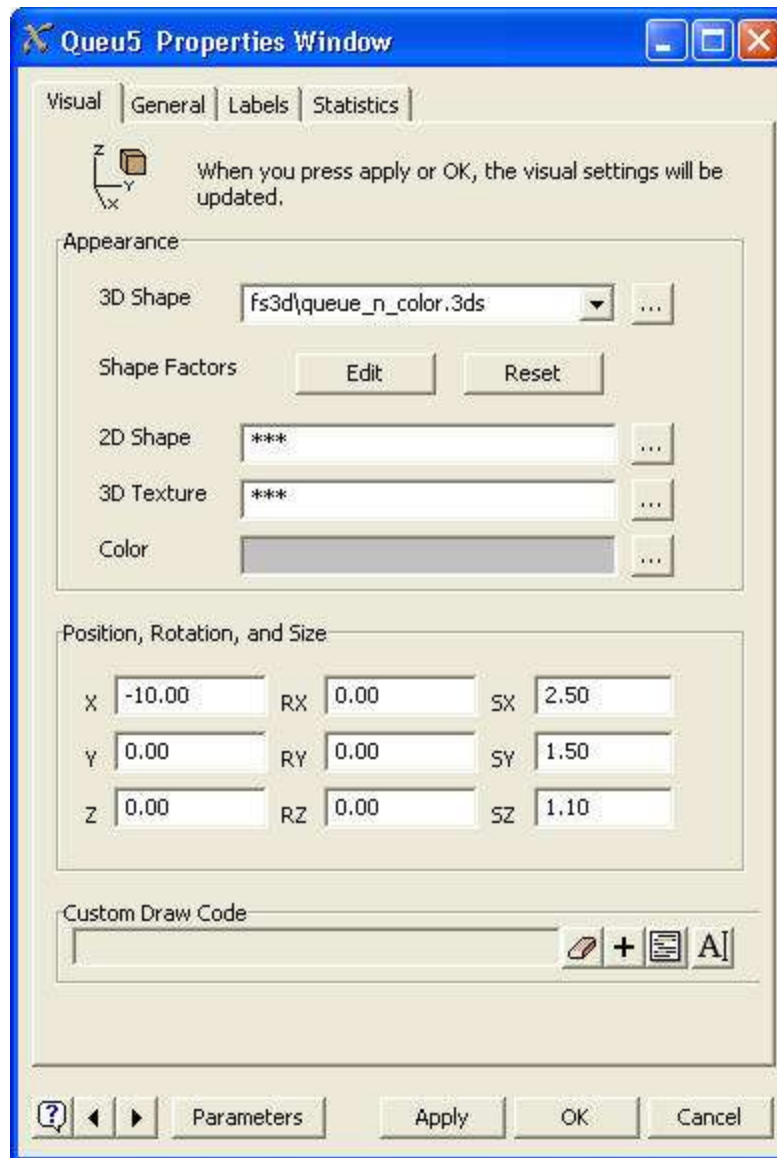
### Object Properties

The Properties Window is the same for every Flexsim object. There are four tabs in the Properties Window: Visual, General, Labels, and Statistics. Each tab contains information pertaining to the selected Flexsim object.

**General properties:** the General tab contains general information about the object such as the name, type, location, port connections, display flags, and a user description.



**Visual properties:** The Visual tab allows the modeler to specify the visual aspects such as 3D shape, 2D shape, 3D textures, color, position, size, rotation, and custom OnDraw code. Position, size, and rotation reflect the current properties of the object. The modeler can change values in the fields to change position, size, or rotation, or the modeler can change them by using the mouse in the model view window.

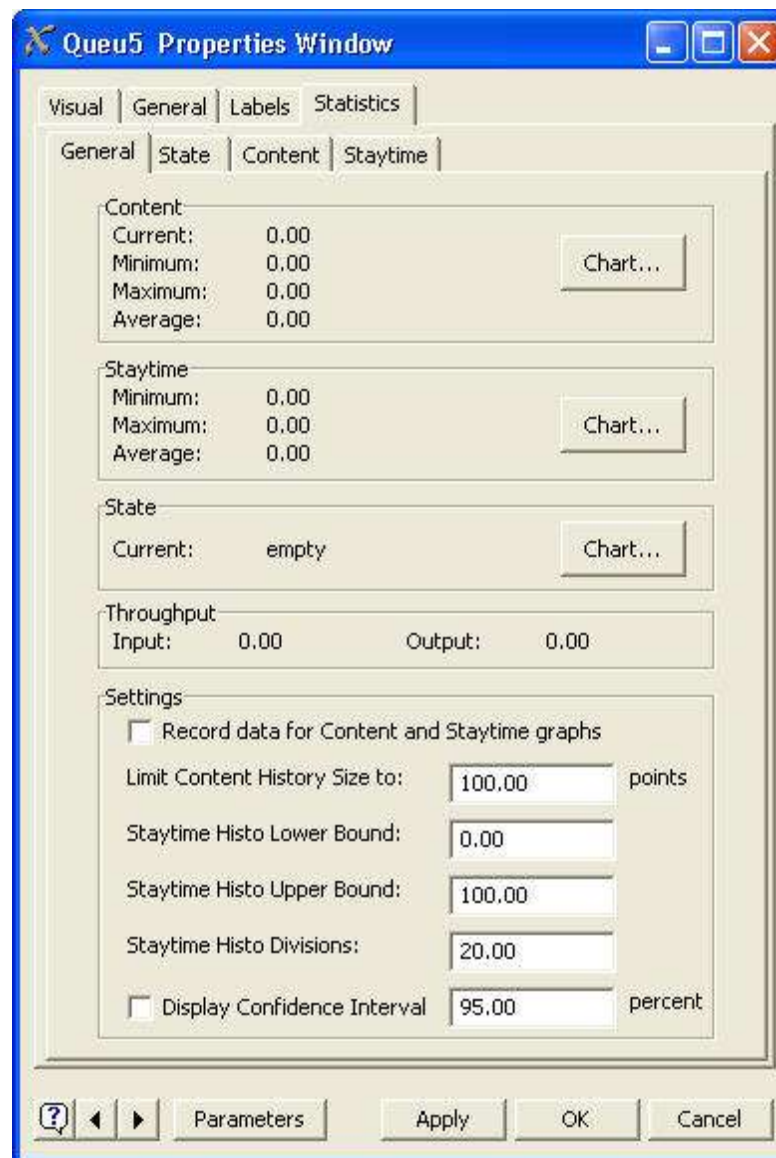


**Label properties:** The Labels tab displays the user defined labels that are assigned to the object. Labels are a mechanism through which modelers can store temporary data. There are two parts to a label, a name and a value. The name can be anything you want and the value can be numeric or alphanumeric (a string). To add a label that will hold only numeric data, click on the "Add Number Label" button at the bottom of the window. Likewise, if a label will hold both numbers and letters, click on the "Add String Label" button. You can then modify both the names and values of the labels using the table.

Labels can also be updated, created, or deleted dynamically while running the model. This tab will show all the labels and their current values. All information is displayed in real time as the model runs. This information is very helpful to modelers as they test logic and debug models.

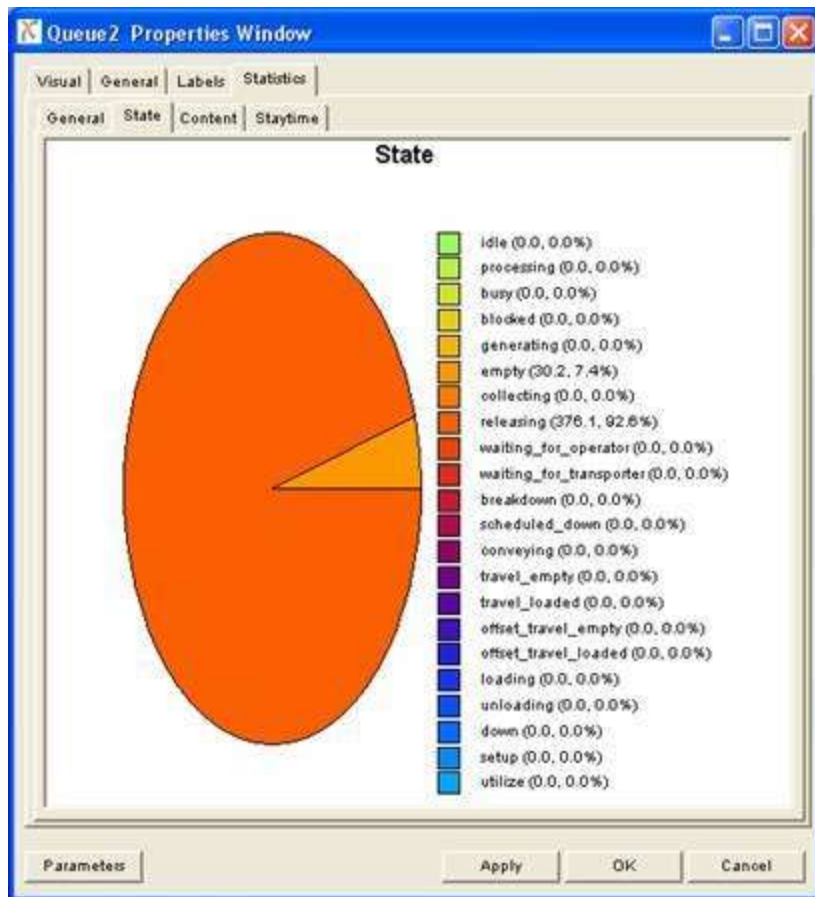


**Statistic properties:** The Statistics tab displays the default information that is collected on an object. This information is updated dynamically and displayed as the model runs. When this tab is selected, four sub-tabs are displayed.



**Statistics General properties:** Displays point statistics for the content, staytime, state, and throughput of the object. The Settings options allow the user to determine how much data will be displayed in the content and staytime charts.

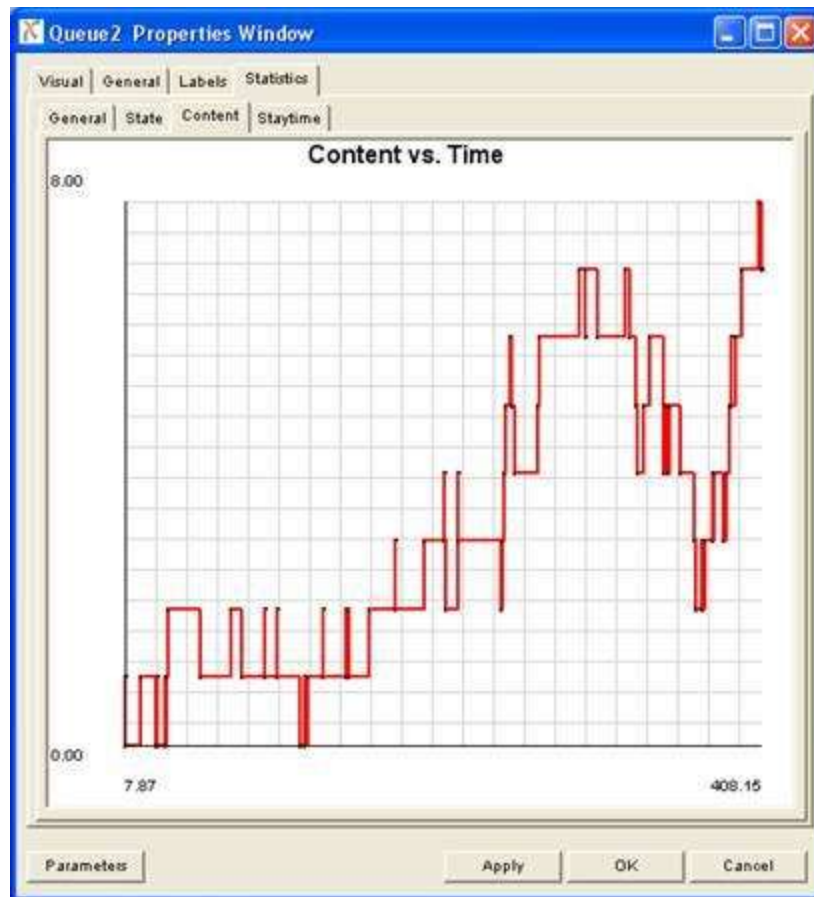
**Statistics State properties:** The state properties chart shows the percentage of time the object was in each of its states.



The state chart is dynamically updated as the model runs. A separate window with the chart view can be displayed by selecting the chart button from the general statistics tab.

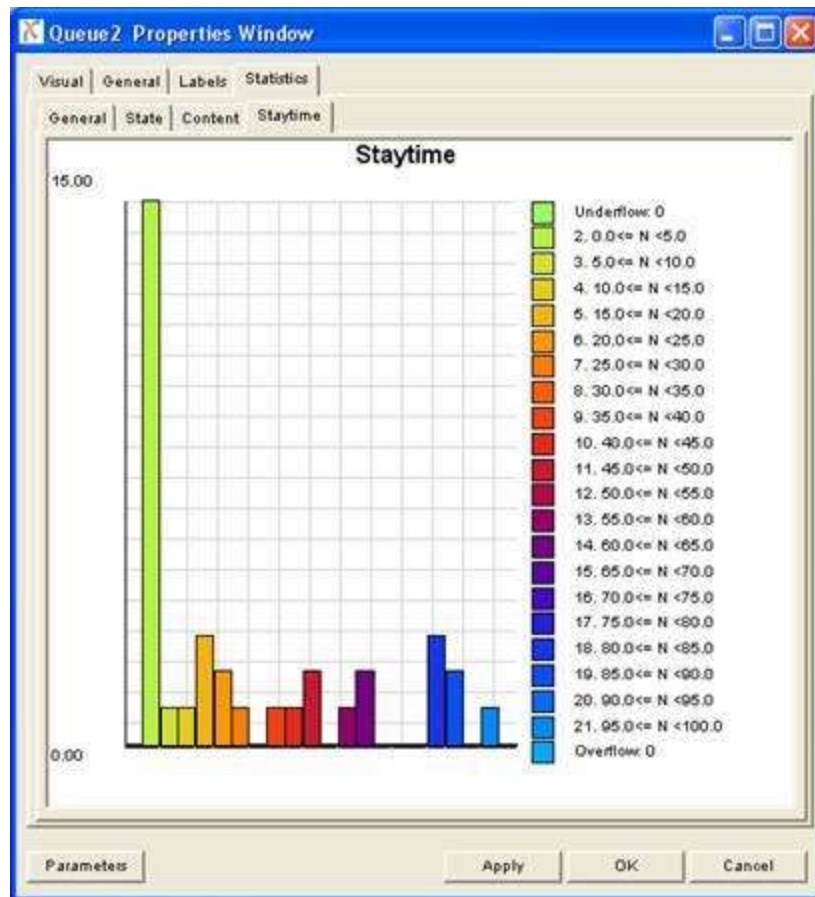
**Statistics Content properties:** The content properties chart shows the content of the object over time. Stats collecting must be turned on for this chart to be created.





The content chart is dynamically updated as the model runs. A separate window with the chart view can be displayed by selecting the chart button from the general statistics tab.

**Statistics Staytime properties:** The staytime properties chart shows a histogram for the amount of time flowitems dwelt in the object. Stats collecting must be turned on for this histogram to be created.

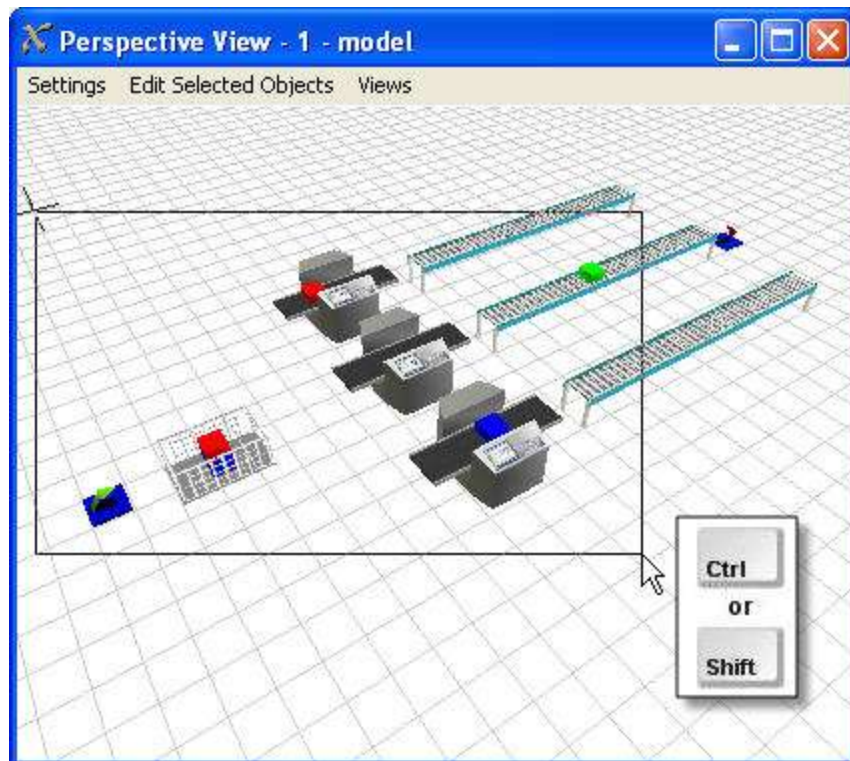


The staytime histogram is dynamically updated as the model runs. A separate window with the chart view can be displayed by selecting the chart button from the general statistics tab.

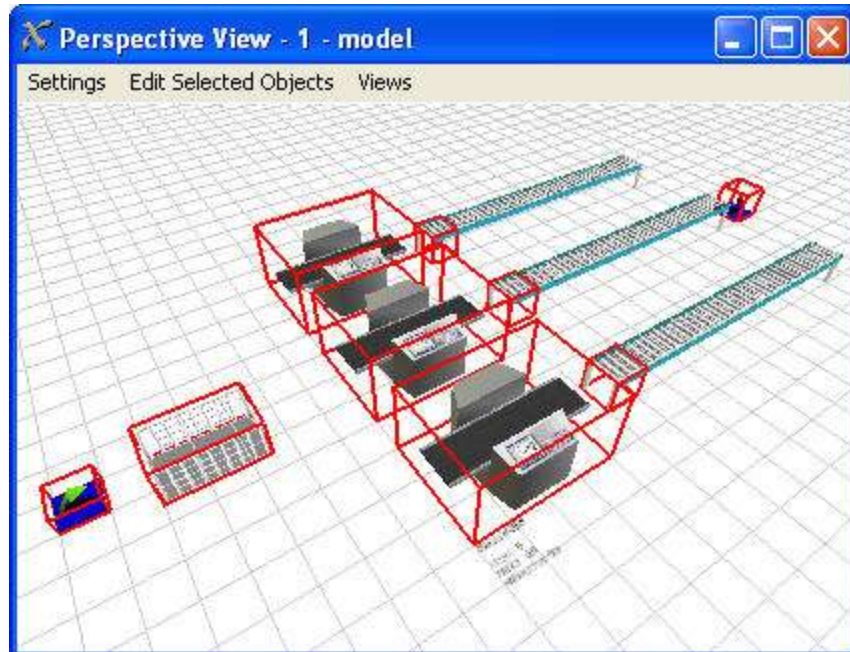
**NOTE:** To view reports, content charts and staytime histograms, the modeler must turn on Object Graph Data collecting for the object. Since the storing of histories can require large amounts of hard disk space, this is turned off by default. The following steps are needed to turn it on.

### Step 1: Selecting objects for stats

In the model view window you will need to make a selection of objects for which you would like to record stats. This is done by holding the "Shift" key on the keyboard while dragging the mouse to include the objects you want to select. Pressing the "Ctrl" key and then clicking an object adds or removes individual objects from the selection set.

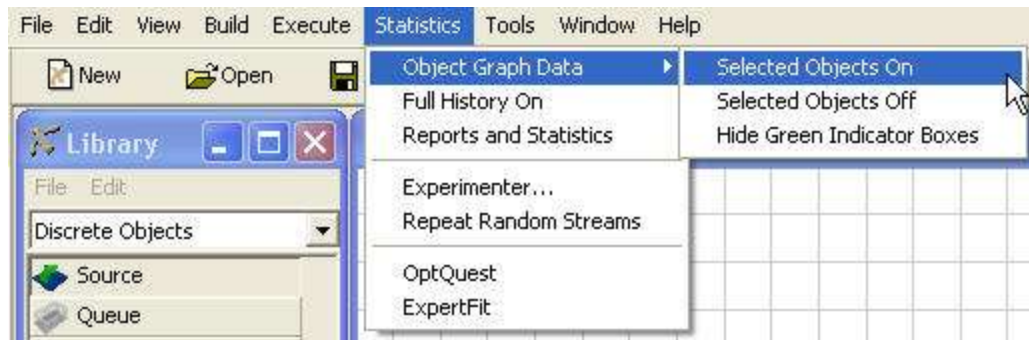


Once an object is selected you will see a red box around it.

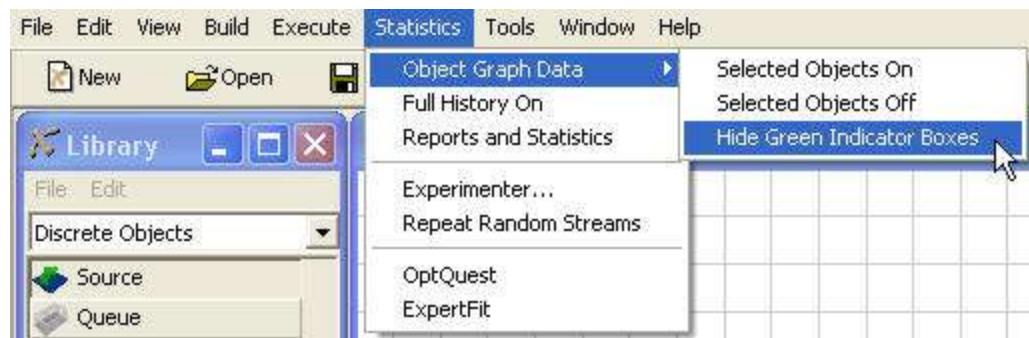
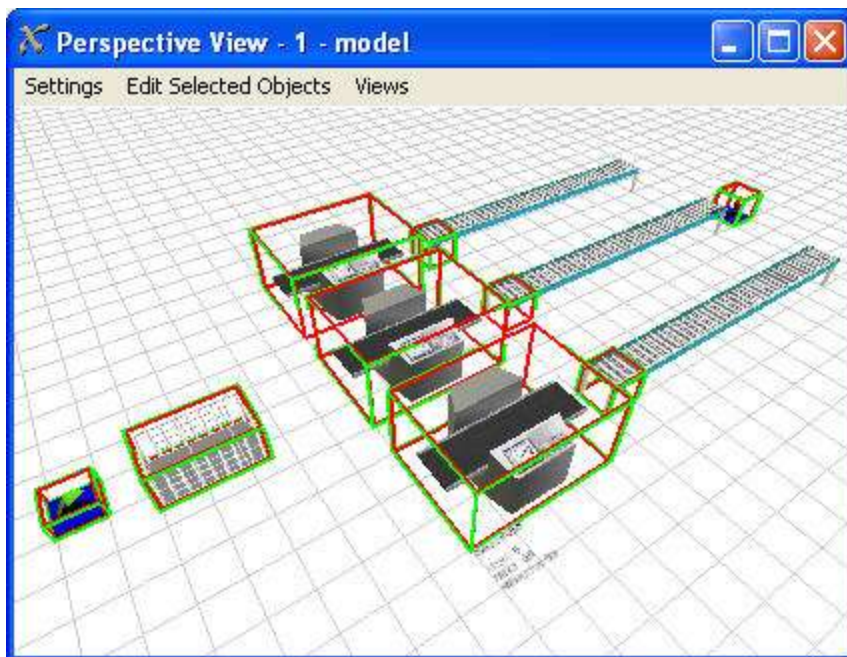


### Step 2: Stats On

To gather stats histories on the selected objects, click on Statistics > Object Graph Data > Selected Objects On, and be sure that "Global On" is checked as well.



Once Stats Collecting has been turned on, you will see a green box around the objects that are recording stats histories. You may choose to turn off the display of green boxes by selecting Statistics > Object Graph Data > Hide Green Indicator Boxes.

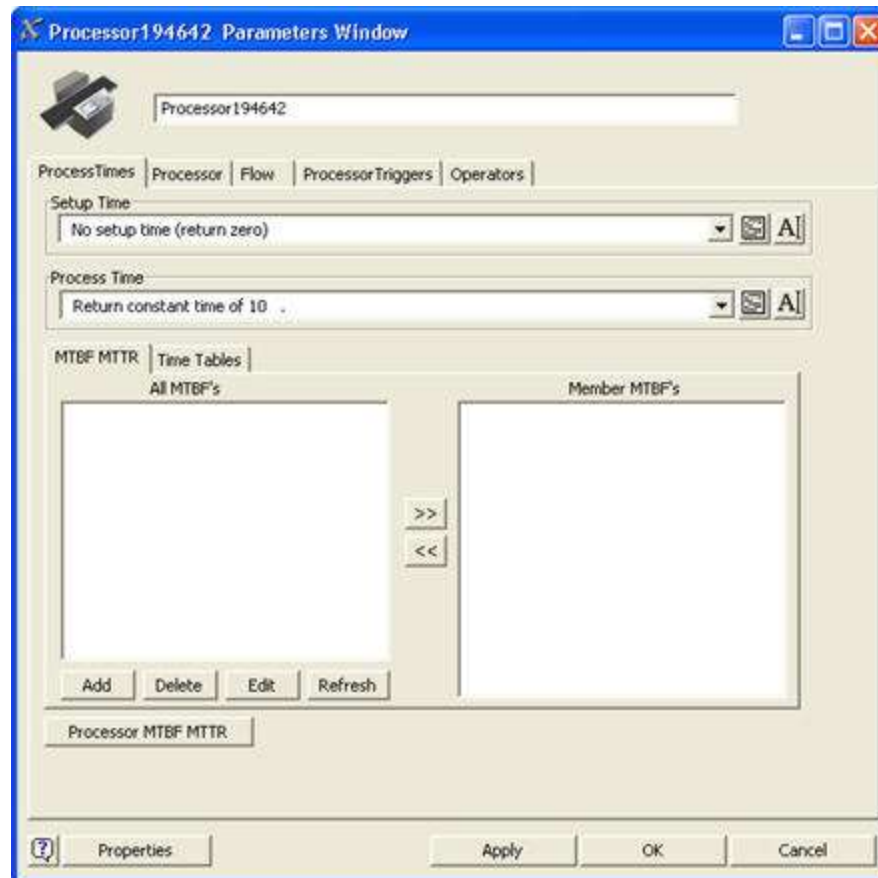


You can now run the model and collect stats histories on all objects that were selected.



## Object Parameters

The object's parameters window will differ depending on the object selected. Certain page tabs will be similar on all objects and others will be very specific to the object. For a specific definition of all the object parameters for each object, see the Flexsim Object Library. Double clicking on an object accesses that object's parameters.

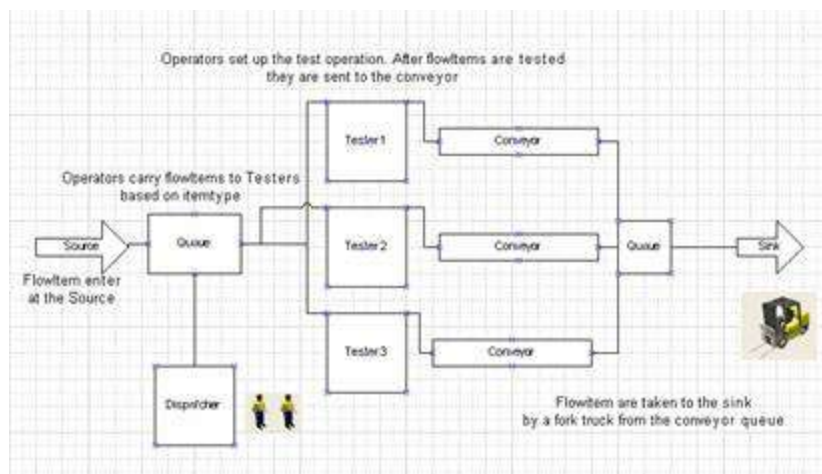


This ends the Flexsim software concept learning section. It is now time to build model 2.

## Model 2 Description

In model 2 we will use a team of operators to perform the setup for the testing procedure for the flowitems in the model. One of two operators will need to set up the test. Once set up, the test can run without the need for the operator to be present. The operators must also carry the flowitem to the test station before the setup can begin. When the test is complete, the flowitem moves to the conveyor without the assistance of the operator.

When the flowitem reaches the end of the conveyor it will be placed in a queue where it will be picked up by a fork truck and taken to the sink. We may find it necessary to have more than one fork truck once we view the model as it runs. After the model is completed, view the default charts and graphs and address any bottlenecks or efficiency concerns. Below is a flow diagram of model 2.



## Model 2 Data

**Tester set-up time:** Constant time of 10 seconds

**Product handling:** Operator from queue to tester. Fork truck from conveyor queue to sink.


**Conveyor Queue:** Capacity=10

## Step-By-Step Model Construction

## Building Model 2

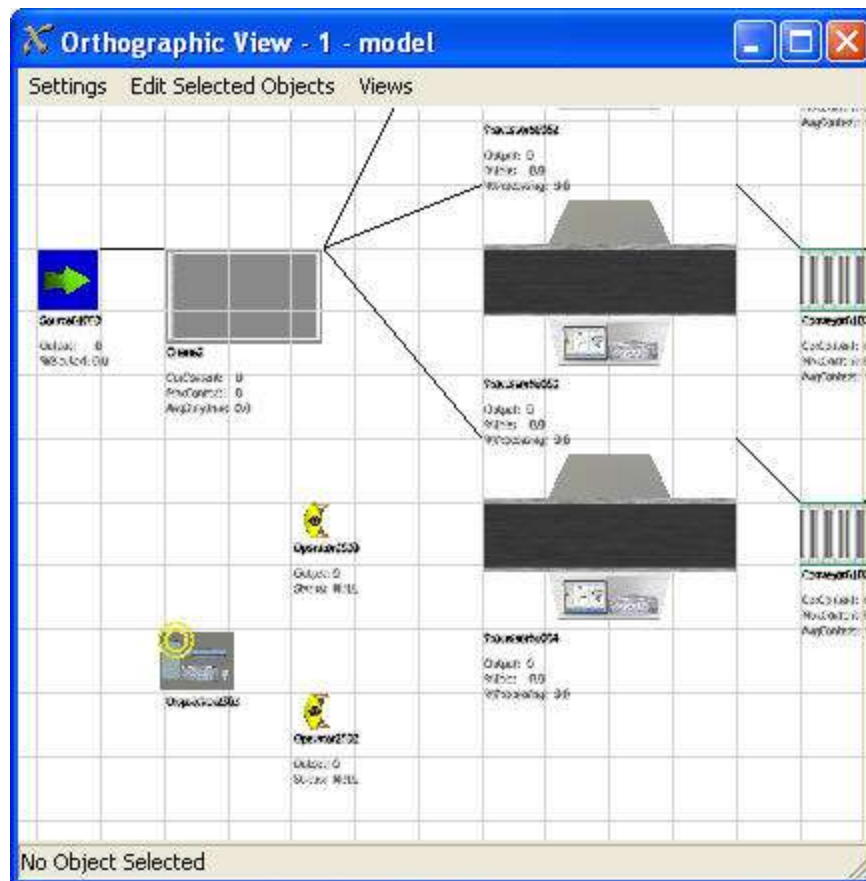
To start building model 2 you will need to load model 1 from the previous lesson.

### Step 1: Load model 1

Load model 1 by selecting the  button on the toolbar. Select the file for model 1 (.fsm file) saved from lesson 1.

## Step 2: Add a dispatcher and 2 operators to the model

The Dispatcher is used to queue up task sequences for a team of operators or transporters. In this case it will be used with 2 Operators that will be used to move flowitems from the queue to the testers. To add the dispatcher and 2 operators, click-and-drag them from the library and place them in the model.

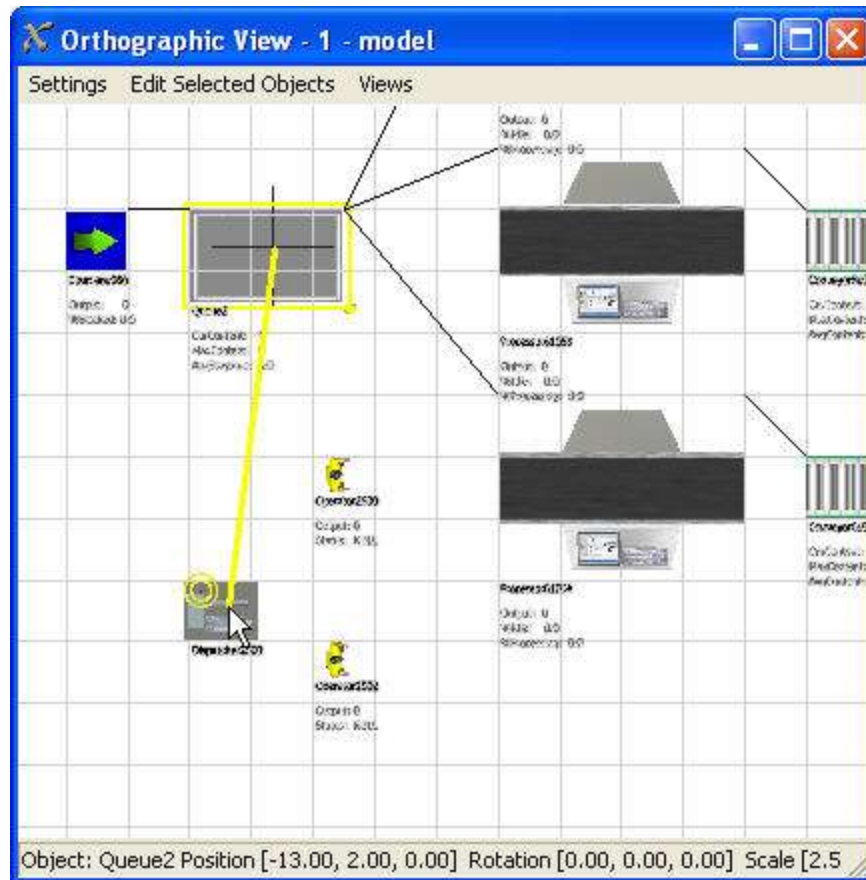


### Step 3: Connecting Central and Input/Output Ports

The queue is going to request an operator to pick up the flowitem and take it to one of the testers. The flow logic has already been set on the queue in lesson 1. You will not need to change the flow logic. You will only need to request an operator to be used for the task. Since we are using 2 operators, we will use a dispatcher to queue

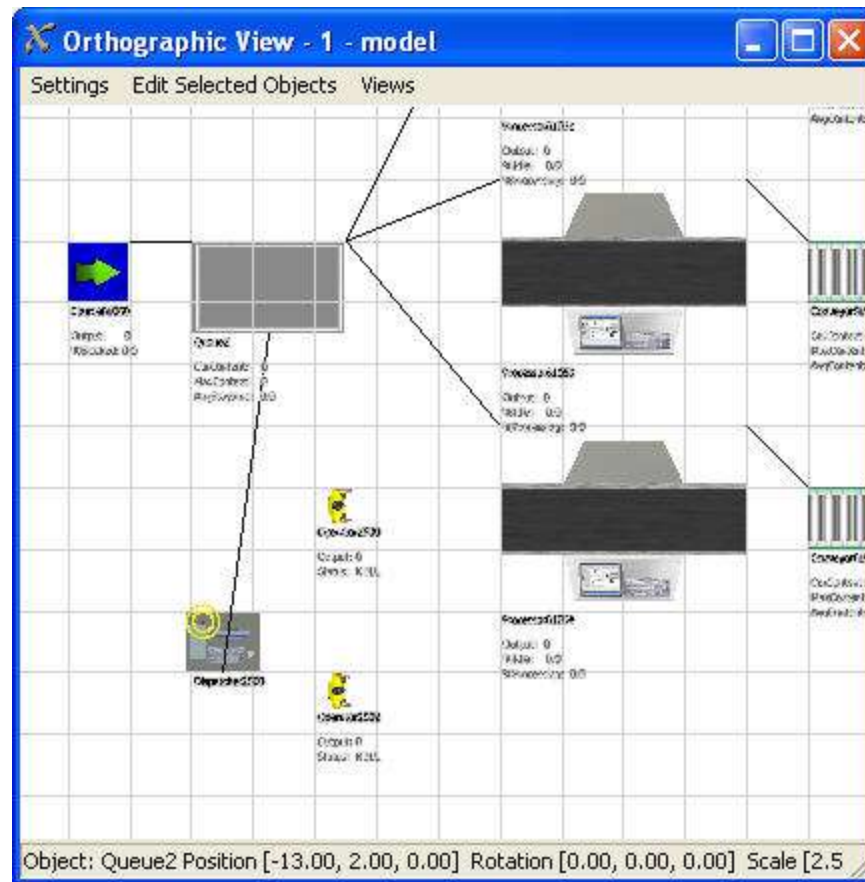
the requests and then pick a free operator to perform the task. If we only had 1 operator, we would not need the dispatcher and could connect the operator to the queue directly.

In order to use the dispatcher to direct a team of operators to perform a task, the dispatcher must be connected to the central port of the object requesting the operator. To connect the central port of the dispatcher to the queue, press and hold the "S" key on the keyboard and then click-and-drag from the dispatcher to the queue.



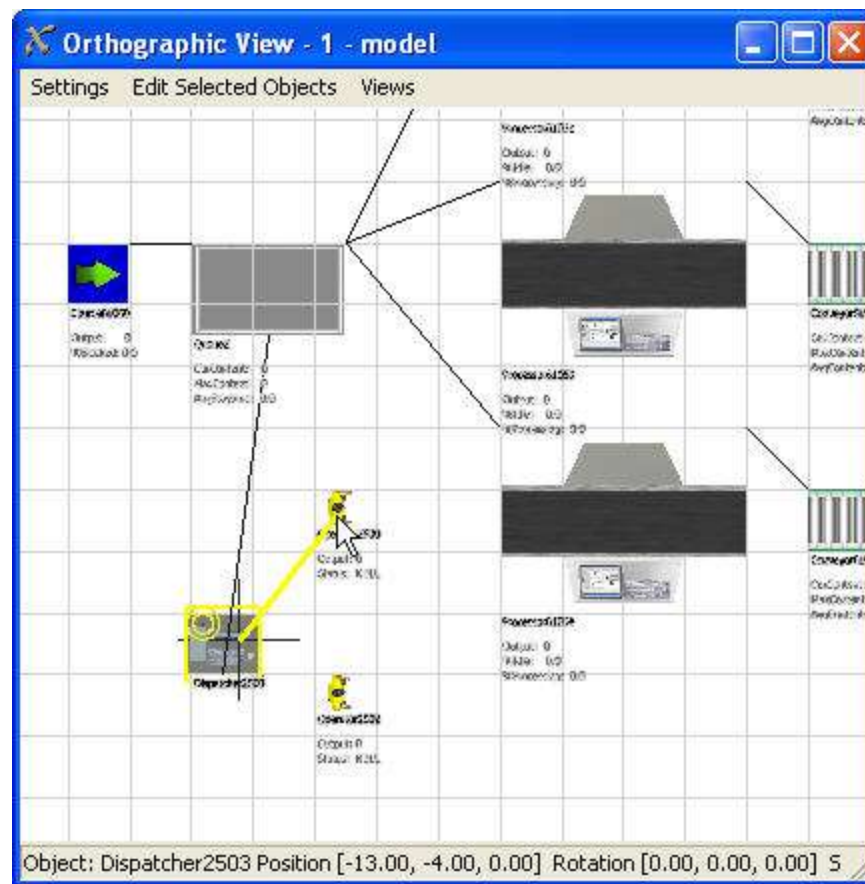
When you release the mouse a connection from the dispatcher's central port will connect to the central port of the queue.

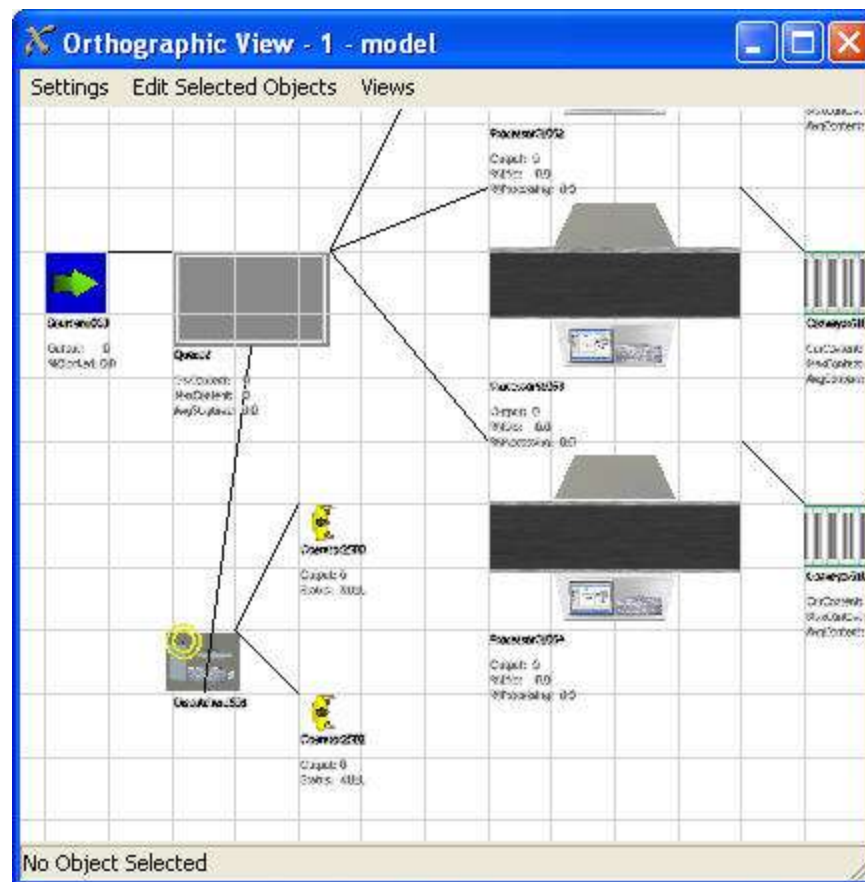




The central port is located in the center bottom position of the object.

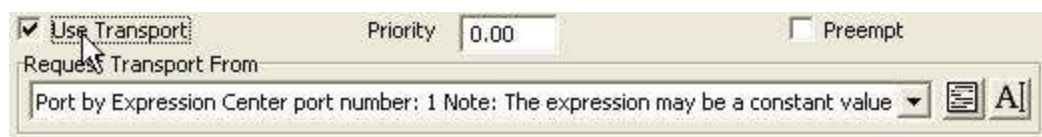
In order for the dispatcher to send tasks to the operators, the dispatcher's output ports needs to be connected to the operators' input ports. This is done by pressing and holding the "A" key on the keyboard and click-and-drag from the dispatcher to the operator. This must be done for each operator. The connections are shown below.






### Step 4: Modifying Queue Flow to Use Transport (Operators)

The next step is to modify the queue's Flow parameters to use the operators to make the move. This is done by double clicking on the queue to bring up the Parameters Window. Once the window has appeared, select the Flow tab. Check the box to "Use Transport" under the "Send To Port" picklist.



A new pick list will appear called "Request Transport From" when "Use Transport" is checked. This pick list allows you to select which Transporter or Operator to move the item based on the port number. In this case it is the object connected center port 1, or the Dispatcher, that assigns the operator to the task. Select "OK" to close the window.

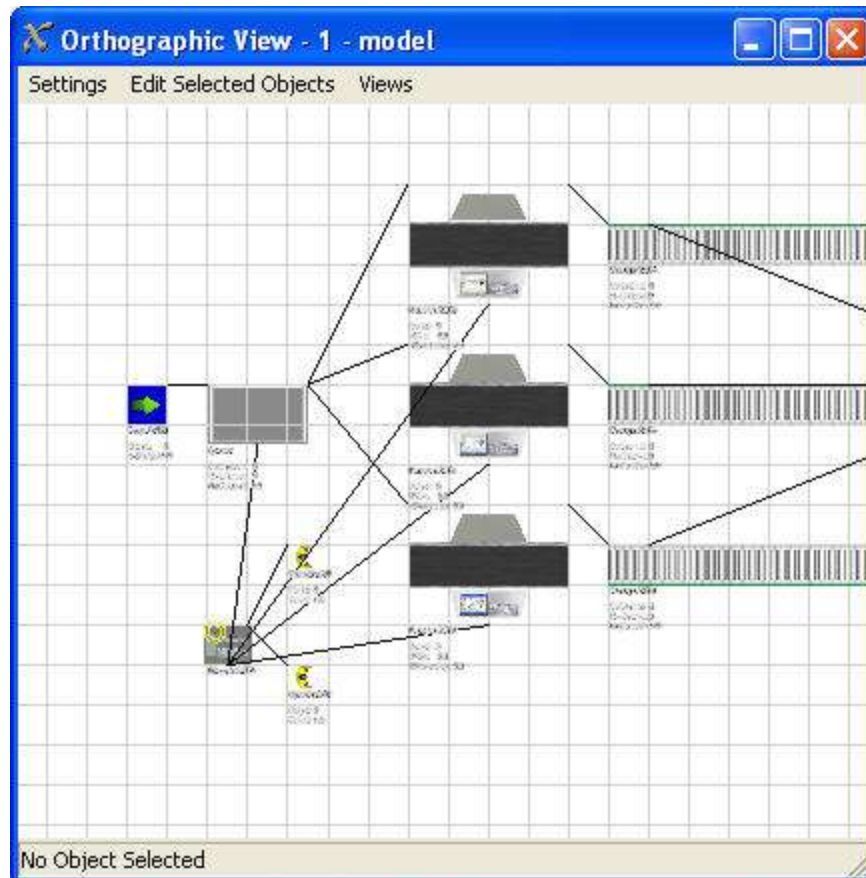
## Step 5: Save the Model, and Test Run

Now we should run the model to make sure that the changes we have made are working. Reset the model and then save the model by pressing the  Save button on the toolbar.

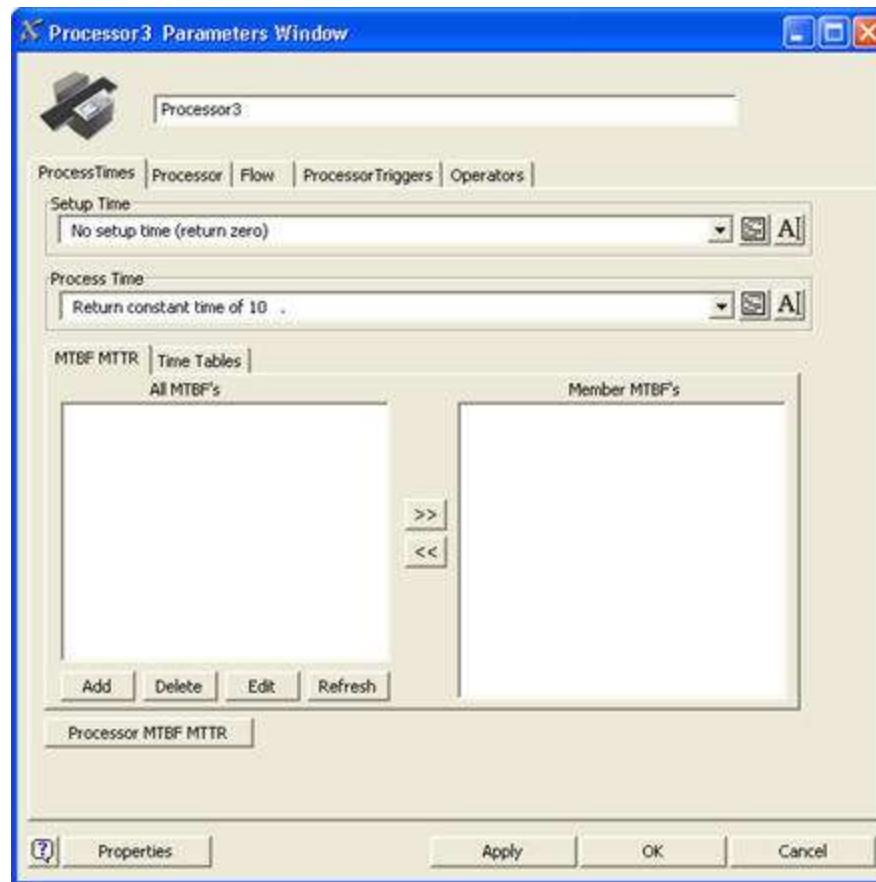
Run the model to verify that the operators are moving the flowitems from the queue to the testers.

### Step 6: Incorporating Operators for the Setup Time for the Testers

In order for the testers to use the operators during set up, a connection must be made between the central ports of each tester to central ports of the dispatcher. This is done by pressing and holding the "S" key on the keyboard and click-and-dragging between the dispatcher and each tester. The ports will be displayed as shown below when completed.



Now we need to define the setup time for the testers. Double-click on the first tester to bring up the Parameters Window.



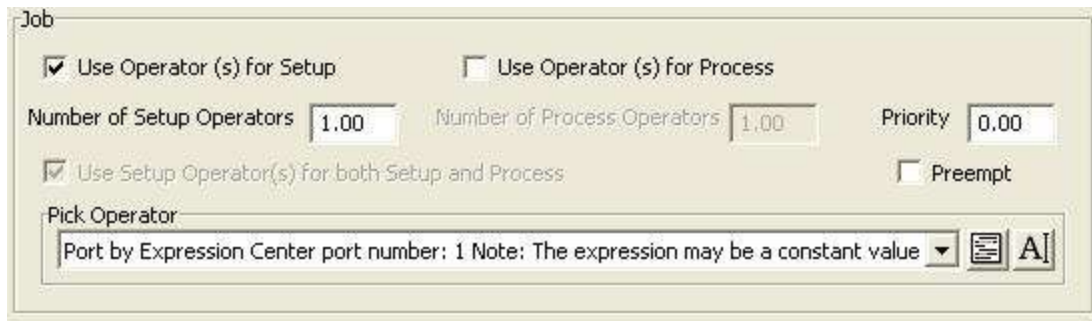
On the "Setup Time" pick list select the "By Expression" option, and then change the time to 10 in the template pop-up window.

By Expression  
 Expression: 10

Note: The expression may be a constant value or the result of a command (getitemtype(), getlabelnum(), etc).

Click "Apply" to save the change. Now open the "Operators" tab.

Check the box next to "Use Operator(s) for Setup". When the box is checked you will see the "Number of Setup Operators" edit field and the "Pick Operator" pick list appear. The number of operators needed for the setup is 1 and the reference for the "Pick Operator" should be set the central port number 1.



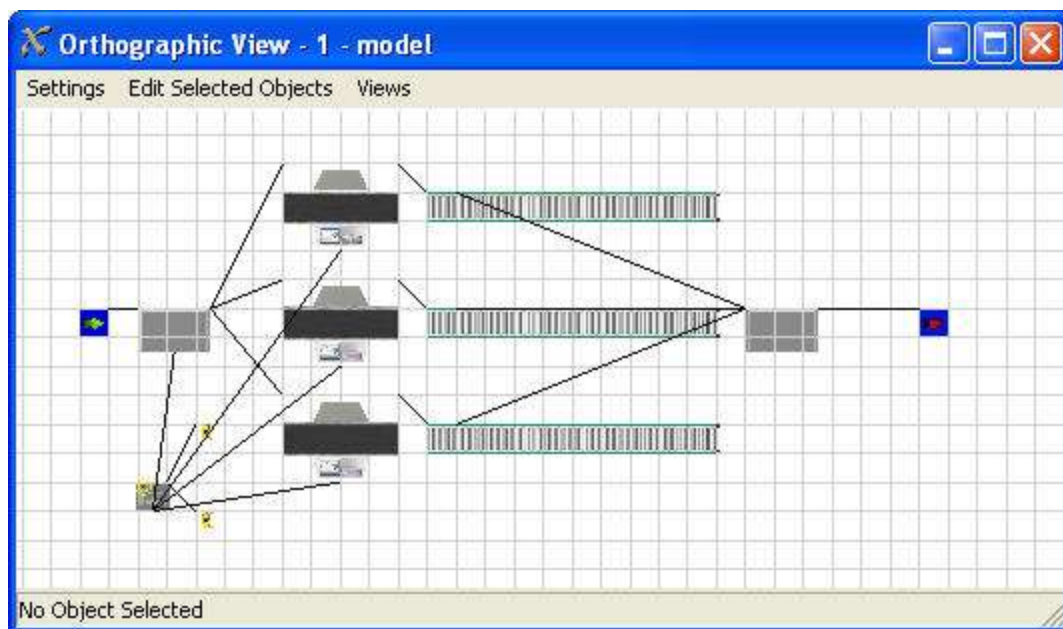
Click "OK" to save the changes and close the window. Repeat this process for each tester in the model. Then reset and run the model to verify that the operators are being used during the setup time.

The next step in the model is to add the conveyor queue and reconnect the input and output ports.

### Step 7: Disconnecting Ports from Conveyors to the Sink

Before adding the conveyor queue it is necessary to disconnect the input and output port connections between the conveyors and the sink. This is done by pressing and holding the "Q" key on the keyboard and then click-and-dragging from the conveyor to the sink.

Once the ports are disconnected, drag-and-drop a queue from the library to the end of the middle conveyor. Then connect the conveyors' output ports to the queue input port by pressing and holding the "A" key and click-and-dragging from each conveyor to the queue. Then connect the output port of the queue to the sink using the same process.

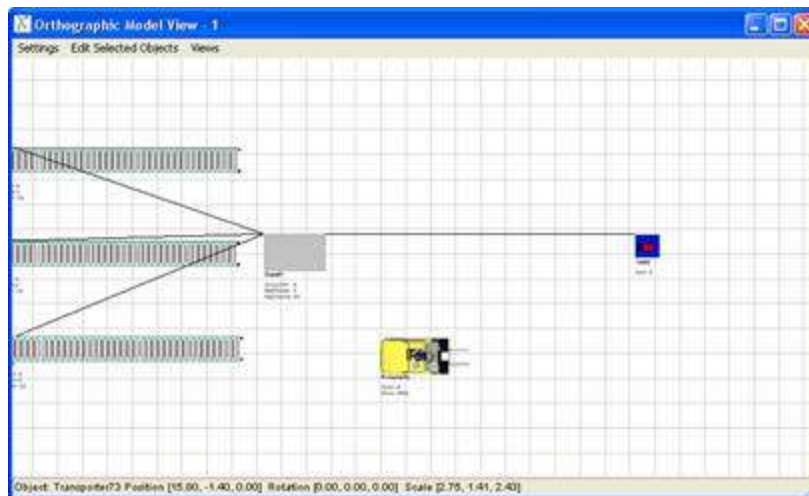


Now that the layout has been revised and the port connections created, the fork truck can be added.

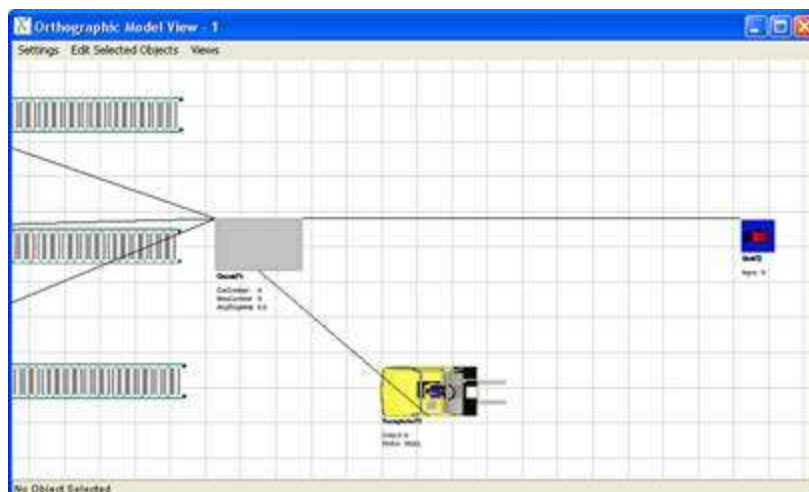
### Step 8: Adding the Fork Truck

Adding the fork truck to the model to move flowitems from the conveyor queue to the sink is exactly the same as adding operators to move flowitems from the input queue to the testers. Since there will be only 1 fork truck in the model, there is no need for a dispatcher. The fork truck will be directly connected to a central port of the queue.

Drag and drop a Transporter from the library into the model view.



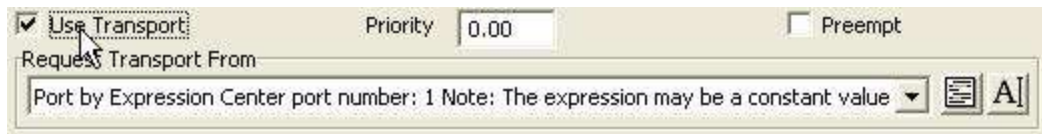
Once the fork truck is in the model, connect the central port of the queue to the fork truck. Press and hold the "S" key on the keyboard and click-and-drag from the queue to the fork truck.



### Step 9: Adjust the Queue Flow Parameters to Use the Fork Truck

The next step is to adjust the queue flow parameters to use the fork truck. Double click on the queue to bring up the Parameters Window



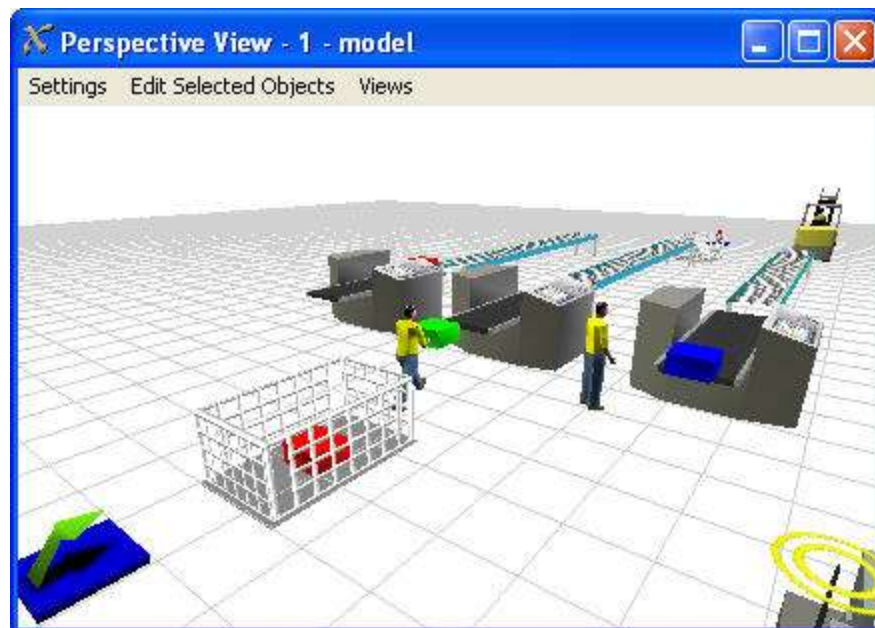


Select the Flow tab and check the "Use Transport" option. The central port 1 of the queue is already connected so there is no need for any adjustments. Select "OK" to close the window.

Reset and save your model.

### Step 10: Run the Model

This is the rewarding part of building the model. It's time to check the model to see if it is working the way you want. While the model is running, you can use the animation to visually inspect the model to see if everything is working properly



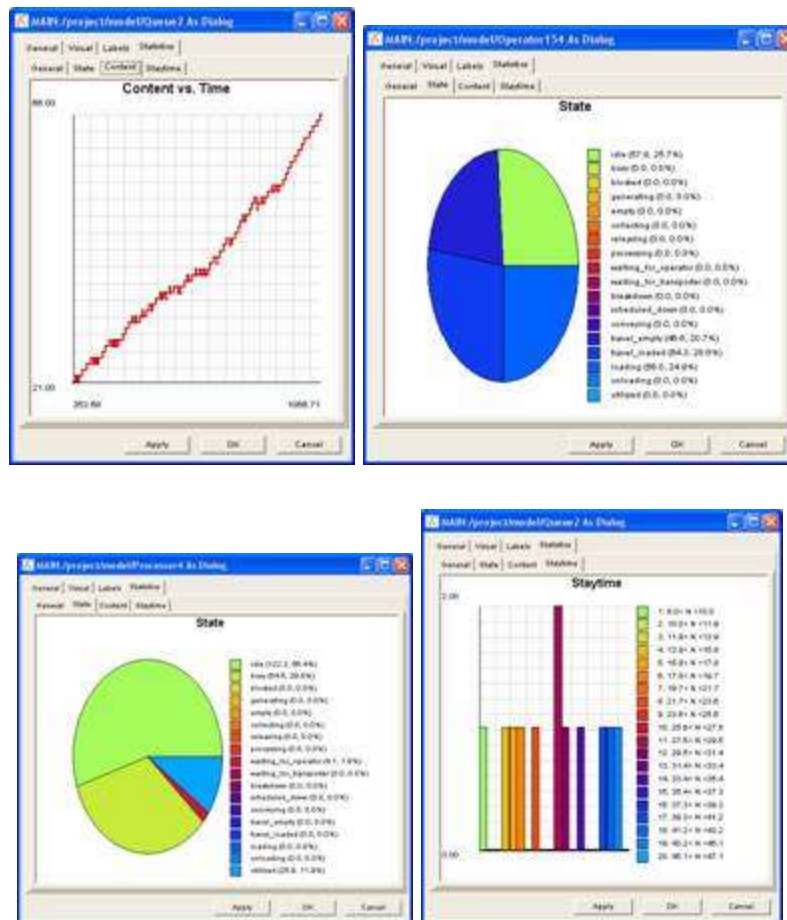
You should see the operators moving back and forth and the fork truck moving flowitems from the queues to the sink.

You will notice that when a tester is waiting for an operator to perform the setup, a yellow square is placed under the tester.

### Step 11: Analyzing the Output

Using the information about how to turn on stats collecting explained earlier in the lesson, view the object statistics in the properties window. By looking at the animation and the charts, does this model have a bottleneck?





It becomes obvious that if you add one more operator the model will run better. Even though the flowitems may still back up at the input queue it will be in its optimum configuration with the addition of a third operator.

Add another operator by simply drag-dropping another operator from the library. Then connect the dispatcher to the operator by pressing and holding the "A" key and click-and-dragging a connection. Reset, save, and then run. This ends Lesson 2. Congratulations! Can you go the extra mile?

To continue the tutorial, go to Lesson 2 Extra Mile.

## Lesson 2 Extra Mile

### Lesson 2 Extra Mile

#### Introduction

This extra mile session is designed to teach the modeler how to add the extra touch to make the model show data and information as the model is running. In this lesson we will look at how to add 3D charts and graphs, and visual 3D text to the model you finished in lesson 2.

#### What You Will Learn

- How to add a 3D content graph for the Queue
- How to add a 3D histogram to show the wait time for the Queue
- How to add a 3D pie chart to show the state profile for each operator
- How to add 3D visual text to show the average wait at the Conveyor Queue
- How to position the graphs, charts, and text for best viewing

#### New Objects

In this lesson you will be introduced to the VisualTool and Recorder objects.

#### Approximate Time to Complete this Lesson

This lesson should take about 20-30 minutes to complete

## Step-By-Step Model Construction

### Building Model 2 Extra Mile

To start building model 2 extra mile you will need to load model 2 from the last lesson.

#### Step 1: Load Model 2

#### Step 2: Save Model as "Model 2 Extra Mile", and turn on History

Go to the menu option File > Save Model As... to save your model under a new name. Before you start making changes, make sure that you have turned on the stats collecting for all your objects using the menu option Stats > Stats Collecting > All Objects On. Stats collecting needs to be on to show histograms and content plots (see Lesson 2, page 2-11 - 2-13).

#### Step 3: Add a Recorder to Show the Content of the Queue

Drag a Recorder from the library and place it above and to the left of the Source object as shown in Figure 2-32.

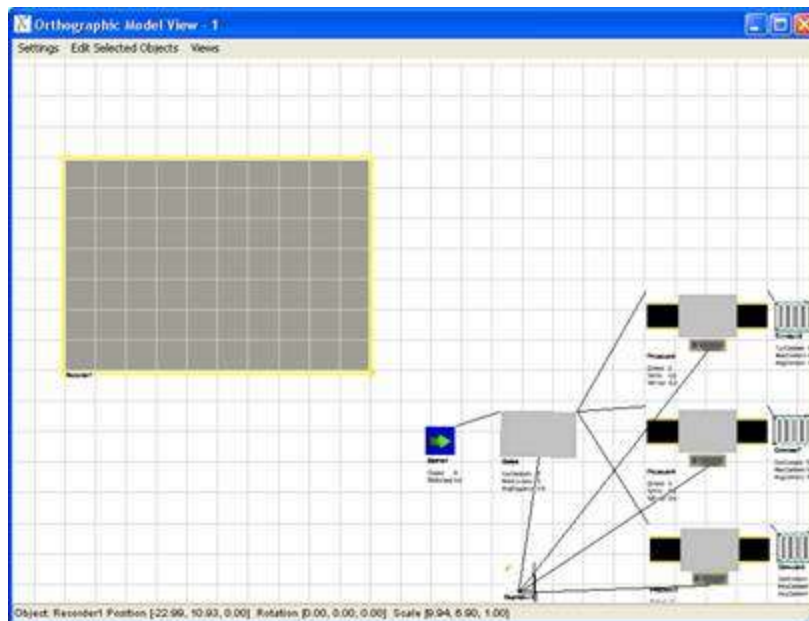


Figure 2-32

#### Step 4: Adjust the Parameters of the Recorder to show a Content Graph of the Queue

Double-click on the Recorder object to bring up its parameters window as shown in Figure 2-33.



Figure 2-33. Recorder Parameters

Press the Data Capture Settings button. In the Type of Data field, select the "Standard Data" option. Then select the queue in the pick list of the Object Name field. In the Data to capture field, select "Content" (see Figure 2-34).



Figure 2-34. Capture Data Options

Then press the "Next" button.

#### Step 5: Set the Display Options of the Recorder

Now select the Display Options button on the Recorder window (see Figure 2-33). In the Graph Title field, type the title "Queue Content Graph" (see Figure 2-35). This is a user defined field for the title of the graph. You can type anything you want here. Press the Done button when you are finished.

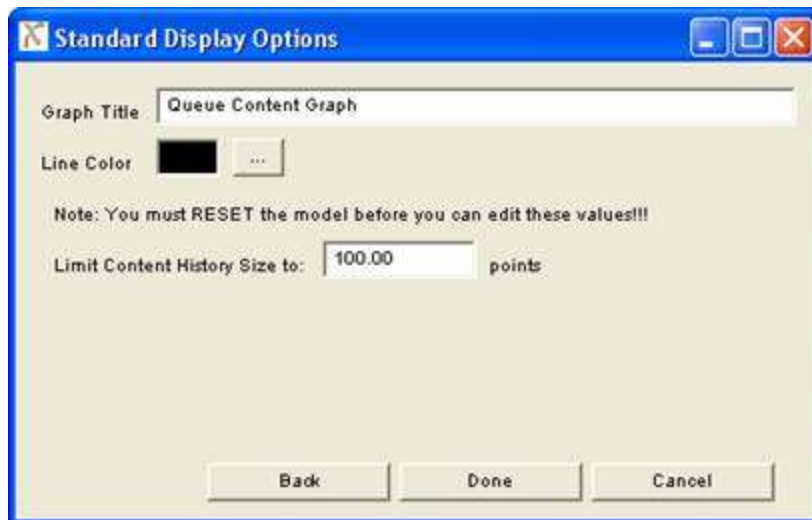


Figure 2-35. Standard Display Options

### Step 6: Adjust the Visual Properties of the Graph

The visual properties of the graph can be edited in the Properties Window by right clicking on the Recorder and then selecting Properties (see Figure 2-36).

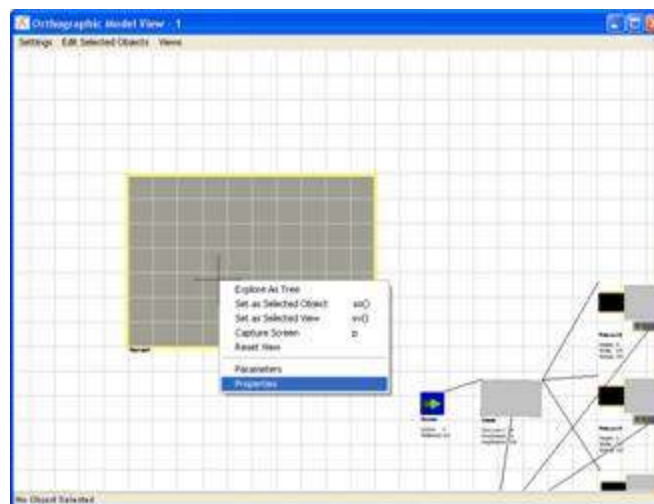


Figure 2-36. Selecting the Properties Window

By default the graph will lay flat on the floor of the model. It makes a good impression if the chart is tilted up 90 degrees to stand straight up. This is done by changing the rotation and height of the recorder (see Figure 2-37).

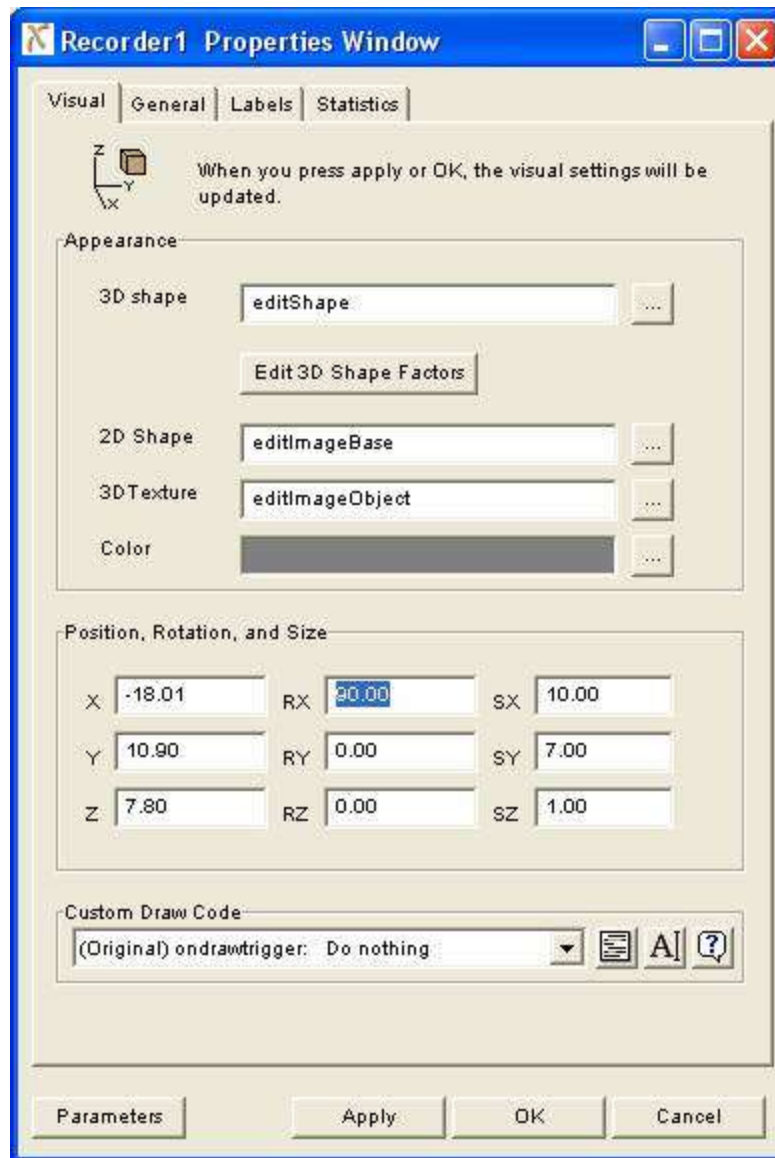


Figure 2-37. Recorder Properties

Change the "Z" (position) to 7.80 and the "RX" (rotation) to 90. This will rotate the chart up and set the height to place the bottom of the chart on the floor level (see Figure 2-38).

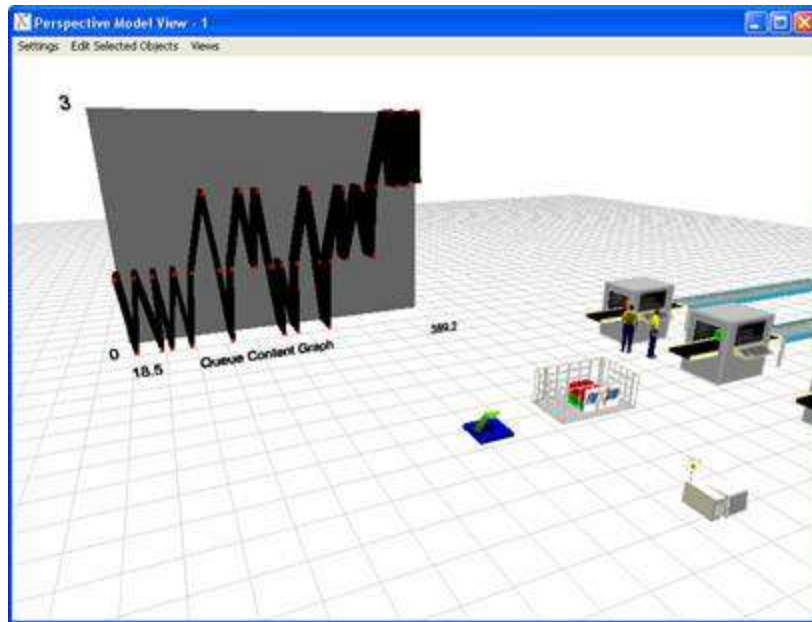


Figure 2-38. Adjusted content graph.

After you reset, and run, you should see the graph display the content of the queue over time. If it is not, you might need to turn on stats histories from the Stats > Stats Collecting > All Objects On.

#### Step 7: Add a Recorder to show the Staytime Histogram of the Queue

Following the same steps as adding a content graph, add a new Recorder to the model to be a staytime histogram. The only difference will be that you will select the "Staytime" option (see Figure 2-39) in the Data to Capture of the recorder's parameters.

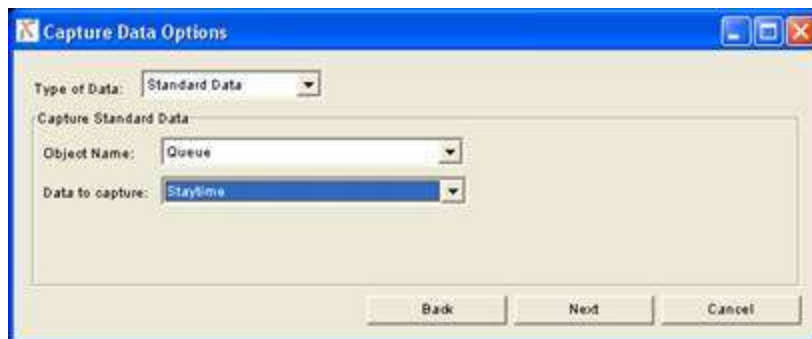


Figure 2-39. Selecting the "Staytime" option

Place the recorder just to the right of the content graph. Select the properties as shown in step 6, rotate the graph, and change the height. Reset, and run, and the graphs should look something like Figure 2-40.

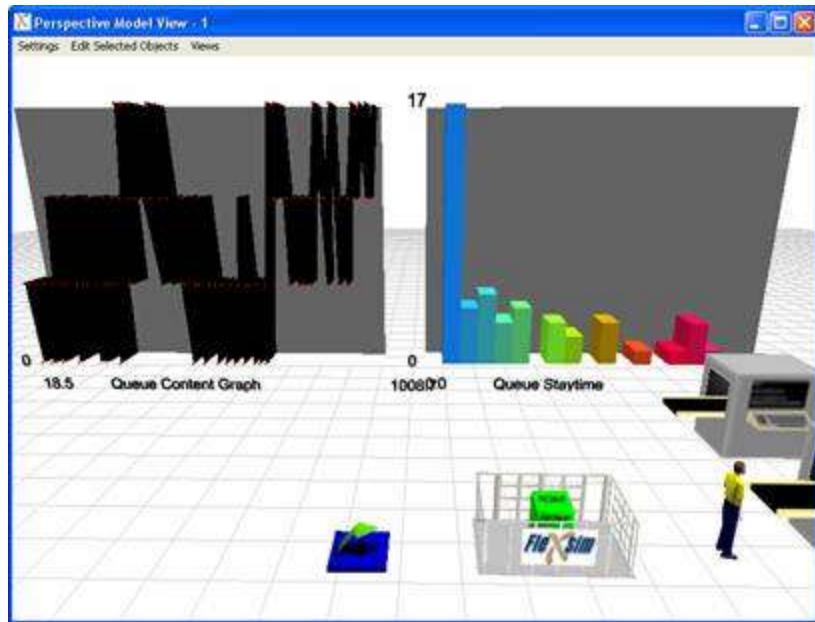


Figure 2-40. Content and Staytime graphs

#### Step 8: Add a State Pie Chart for each Operator

Follow the same procedure outlined in steps 3-5 to add a state pie chart for each operator. The only difference is to select the "State" option in the Data to capture field (see Figure 2-41).

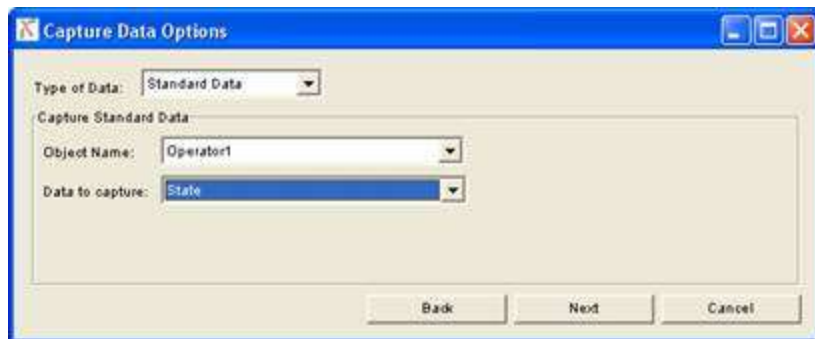


Figure 2-41. Selecting the "State" option

Size both graphs to be 5 by 5 in their properties windows (see Figure 2-42).



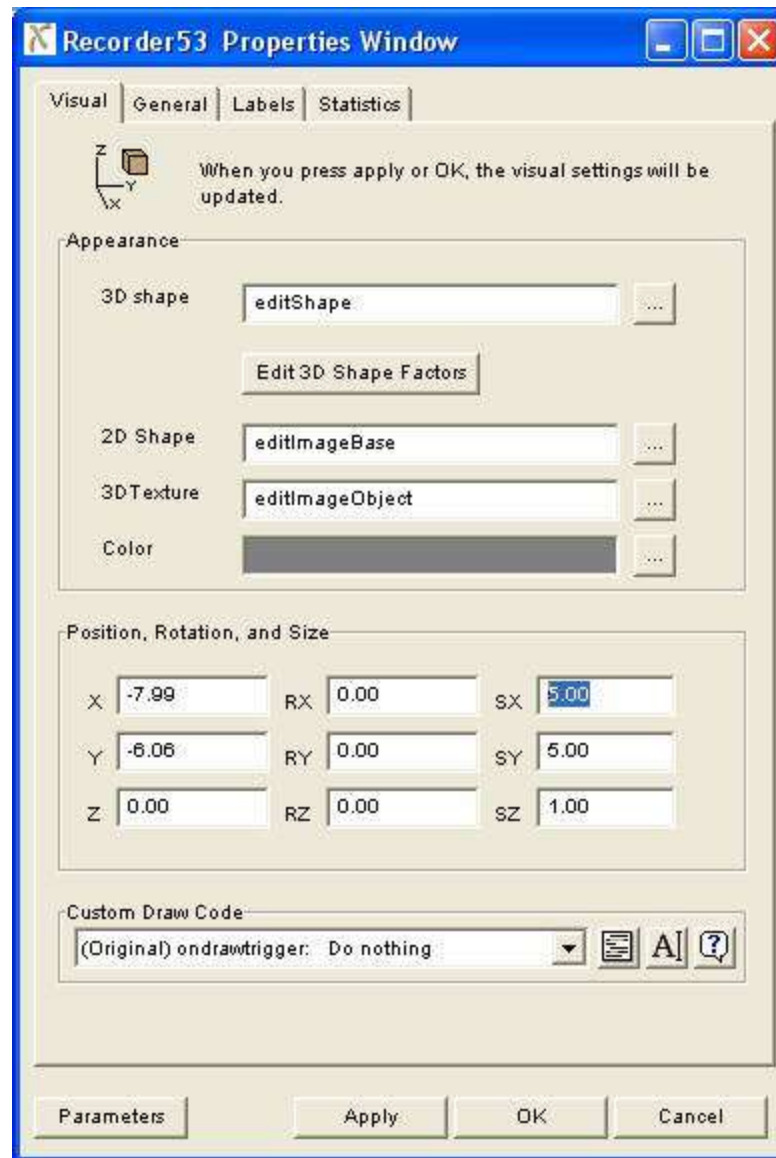


Figure 2-42. Sizing the graphs to "SX" 5 and "SY" 5

We will leave these two pie graphs flat on the floor. We do not need to change their rotation values.

When reset and run, the pie charts should look something like Figure 2-43.

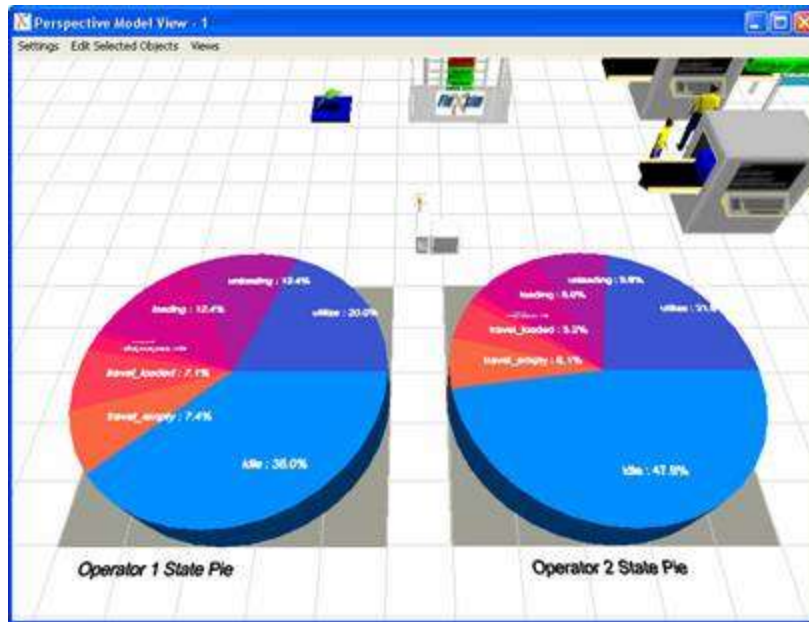


Figure 2-43. State pie charts for Operator 1 and Operator 2

### Step 9: Adding 3D Text to the Model

Another way to add information to the model that can show performance measures while the model is running is to place 3D text at strategic points in the layout. This is done using the VisualTool object, selecting the "Text" option for the Visual Display. In this model we will add 3D text to show the average wait time of flowitems in the "Conveyor Queue".

Drag out a "VisualTool" object into the model and place it by the conveyor queue (see Figure 2-44).

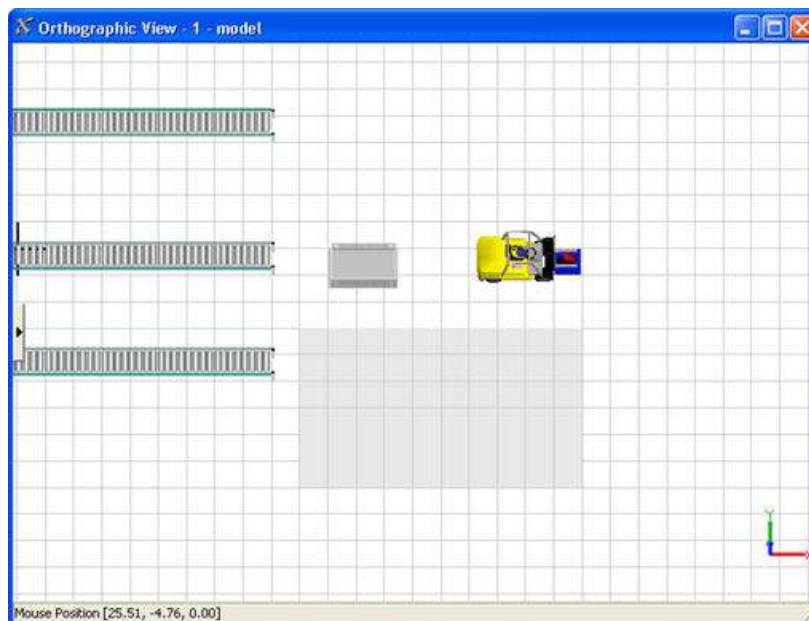


Figure 2-44. VisualTool Object

The VisualTool default display is a plane showing the Flexsim logo. Double-click on the VisualTool to bring up its Parameters Window (see Figure 2-45).

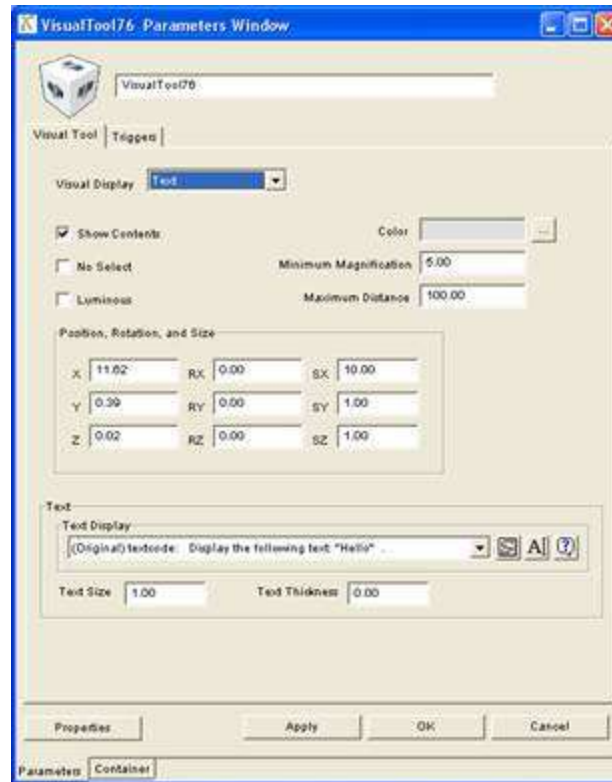


Figure 2-45. VisualTool Parameters

Select the "Text" option for the Visual Display. Now you can define the Text parameters. In the Text Display pick list select the option for "Display Avg StayTime" (see figure 2-46).

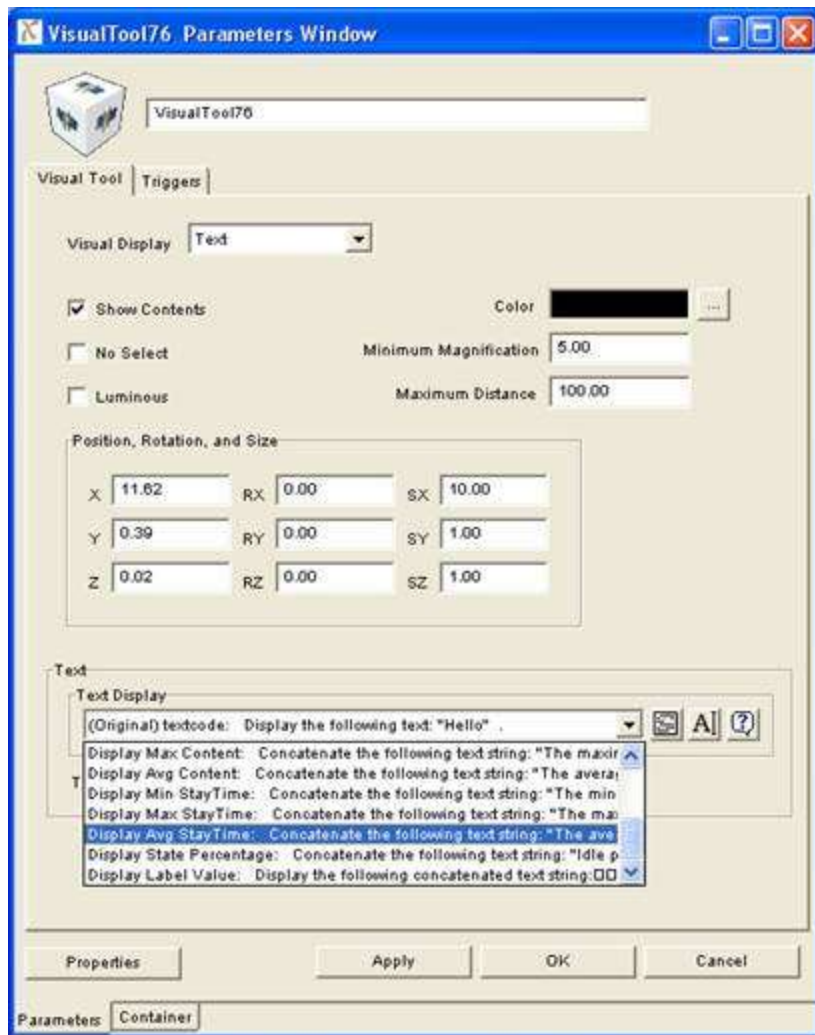



Figure 2-46. Text Display pick list

Then select the code template button

 to change the displayed text to read "The average staytime of the Conveyor Queue is:" as shown in Figure 2-47.

```

Display Avg Staytime
Text: "The average stay time of the Conveyor Queue
is: "
Object: centerobject(current,1)

```

Figure 2-47. Defining the 3D text display

You will notice that at the end of the text display string that there is a reference to the "centerobject(current,1)" statement (see Figure 2-47). This reference is used to tell the VisualTool to look for the data to show. The centerobject(current,1) simply means to display the average staytime of the object connected to the first center port of the VisualTool. This means that you will have to make a center port connection between the conveyor queue and the VisualTool objects. This is done by pressing the "S" key on the keyboard and click-and-dragging between the VisualTool and the conveyor queue (see Figure 2-48). To click on the VisualTool, click directly on the 3d text that is showing. It will not create the connection correctly if you click on white space between the letters.

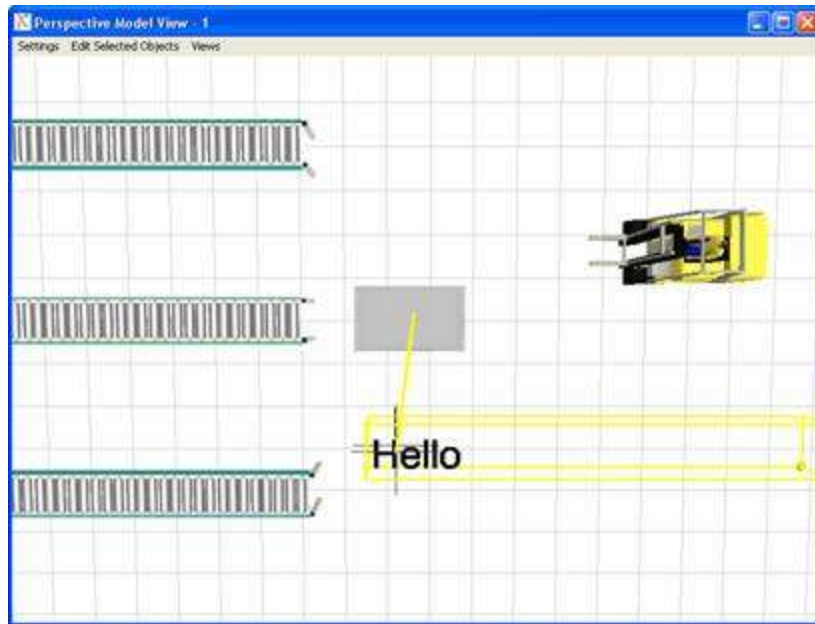


Figure 2-48. Connecting the VisualTool with the Conveyor Queue

You will see the text in the model view (see Figure 2-49).

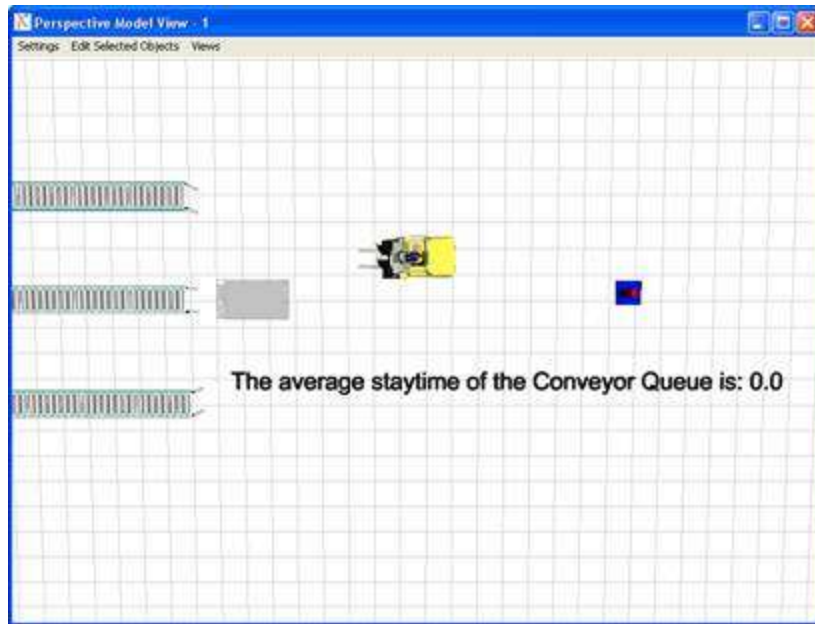


Figure 2-49. 3D text in the model view

At this point you may want to adjust the display of the text. The text size is set to 1 by default, and you may want to make it smaller. You may also want to have the text hover over the queue.

To make the text smaller, type the desired size, 0.5, in the text parameters of the VisualTool (see Figure 2-50). Also adjust the thickness to 0.1 to give the text a 3D appearance.

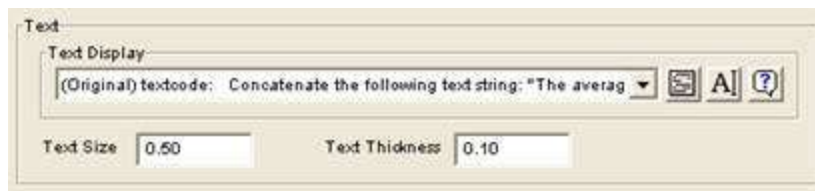


Figure 2-50. Adjusting the text size and thickness

At the bottom left of the VisualTools Parameters Window, select the "Properties" button to open the properties window (see figure 2-51).



Figure 2-51. The Properties Button

In the Properties Window, rotate the text to 90 in the "RX" field (see Figure 2-52).

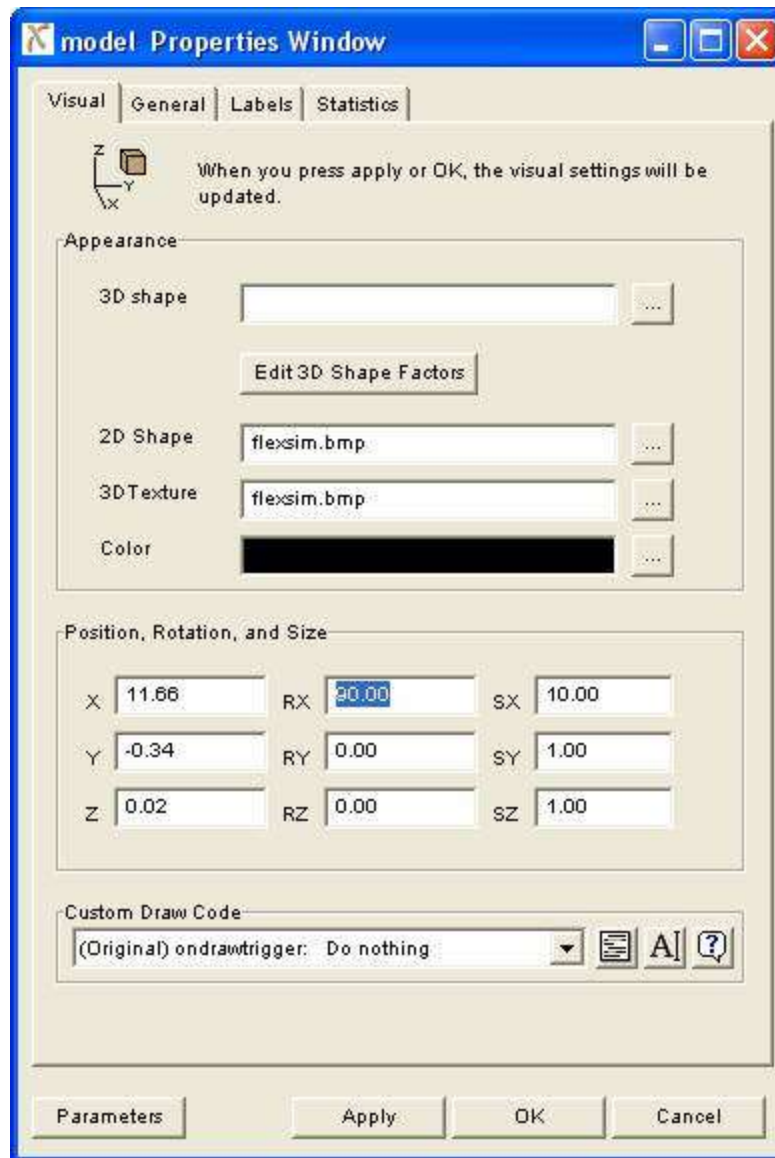


Figure 2-52. Rotate the text to 90 degrees

Press "OK" on the properties and parameters windows. The text will now be rotated in the model. Use your mouse to select and position the text as desired. Remember that the height of the text can be controlled by selecting the text with both the left and right mouse buttons and moving the mouse forward and back, or selecting the text and then rolling the mouse wheel to move the text up or down (see Figure 2-53).





Figure 2-53. Positioning the 3D text

#### Step 10: Reset, Save and Run

When the text is placed where you want, reset and save the model. You are then ready to run the model and look at the graphs, charts, and 3D text you have just added (see Figure 2-53).

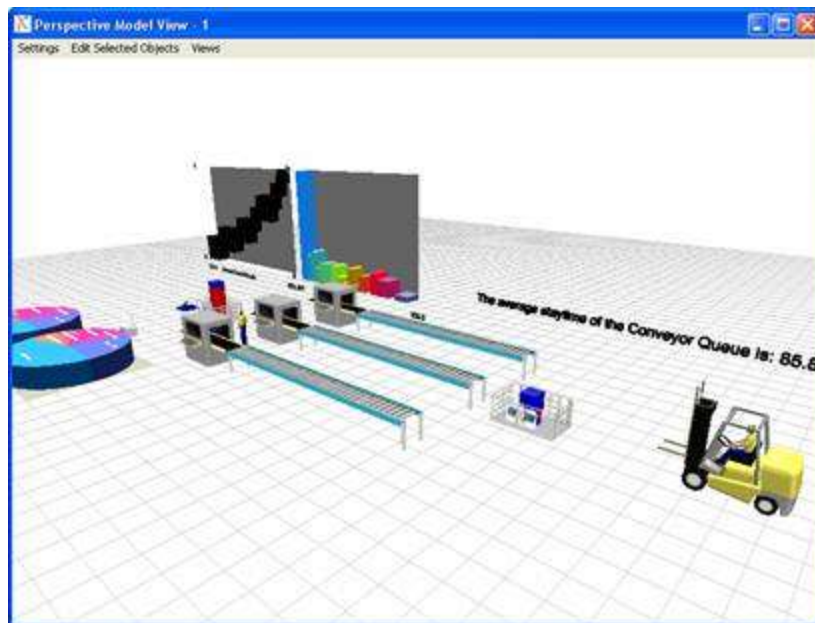


Figure 2-54. The completed model

This ends the "Model 2 Extra Mile" lesson. As you can see, it is very easy to add powerful 3D reporting visuals to your simulation models.



To continue the tutorial, go to Lesson 3.

## Lesson 3

### Lesson 3

#### Introduction

Lesson 3 introduces the Rack and NetworkNode objects. You will have a chance to work with spline points, conveyors, advanced statistics, and global tables. With lesson 3 you will be introduced to the Experimenter allowing you to do multiple run and multiple scenario analyses of your model. Lesson 3 will use the model from lesson 2 as a starting point. Make sure you have completed lesson 1, and lesson 2 before starting lesson 3.

Lesson 3 assumes you have worked through lessons 1 and 2, and are familiar with the tasks of working in the properties and parameters windows. In the previous lessons, almost every step was illustrated to make sure you had a complete understanding of the steps needed to build the model. In lesson 3 some of the simple tasks such as adding a new object to the model and entering basic parameters will be still be identified in the step-by-step description, but screen shots may not be provided.

**Note on using the Evaluation version of Flexsim:** If you are using the Evaluation version of Flexsim, you will not be able to complete this model. The Evaluation version of Flexsim only allows you to create a certain number of objects, and this lesson exceeds that number.

#### What You Will Learn

- How to use global tables to define routings
- How to set up a travel path network for a transporter
- How to create splines in a travel path network
- How to create a custom output report
- How to execute multiple runs of the model

#### New Objects

In this lesson you will be introduced to the Rack, NetworkNode, and SplinePoint objects.

#### Approximate Time to Complete this Lesson

This lesson should take about 45-60 minutes to complete

## Flexsim Software Concept Learning

### Spline Control Points

Spline control points are used in Flexsim when laying out a travel path network. Flexsim uses spline technology to give you a convenient method to add curves, inclines, and declines to NetworkNode paths.

When two NetworkNodes are placed in the model view and connected together by click-and-dragging with the "A" key, a black line will be displayed (Figure 3-2). There are also two green boxes with arrows at about 1/3 and 2/3 of the way down the line.



Figure 3-2. Connecting NetworkNodes

These green boxes indicate the attributes of the path going in the indicated direction. Green means it is a passing path, yellow means it is a nonpassing path, and red means it is a "no connection" or in other words it is a one-way path going the other way. To switch between these colors, you can right click on the node and select an option. (Figure 3-3). You can also make the path a curved path by selecting the "Curved" option in the drop down menu. This will create two spline control points that you can move to create a curved path. You can also configure how connections are made by default using the Travel Networks tool panel in the ortho/perspective view's toolbar.

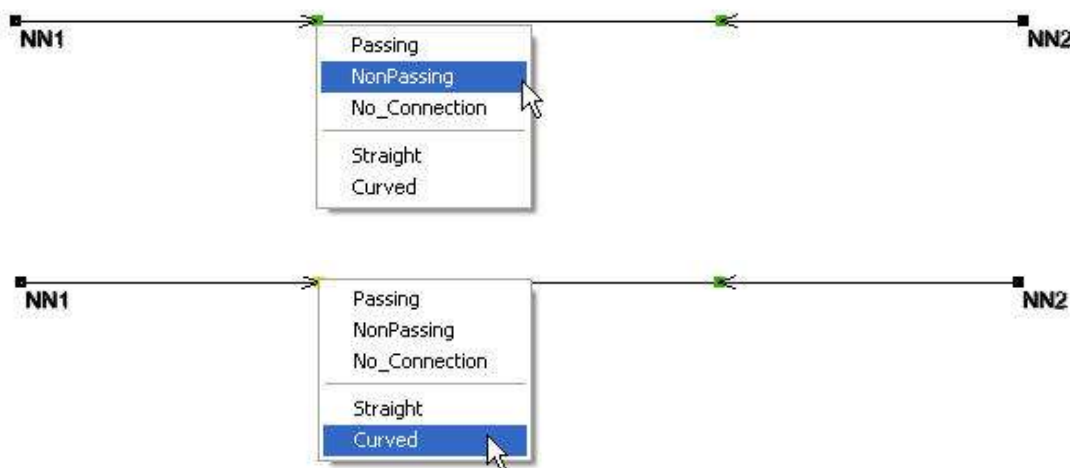


Figure 3-3. Editing path attributes

### Spline Control Point parameters

Once you have created a curved path, move the small control points with the mouse.

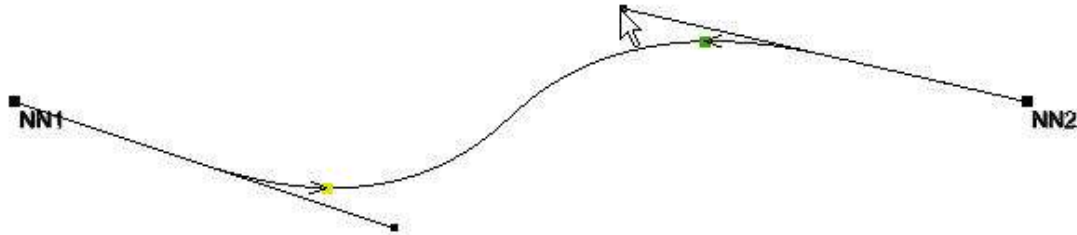


Figure 3-4. Selected spline control point.

To change the Z height of the spline control point, click on it and roll the mouse wheel up and down. (Figure 3-5).

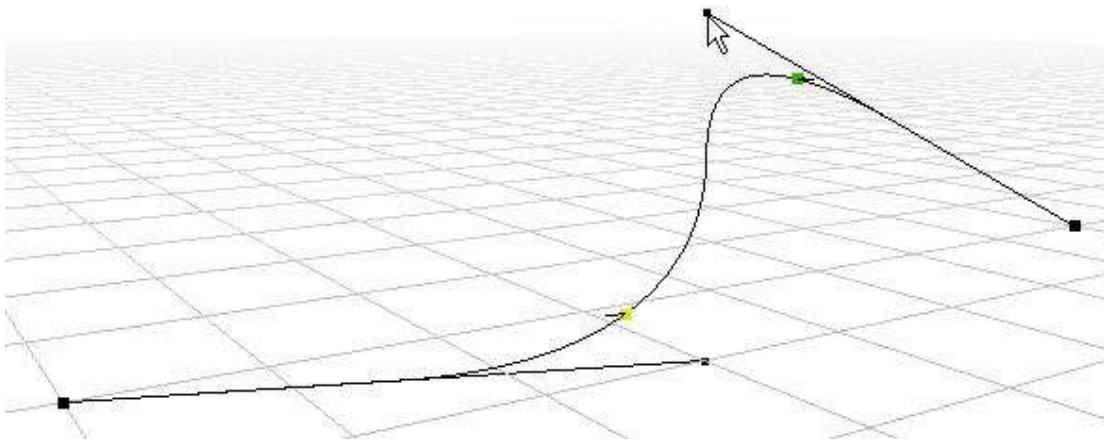


Figure 3-5. Changing the Z height of the spline control point

NetworkNodes can be configured to specify the direction of the path. Again, you can use the right-click menu on the colored box, or, for a quicker method, you can hold down the "X" key and click on the colored box. (Figure 3-9).




Figure 3-9. Single direction path

When a path has been configured using spline paths, the travelers that use the path will automatically follow the spline that has been defined. The display of the spline control points, as well as the colored boxes, can be toggled on and off by holding down the "X" key and clicking on one of the black boxes, the NetworkNodes, in the path network (Figure 3-10). Multiple "X" clicks will toggle between several different showing modes for the network.



Figure 3-10. spline control points turned off with "X" click on the NetworkNode

### The Model Tree View

The model tree view is used in Flexsim to explore the model structure and objects in detail. To access the model tree view select the  button from the toolbar. The model tree view will then be displayed (Figure 3-11).

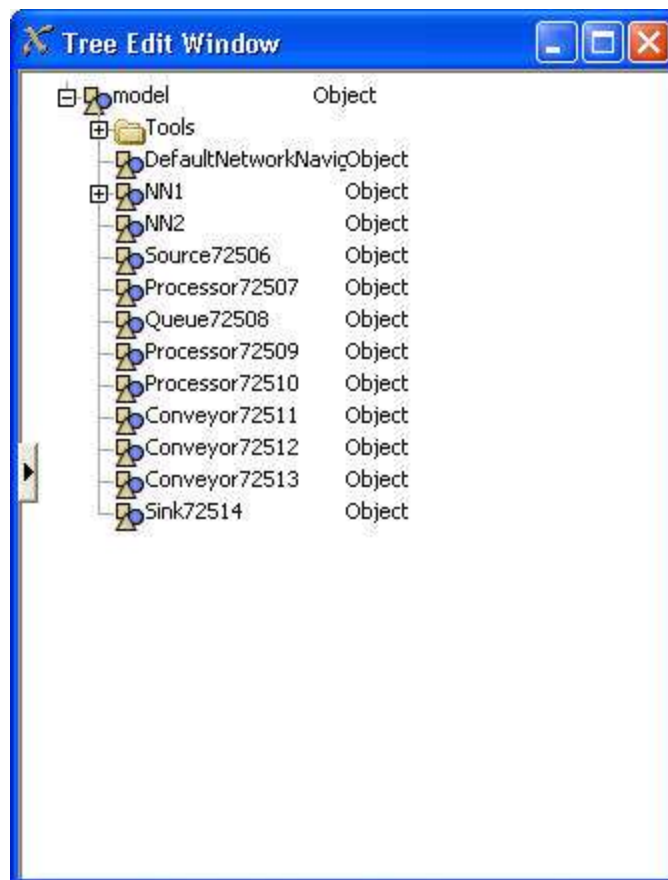


Figure 3-11. Model tree view

The model tree view is a view window that provides many unique features. In this view you can:

- Customize Flexsim objects using C++ or Flexscript

- View all object data

- Access the properties and parameters windows

- Edit the model, delete objects, and modify all data

If you follow a few simple navigation rules you will find the tree view one of the most versatile views within Flexsim. The underlying data structure in Flexsim is contained in a tree. The many edit windows within Flexsim are simply graphical user interfaces (GUIs) that display filtered data from the tree. Since all tree views in Flexsim work the same way, once you understand how the tree view works you will be able to navigate and understand the structure of any tree view that is accessible.

### Tree View Basics

Flexsim has been designed to hold all data and information in a tree structure. This tree structure is the core data structure for all of Flexsim's object oriented design. Those who are familiar with C++ object oriented programming will immediately recognize Flexsim's tree view as the C++ standard for object oriented data management.

There are several symbols used in the tree view that will help you understand the structure as you navigate the tree.

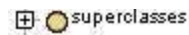
The entire MAIN tree view is referred to as a project. The library and model are contained in a project. The VIEW tree contains all the views and GUI definitions. When a session is saved the MAIN tree and the VIEW tree are saved together.



The folder icon identifies main components of the complete project. The model is a component of the main project. The library is another component of the main project.



The object icon is used to represent Flexsim objects in the tree view.




The node icon is used to specify data nodes within an object. Data nodes can have additional data nodes placed inside them. If a data node has a "+" just to the left of the icon it will contain one or more additional data nodes. Data nodes can hold numeric or alphanumeric values.



Certain data nodes are specified as C++ data nodes that hold C++ code. C++ code can be entered directly in a C++ data node. This code is compiled when the Compile button is pressed.



Data nodes can also be specified as a "Flexscript" node. This node can contain Flexscript code and will be auto-compiled during the running of the model.

Flexscript commands are pre-compiled C++ functions. Flexscript commands can be viewed by selecting the  **Commands** button in the toolbar (Figure 3-12). Most Flexscript commands are also usable in C++ code.

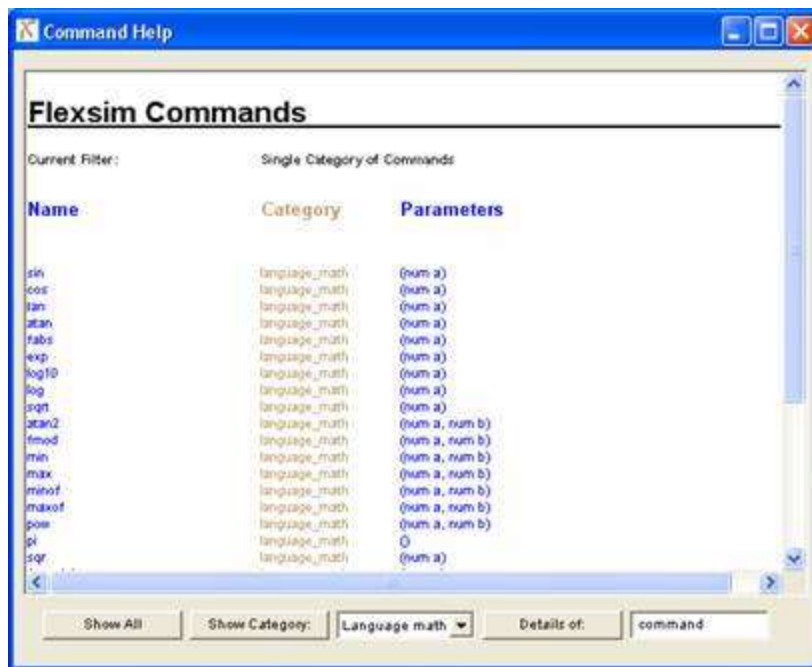
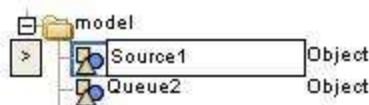



Figure 3-12. Flexscript commands

When you select an object in the tree view by clicking on the icon with the mouse, the tree view will display the object as follows:



A highlighting box will be placed around the object icon and an expand tree symbol  will be placed to the left of the object icon. If you select this expand tree symbol the data nodes for that object will be displayed as shown in Figure 3-13.

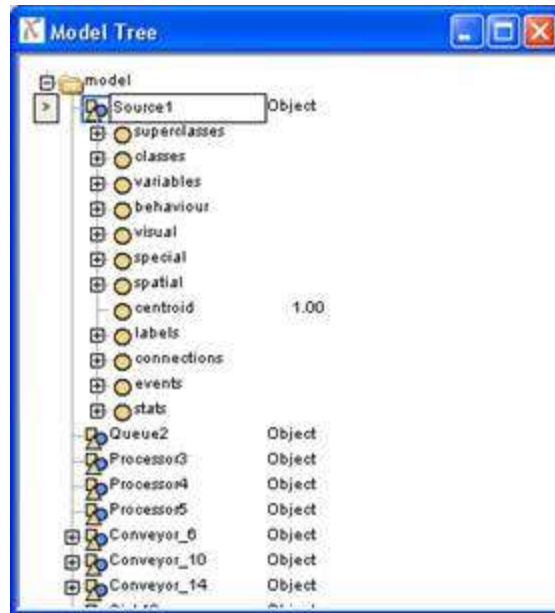


Figure 3-13. Expanded object tree view

As objects and data nodes are expanded, the tree view can quickly grow to be outside the viewing limits of the tree view window. Flexsim allows you to move the tree around in the window by using the mouse. To move the tree around in the window just click-and-drag on the left side of the tree, or use the mouse wheel to scroll up and down.

Data nodes can be expanded by clicking on the "+" to the left of the node icon. Since data nodes can have values or text you will see the text information or the data values to the right of the node (Figure 3-14).

If you select on an object or data node you may not be able to move the tree. Click a spot in the view that is blank, then drag the mouse to move the tree. You can also use the mouse wheel or PageUp PageDown buttons to move the tree up and down.



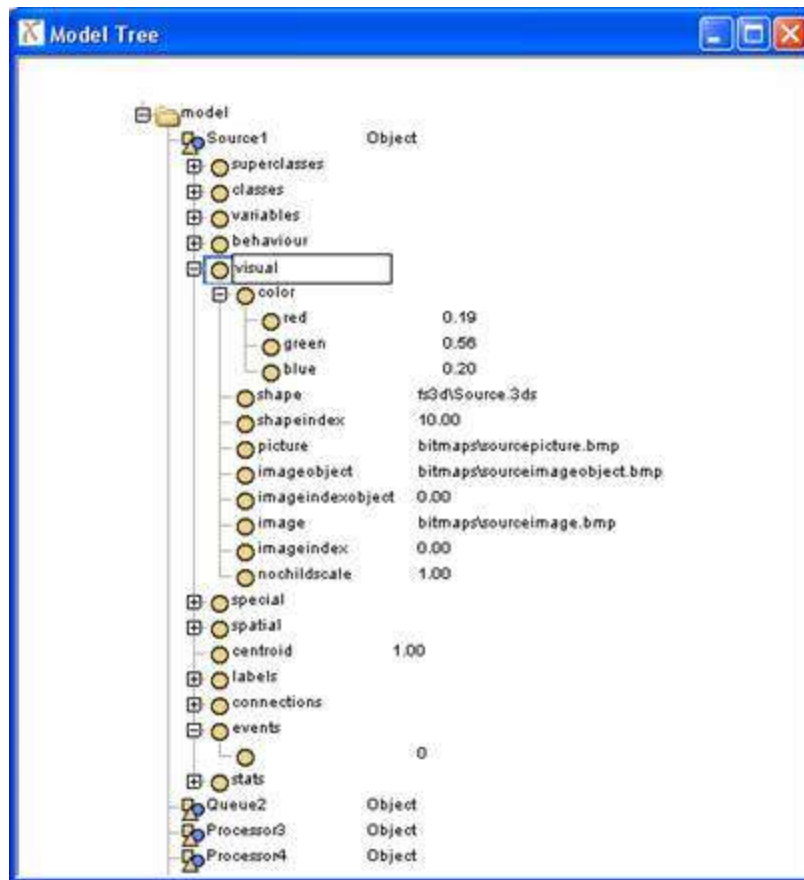


Figure 3-14. Text and value data nodes

Data can be edited directly in the tree by selecting the node you wish to edit. If it is a numeric data node you will be able to edit the number in the field (Figure 3-15). If it is a text data node you will be given a text edit field on the right side of the window to edit the text (Figure 3-16).

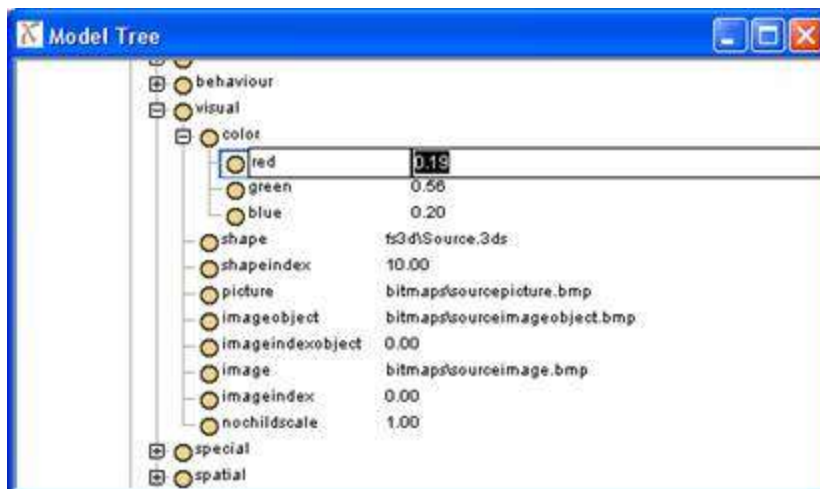


Figure 3-15. Editing of a number data node

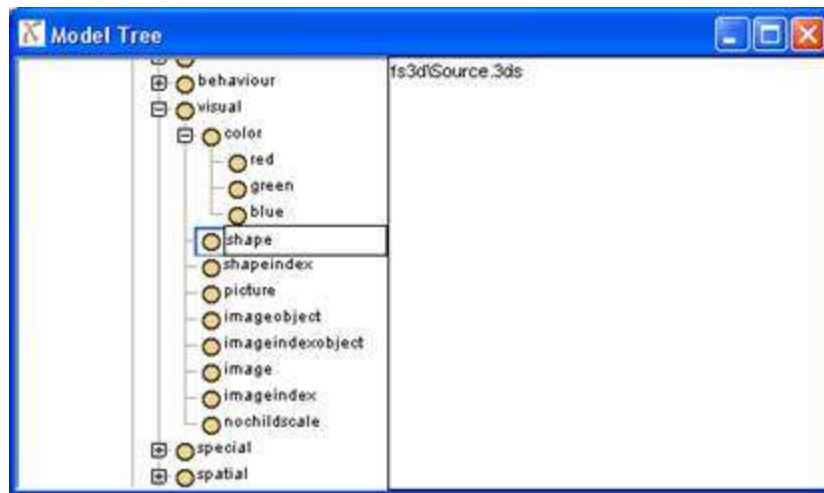



Figure 3-16. Editing a text data node

As you can see, the tree is the repository of all data for the model. The properties and parameters windows are used to provide a more user-friendly way to manipulate the data in the tree. It is possible to completely edit your model from the tree, but it is recommended that you use the parameters and properties windows to avoid inadvertent deletion of model data. The properties and parameters windows are accessible in a tree view by right clicking or double clicking on the object icon  much like in an orthographic window.

## Model 3 Description

In model 3 the sink will be replaced with 3 racks that will be used to store the completed flowitems prior to shipping (Figure 3-1). You will change the physical layout of conveyors 1 and 3 to bend at their ends to that flowitems are conveyed closer to the queue. Using a global table for reference, all itemtype 1 flowitems will be sent to rack 2, all itemtype 2 flowitems will be sent to rack 3, and all itemtype 3s flowitems will be sent to rack 1. Using the networknode object, you will set up a path network for the fork truck to use as it transports flowitems from the conveyor queue to the racks. You will also set up a multiple run simulation using the Experimenter to show statistical variance and calculate a confidence interval for key performance measures.

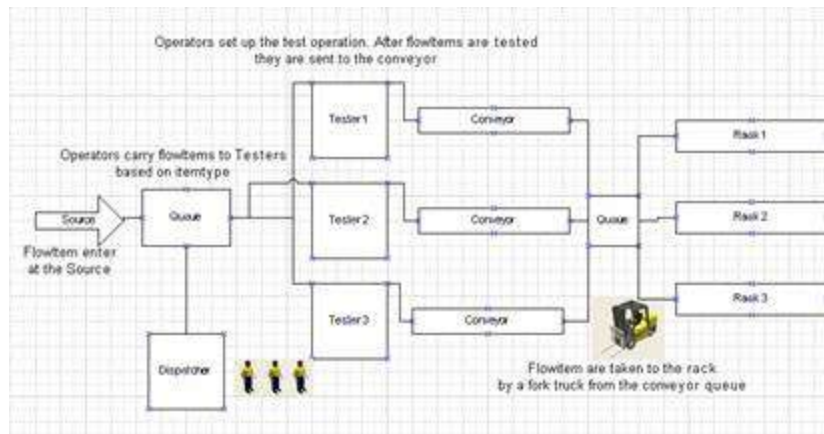


Figure 3-1. Model 3 diagram

## Model 3 Data

Modify conveyor 1 and 3 to convey flowitems closer to the conveyor queue.

Routing from conveyor queue to racks: Use a global table to specify the routing for flowitems as follows:

Itemtype 1 to rack 2

Itemtype 2 to rack 3

Itemtype 3 to rack 1

Set up a path network for the fork truck to travel on between the conveyor queue and the racks.

Set up a flypath for a fly-through model presentation.

## Step-By-Step Model Construction

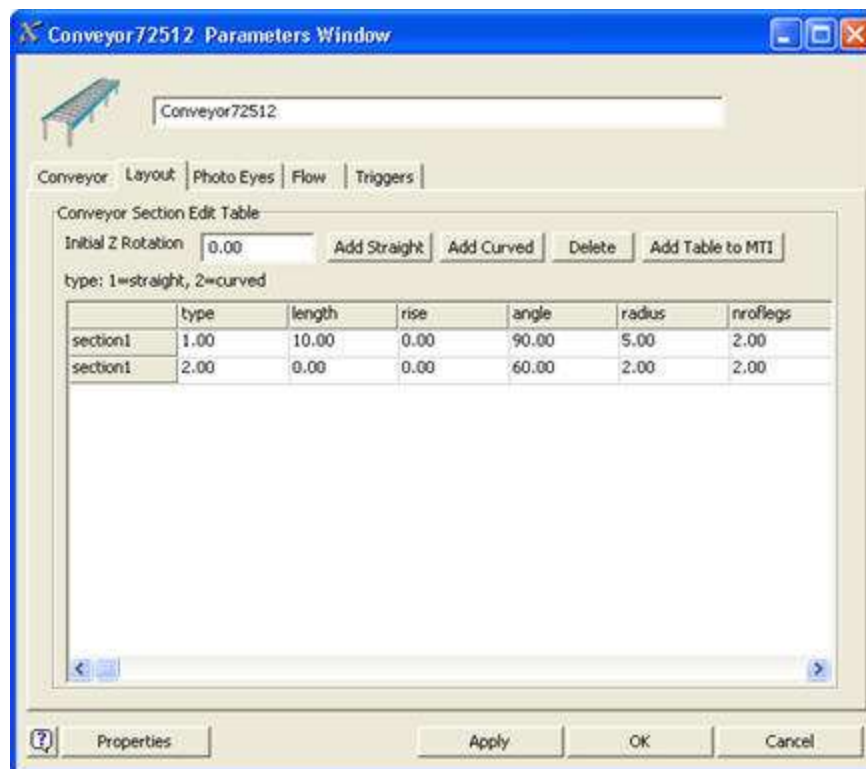
### Building Model 3

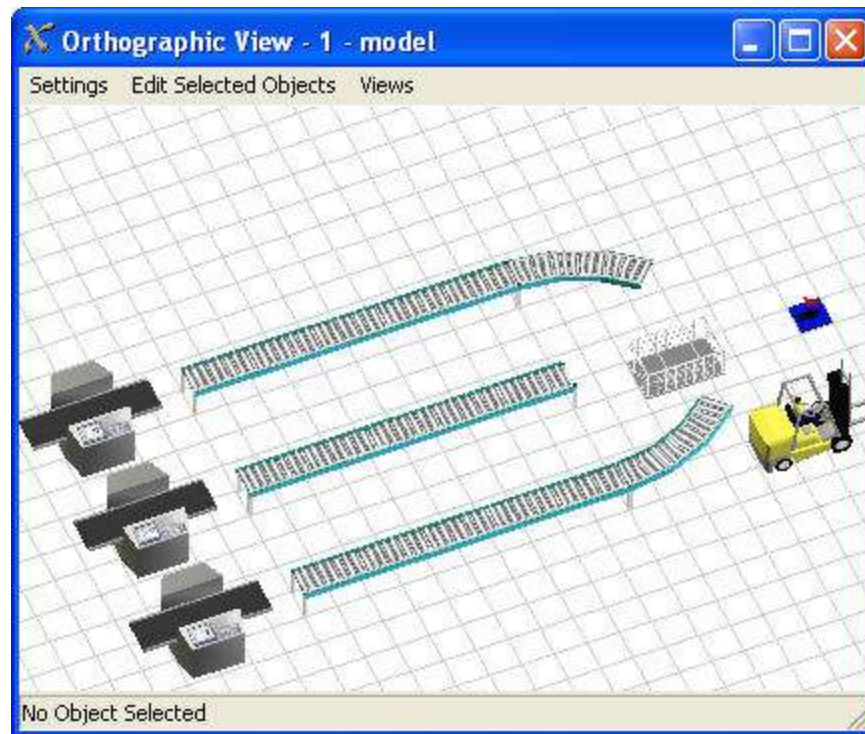
To start building model 3, you will need to load model 2 from the last lesson.

#### Step 1: Load Model 2

#### Step 2: Reconfigure the layout of conveyors 1 and 3.

Using the Layout tab on the parameters window for conveyors 1 and 3, change the layout so that the conveyors have a curved section at the end to convey the flowitems closer to the conveyor queue. You will need to add at least 1 additional curved conveyor section to do this. Note that the "type" for section 2 has a value of 2 signifying it is a curved section. For type 1 sections, the length, rise and number of legs are applicable. For type 2 sections, the rise, angle, radius and number of legs are applicable. You might want to experiment with this Layout tab to create complex curves and rises. Have some fun!






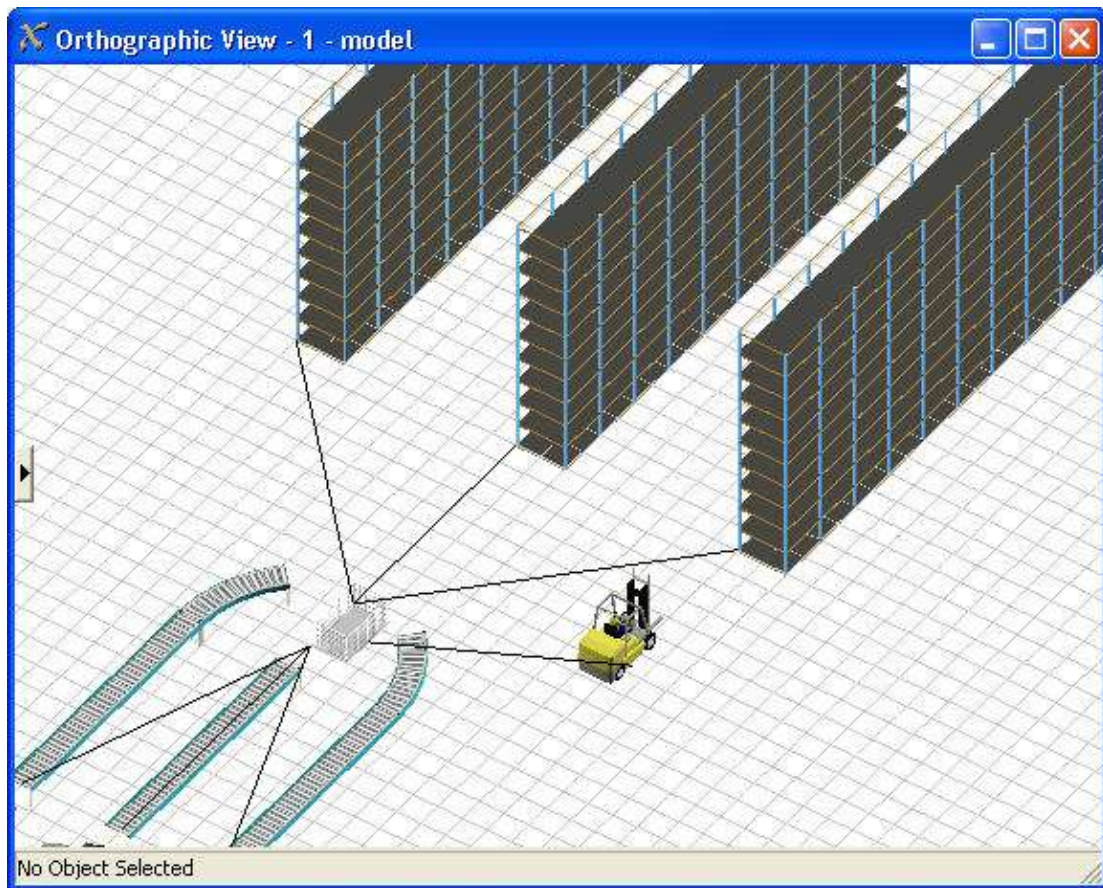
### Step 3: Delete the sink

To prepare the model for adding the racks, the final sink used in model 2 needs to be deleted. This is done by clicking the sink to highlight it yellow, and then press the "Delete" key on your keyboard. When an object is deleted, all port connections to and from that object are deleted as well. Beware that this may affect the port numbering of those objects that used to connect to the deleted object.

### Step 4: Add three racks to the model

Select the rack object in the library , and drag-and-drop three racks into the model. Once the racks are in the model, create port connections from the conveyor queue to each rack by pressing and holding the "A" key and then click-and-dragging a line from the queue to each rack.





Place the racks far enough away from the queue to allow the fork truck some travel distance to reach the racks.

#### Step 5: Set up the global table for flowitem routing from the queue to the racks

The next step is to set up a global table that will be used to reference which rack each flowitem will be sent to (or more accurately stated, which output port of the conveyor queue the flowitems will be sent out of). It is assumed that output port 1 was connected to rack 1, port 2 to rack 2, and port 3 to rack 3. We will send all itemtype 1s to rack 2, all itemtype 2s to rack 3, and all itemtype 3s to rack 1. Here are the steps to setting up a global table:

1. Select the main menu option Tools > Global Tables > Add.
2. In the Global Table Parameters Window, change the name of the table to "rout".

Name:

3. Set the table to have 3 rows and 1 column, then hit the Apply button.

4. Name the rows item1, item2 and item3, then fill in the values which correspond to the output port number (rack number) we want to send the flowitems

	Column1
item1	2.00
item2	3.00
item3	1.00

to.

5. Select the OK button at the bottom of the window. Select the Close button at the bottom of the Global Modeling Tools Window.

Now that the global table is defined, we can adjust the "Send To Port" option on the queue.

#### Step 6: Adjusting the "Send To Port" option on the conveyor queue

Double click on the conveyor queue to bring up its Parameters Window. Select the Flow tab. In the "Send To Port" pick list, select the option "By Global Table Lookup". Once you have selected the lookup table option, edit the template to use the table called "rout".

```
By Global Table Lookup
Table: "rout"
Row: getitemtype(item)
Column: 1
Note: The global table cell should contain the port number to send to.
```

Select the OK button to close the parameters window.

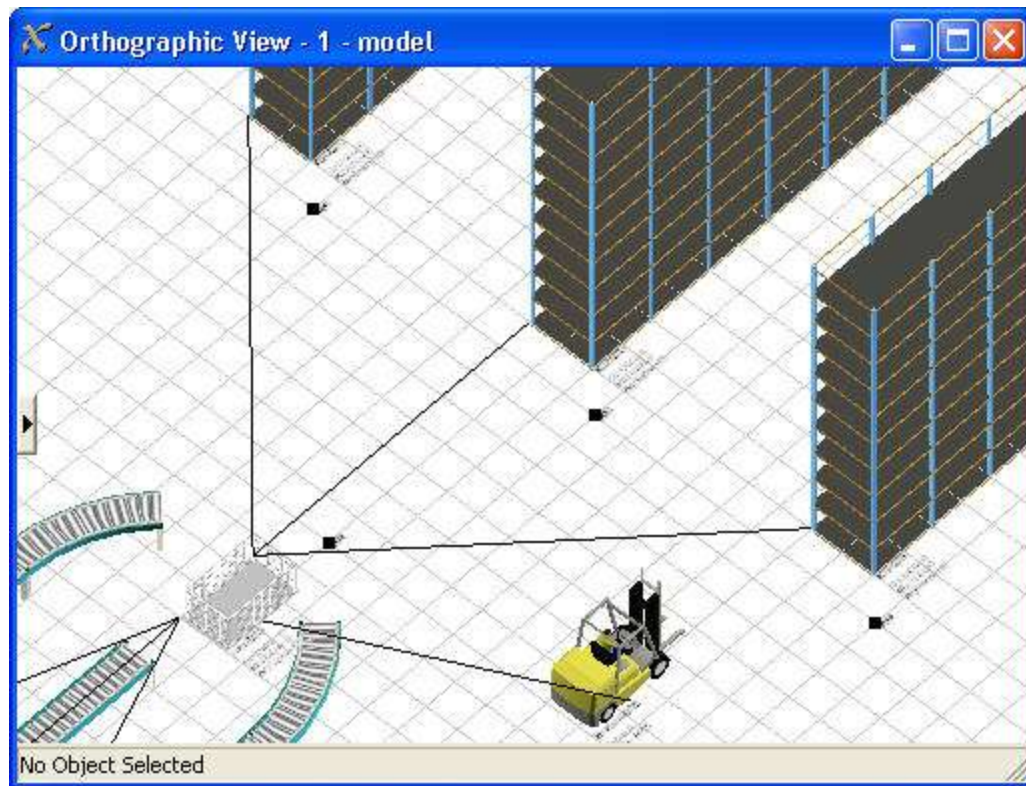
#### Step 7: Reset, save, and run

At this point it would be wise to reset, save the model, and then run the model to verify the changes added to the model. The model should run with the fork truck transporting flowitems to the racks based on the itemtype definition in the global table.

#### Step 8: Adding NetworkNodes to develop a path for the Fork Truck

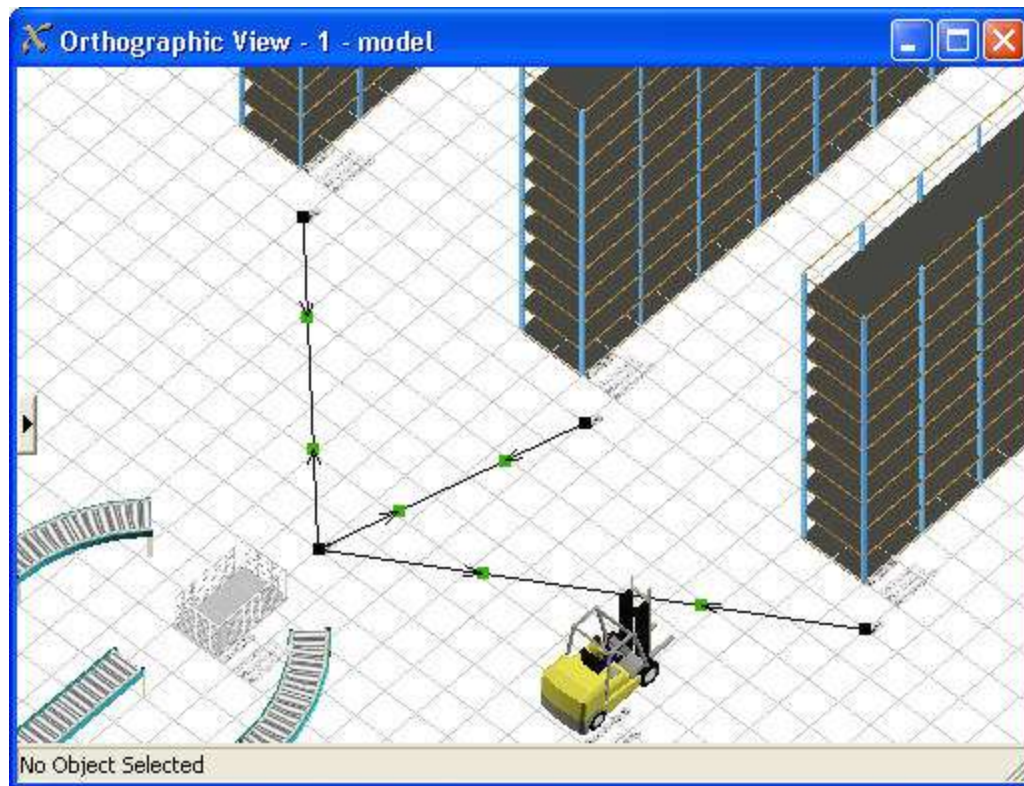
NetworkNode's are used to develop a path network for any task executer object, such as a Transporter, Operator, ASRSvehicle, Crane, etc. In the previous lessons we have used the operator and transporter to transport flowitems around the model. Up to this point we have let the task executer move freely across the model in a direct line between objects. Now we would like to confine the travel of the fork truck to a specific path as it transports flowitems from the conveyor queue to the racks. The following steps are used to set up a simple path.

1. Drag-and-drop NetworkNodes next to the conveyor queue and each of the racks. The nodes will become the pick-up points and drop-off points in the model. You may add additional nodes between these nodes, but it is not necessary.

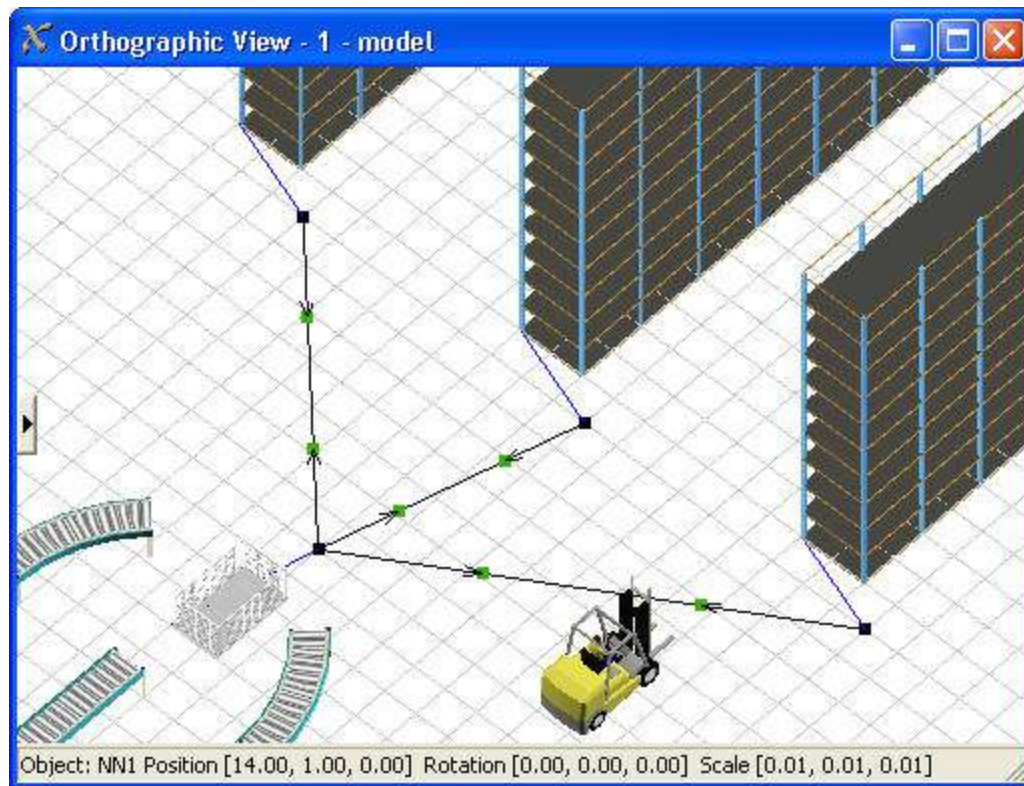


2. Connect the NetworkNodes to each other by pressing the "A" key on the keyboard and click-and-drag a line between each NetworkNode. A line will appear after the connection is made with two green indicator boxes along it, indicating that travel is possible in both directions between the two nodes.

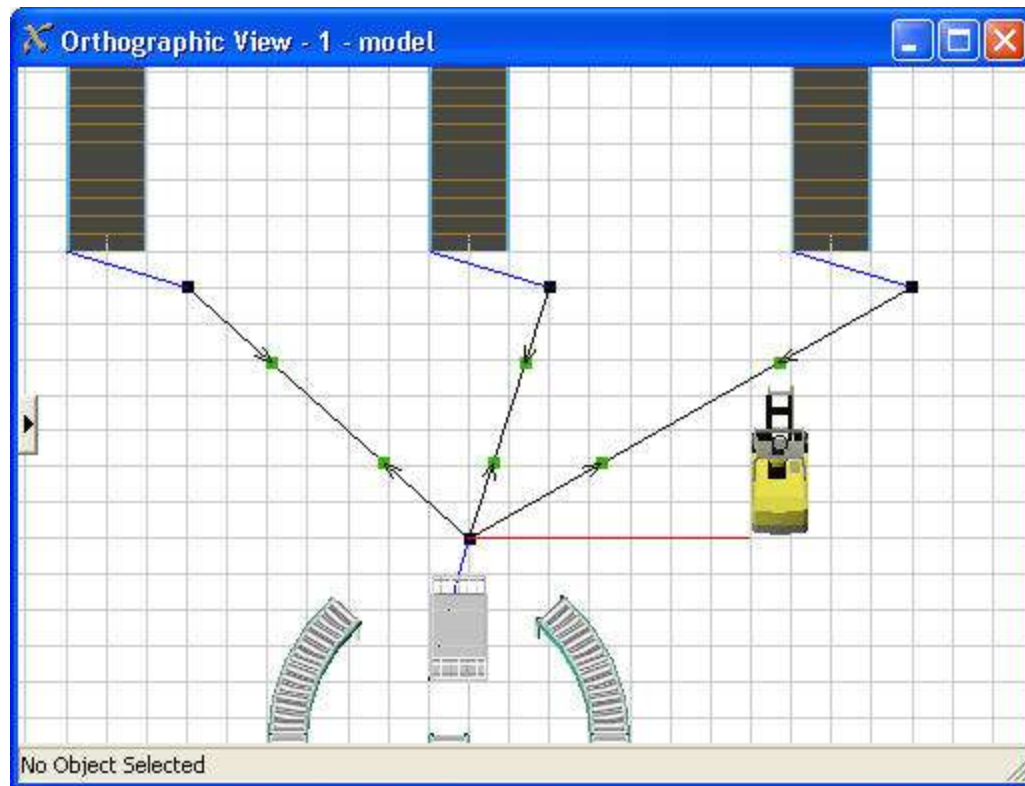




3. Now connect a node to the conveyor queue and a node to each of the three racks. This needs to be done so that the fork truck will know which NetworkNode is associated with each of the pick-up and drop-off locations in the model. This connection is also made by pressing the "A" key on the keyboard and then click-dragging a line from the NetworkNode to the object. A thin blue line will appear when the connection is made correctly.



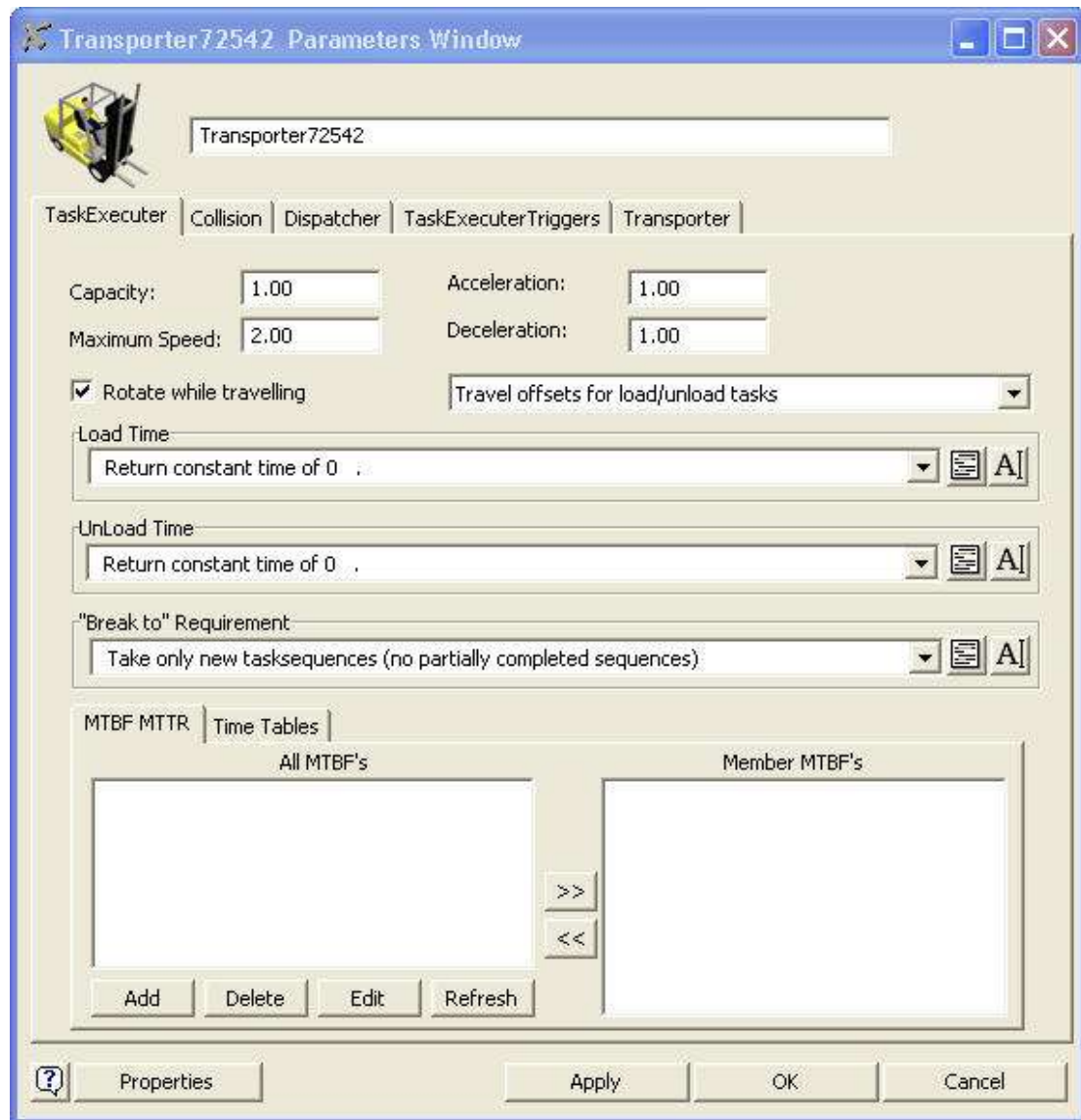
4. The last step is to connect the fork truck to the node network. In order for the fork truck to know that it has to use the path, it must be connected to one of the NetworkNodes in the path network. This is done by pressing the "A" key on the keyboard and then click-and-drag a line from the fork truck to just one of the NetworkNodes. A red line will appear when the connection is made. The node you choose to connect the fork truck to will become the starting location for the fork truck every time you reset and run the model.

**Step 9: Reset, save, and run the model**

Now it would be a good idea to reset, save, and then run the model to make sure the fork truck is using the path network.

**A word about offsets**

As the model runs, you will notice that the fork truck will travel off the NetworkNode when it picks up or drops off a flowitem. This is a result of having the "Travel offsets for load/unload tasks" selected in the fork truck's parameters.

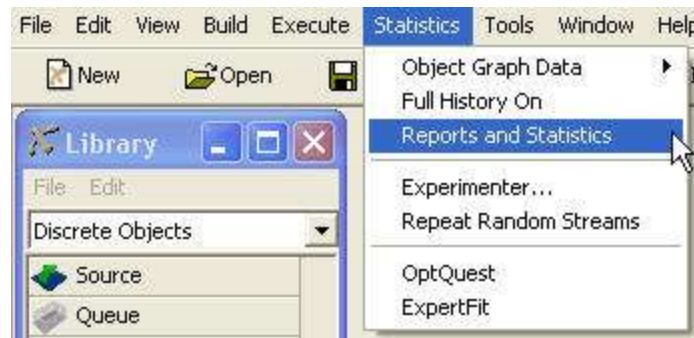


Offsets are used by the fork truck to locate where the flowitem needs to be picked up, or dropped off in the object. This allows the fork truck to travel into the queue and pick up the box, and travel to the specific cell in the rack to drop off the box. To force the fork truck to stay at the NetworkNode and not to travel off the path network, just uncheck the travel offsets box.

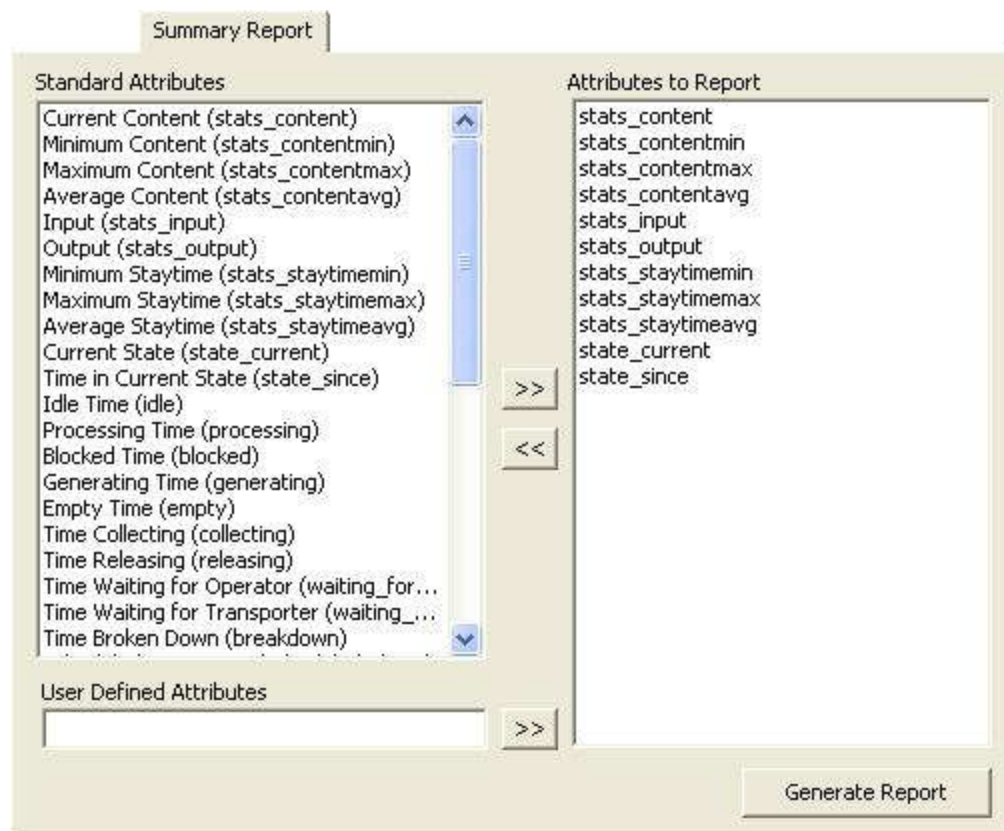
Path networks automatically use Dykstra's algorithm to determine the shortest distance to travel between any two nodes in the network.

#### Step 10: Using the report to view output results

To view summary results of the simulation, select the main menu option Statistics > Reports and Statistics.



Go to the Summary Report tab of the Reports and Statistics dialog window.



To generate a basic report, press the Generate Report button. If you have any other attributes you would like reported, you can add them using the interface provided. The report will be exported to a csv file and automatically displayed in Excel or whichever default program is set up to open csv files on your computer..



Content	min	avg	max	Throughput	Staytime
Source1	1	0	1	0	224
Queue2	25	0	23.11	189	0
Processor	1	0	0.51	1	70
Processor	1	0	0.54	1	74
Processor	1	0	0.83	1	62
Conveyor6	0	0	0.37	2	70
Conveyor7	1	0	4.8	17	73
Conveyor8	20	0	17.81	23	32
Dispatcher	0	0	0	0	0
Operator1	0	0	0.18	1	130
Operator2	0	0	0.1	1	69

To create a state report, go to the State Report tab of the Reports and Statistics dialog window, and press Generate Report.

State	idle	processing	busy	blocked	generating	empty	collecting	releasing	waiting	for waiting	for breakdown	scheduled	conveying	trans	empty	full
Source1	0.00%	0.00%	0.00%	60.50%	39.50%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Queue2	0.00%	0.00%	0.00%	0.00%	0.00%	0.10%	0.00%	99.70%	0.00%	0.20%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Processor	47.80%	37.10%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	2.80%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Processor	46.00%	39.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	1.50%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Processor	16.50%	28.40%	0.00%	43.50%	0.00%	0.00%	0.00%	0.00%	2.30%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Conveyor6	0.00%	0.00%	0.00%	19.70%	0.00%	65.20%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	15.00%	0.00%	0.00%
Conveyor7	0.00%	0.00%	0.00%	66.50%	0.00%	29.50%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.00%	0.00%	0.00%
Conveyor8	0.00%	0.00%	0.00%	89.80%	0.00%	8.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	2.20%	0.00%	0.00%
Dispatcher	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Operator1	49.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	6.60%
Operator2	58.30%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	4.60%

## Step 11: Running multiple runs of your model using the Experimenter

To access the Experimenter in Flexsim, select the main menu option Statistics > Experimenter.

The Simulation Experiment Control window will appear.

Simulation Experiment Control

Experimenter | Performance Measures | **Advanced**

Replications:

Simulation End Time: 3600.000    Replications per Scenario: 5.000    Current Replication: 1.000

Warmup End Time: 0.000    Number of Scenarios: 5.000    Current Scenario: 1.000

☐ Save state after each replication (states will be saved in the 'experiment' subdirectory)

Experiment Variables:

Number of Experiment Variables: 1

	Variable 1	
Path	Not Specified	
Scenario 1	0.000	
Scenario 2	0.000	
Scenario 3	0.000	
Scenario 4	0.000	
Scenario 5	0.000	

Apply    OK    Cancel

The Simulation Experiment Control window is used to run multiple replications of a given model, and to run multiple scenarios of a model. When running multiple scenarios, you can declare a number of experiment variables, and then specify the values you want these variables to be set to for each of the scenarios you want to run. Confidence intervals are calculated and displayed for each of the performance measures you define on the Performance Measures tab. For more information on the experimenter, refer to the Experimenter section of the help documentation.

This completes lesson 3. Congratulations!

## Fluid Objects Tutorial

### Fluid Object Lesson

#### Introduction

This lesson introduces most of Flexsim's Fluid Objects. You will learn how they interact with each other and how to include them in a model with the Discrete Objects. Building a model with the Fluid Objects is more involved and requires more attention to detail than a model with the Discrete Objects. For that reason, it is recommended that you feel comfortable building models with the other objects before you begin to learn about the Fluid Objects.

#### What You Will Learn

- How to model fluid material with Flexsim
- How to convert flowitems into fluid material
- How to transfer and store fluid material
- How to use level marks on a tank to control material flow
- How to mix fluid materials together
- How to convert fluid material into flowitems

#### New Objects

In this lesson you will be introduced to the FluidTicker, ItemToFluid, FluidPipe, FluidTank, FluidMixer, FluidProcessor and FluidToItem objects.

#### Approximate Time to Complete this Lesson

This lesson should take about 45-60 minutes to complete.



## Flexsim Software Concept Learning

### Flexsim Terminology

Before you start this model it will be helpful to understand some of the basic terminology of Flexsim's fluid system.

**Fluid:** Any material that is not easily or efficiently modeled with discrete flowitems. Typically, material that is measured by weight or volume is hard to model with flowitems. This is because frequently part of a unit (for example, half a gallon) can be moved by itself. There is no easy mechanism for moving half of a flowitem. Fluid material can also represent objects that are so numerous that flowitems are impractical. For example, thousands of bottles in a filling line will slow down a model that uses a flowitem for each bottle. Instead, the Fluid Objects can be used to model these bottles without the overhead that comes with the flowitems.

**Fluid Objects:** The eleven objects that are designed to handle fluid material. Nine of them cannot interact with Flexsim's Discrete objects, but two of them are designed to work as an interface between the Fluid Objects and the Discrete Objects. More information can be found [here](#).

**Tick:** The Fluid Objects send and receive material at set intervals. These intervals are called "ticks". At the end of each tick, the Fluid Objects calculate how much material they sent and received during that time period.

**Tick time:** The length of each tick. The modeler can set this value to some value that is appropriate for their model. A shorter tick time may make the model more accurate, but it may also make it slower. A longer value will be a faster model, but the cost is a loss in accuracy. It is up to each modeler to decide the optimal trade-off of speed and accuracy for their model.

**Rate:** The maximum speed at which material enters or leaves an object. Generally, the Fluid objects have both an input rate and an output rate that are separate from each other. In a few objects, the rate at which material enters will affect the rate at which it leaves. For these objects, the modeler is not given the opportunity to edit the output rate. The actual rate at which material enters or leaves is based on several factors: the output rate of the upstream object, the input rate of the downstream object, the amount of material available to send and the amount of space available in the downstream object.

**Object Rate:** This is the maximum rate at which material can enter or leave an object through all input or output ports combined. The objects typically have a separate rate for the input ports and the output ports. If, at the end of any tick, the object calculates that the amount of material it has sent or received has reached the maximum object rate, no more material will be sent or received for that tick, even if there are ports that have not yet sent or received material.

**Port Rate:** This is the maximum rate at which material can enter or leave any single port on the object. The objects typically have different port rates for input and output ports. This single value applies to all of the input or output ports. It cannot be changed to affect individual ports.

**Port Scale Factor:** This is a number that is used to change the port rate for each individual port. There is one scale factor available for every input and output port. The value for each port is multiplied by the maximum port rate to find the actual maximum rate for that port.

---

## Fluid Lesson Description

In our fluid model we will have an operator carry boxes of two different types of material into the model. These boxes will be converted into two fluids which will be transported by Pipes to two Tanks. From the Tanks the material is sent to a single Mixer which will mix the two products into a new product. That product is sent through a FluidProcessor, and then converted into flowitems which are carried by a Conveyor to a Sink. The fluid in this model will be measured in gallons, and the time will be in seconds.

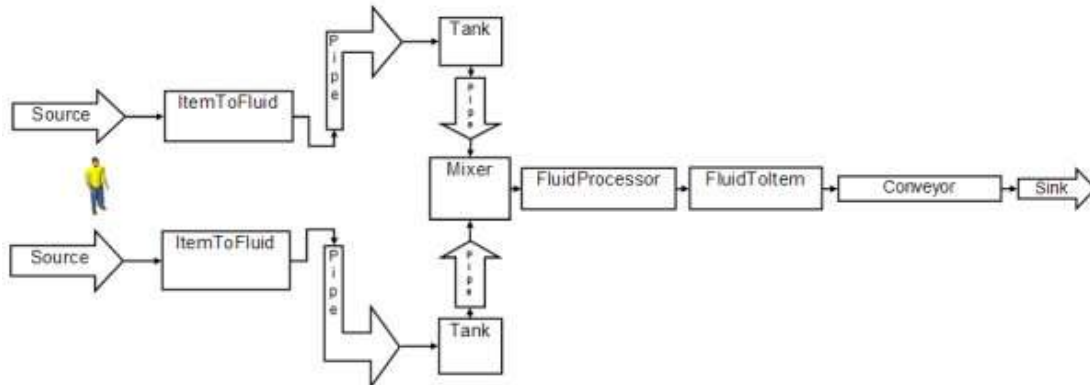


Figure 4-1 Fluid Model diagram

## Fluid Model Data

**Flowitem arrival rate:** exponential(0,10) seconds

**Maximum Content of ItemToFluid:** 20 gallons

**Fluid Units per Discrete Unit (ItemToFluid):** 10 gallons per flowitem

**Maximum Content of Pipe leading to Tank:** 20 gallons

**Transfer Rate (ItemToFluid to Tank):** 2 gallons per second

**Tank Low Mark:** 1 gallon

**Tank High Mark:** 45 gallons

**Maximum Content of Pipe leading to Mixer:** 10 gallons

**Transfer Rate (Tank to FluidToItem):** 1 gallon per second

### Mixer Steps:

Step 1: Material1, no delay time

Step 2: Material2, 10 second delay

**Mixer Recipe:**

Material1: 10 gallons, step 1

Material2: 20 gallons, step 2

**Maximum Content of FluidToItem:** 10 gallons

**Fluid Units per Discrete Unit (FluidToItem):** 10 gallons per flowitem

## Step-By-Step Model Construction

### Building the Fluid Model

To start building the fluid model, you will need to start with a new model. Do not begin with the models you saved from previous lessons. It is expected that you know how to create objects, connect their ports, and use their Parameters and Properties GUIs.

#### Step 1: Create and connect the objects required for this model.

To begin this lesson, drag out the following objects:

- 2 Sources
- 1 Operator
- 2 FluidToltems
- 2 FluidPipes
- 2 FluidTanks
- 2 more FluidPipes
- 1 FluidMixer
- 1 FluidProcessor
- 1 FluidToltem
- 1 Conveyor
- 1 Sink

Note: When you create the first Fluid Object, a Ticker is automatically created and placed at (0,0). You can move this to any point in your model where it is not in the way, but do not delete it. It is required for the Fluid Objects to work.

Arrange the objects as shown in figure 1.

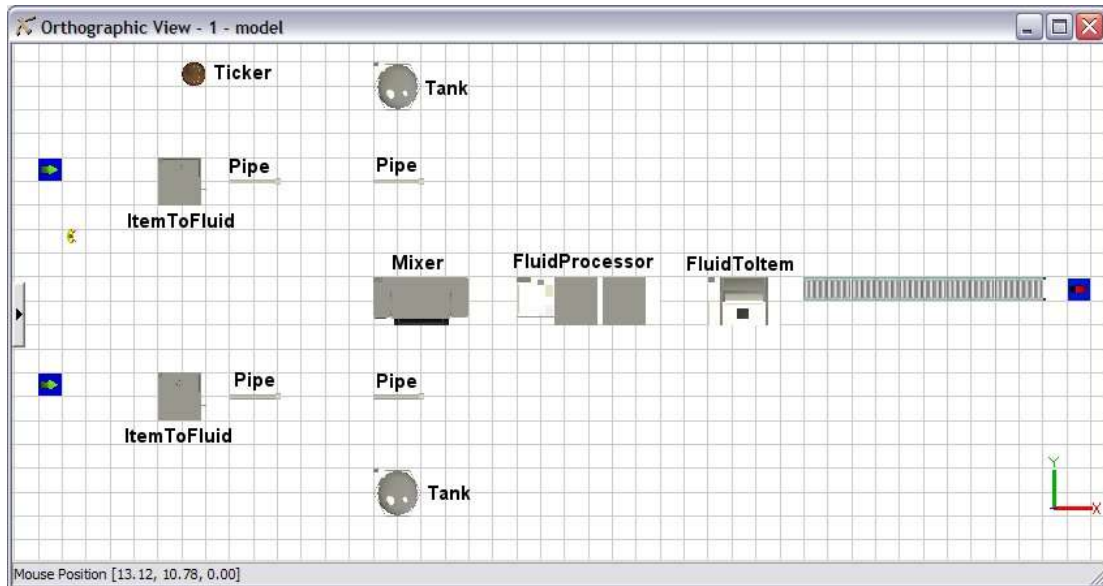


Figure 1: The layout of the required objects. The FluidObjects have been labeled.

Once the objects have all be created and positioned where you want them in the model, they need to be connected. You use the same keys to connect Fluid Objects as you do to do connect Discrete Objects: the A key creates an input/output connection and the S key creates a center port connection. Connect the objects so that there is a processing line from one of the Sources to a ItemToFluid, to a Pipe, to a Tank, to another Pipe, and to the Mixer. The Source should call for an Operator to transport the flowitem to the ItemToFluid object. There is a matching processing line starting from the other Source. The Mixer should be connected to the FluidProcessor. That should lead to the FluidToItem. From there the flowitem will travel to the Conveyor and to a Sink. See figure 2.

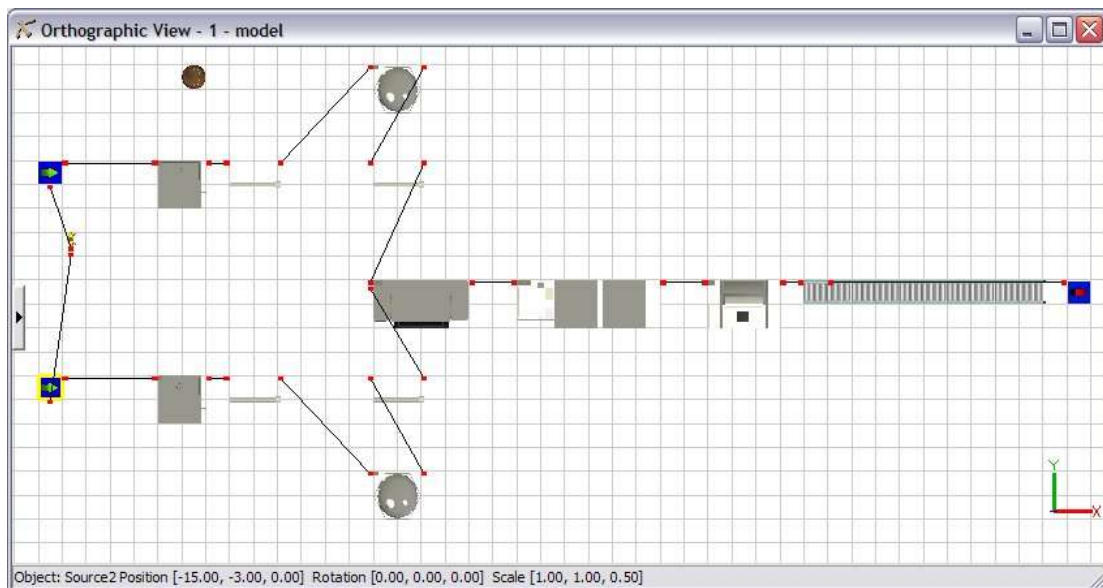


Figure 2: The objects have been connected.

**Step 2: Configure the Sources.**

The default inter-arrival time for the Sources will work well for this model. They just need to call an Operator to transport the flowitems they create to the ItemToFluid objects. In their Parameters GUIs, in the Flow tab, check the box called "Use Transport". Make sure the Operator is connected to center port 1 of each Source, because this is the port that will be used to call the Operator by default. See Figure 3.

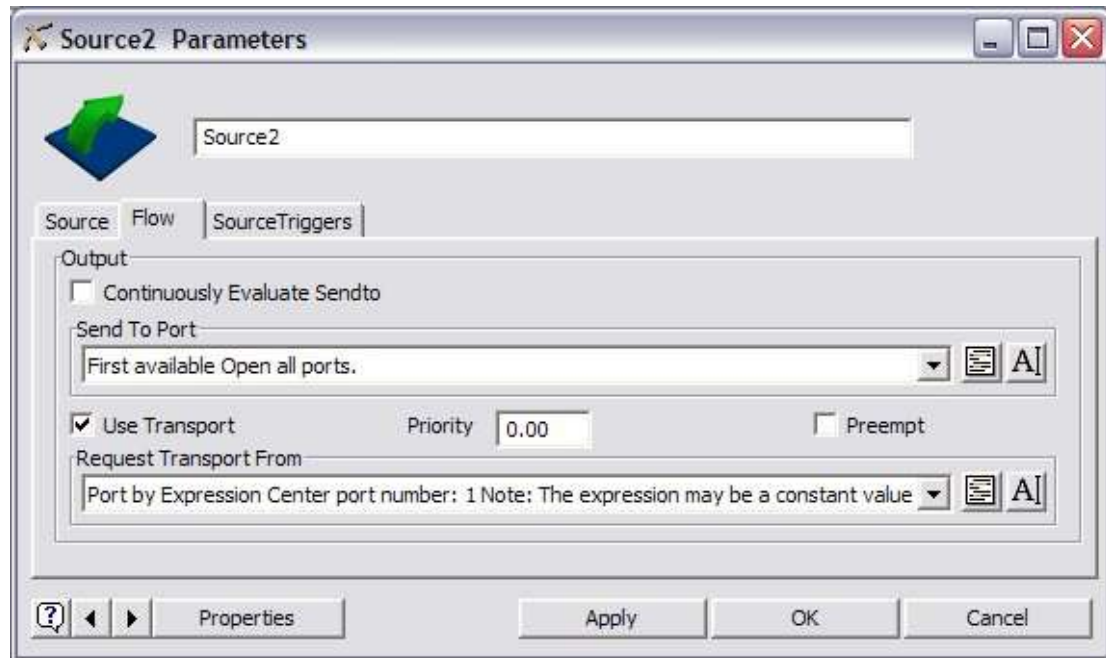


Figure 3: The Source calling an Operator for transportation

**Step 3: Set the colors of the objects**

When the objects are created they will be different colors, depending on their class. It is often helpful to color the objects based on the type of material that they will be processing. In this model there are two processing lines that are each composed of an ItemToFluid, a Pipe, a Tank and another Pipe. using the Properties GUI, change the color of those four objects on one line to blue and the objects on the other line to red.

**Step 4: Configure the ItemToFluid objects**

Next the ItemToFluids have to be configured to create the correct amount of material for each flowitem that comes in. In the Parameters GUI for each ItemToFluid, set the "Fluid Units per Discrete" field to 10. Make sure the "Discrete Units per Flowitem" field is still 1. This tells the ItemToFluid to create 10 gallons of fluid for each flowitem that enters.

The output rate also needs to be changed. Change the Maximum Object Rate and Maximum Port Rate to 2. If these two fields are different, the smaller one will be the limiting factor when material is leaving the object.

The ItemToFluids will receive flowitems as long as there is still space in it to hold the material that is created. We want to limit the amount of material these ItemToFluids can hold. Set the Maximum Content to 20 gallons.

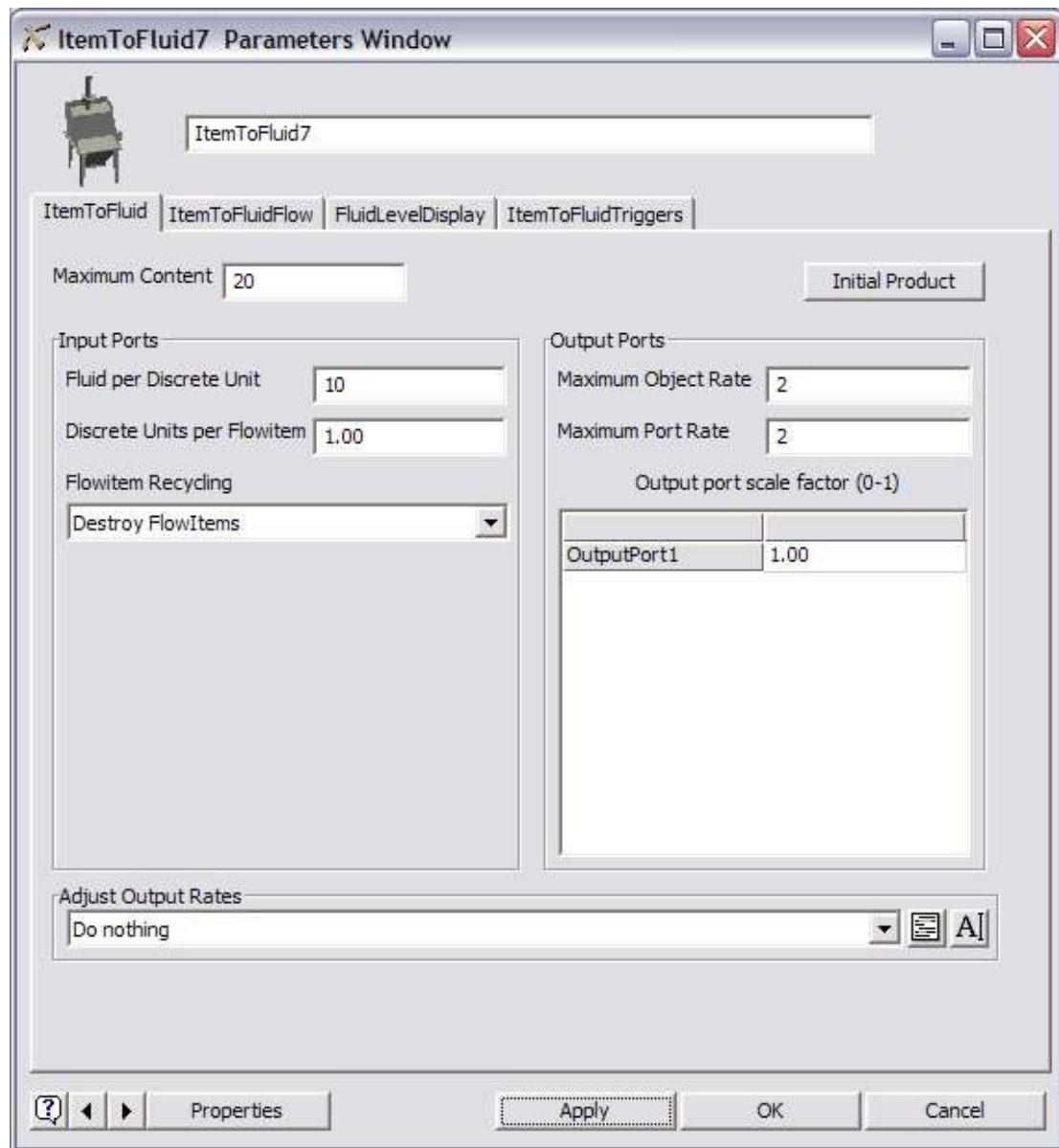


Figure 4: A fully-configured ItemToFluid

#### Step 5: Change the Pipes' rates and visuals.

The Pipes that lead away from the ItemToFluids are the next objects that need to be configured. Pipes do not allow the modeler to specify both the input and output rates. The output rate is based on the actual rate that material was received. For these Pipes, set the Maximum Flow Rate to 2 gallons per second. The Pipe has a field called "Flow Mode" that should be set to "Flow Evenly".



The amount of time that material stays in the Pipe is based on the maximum flow rate and the capacity of the Pipe. Set the Maximum Content of these two Pipes to 20. This will ensure that material spends time in the Pipe, but not too much time.

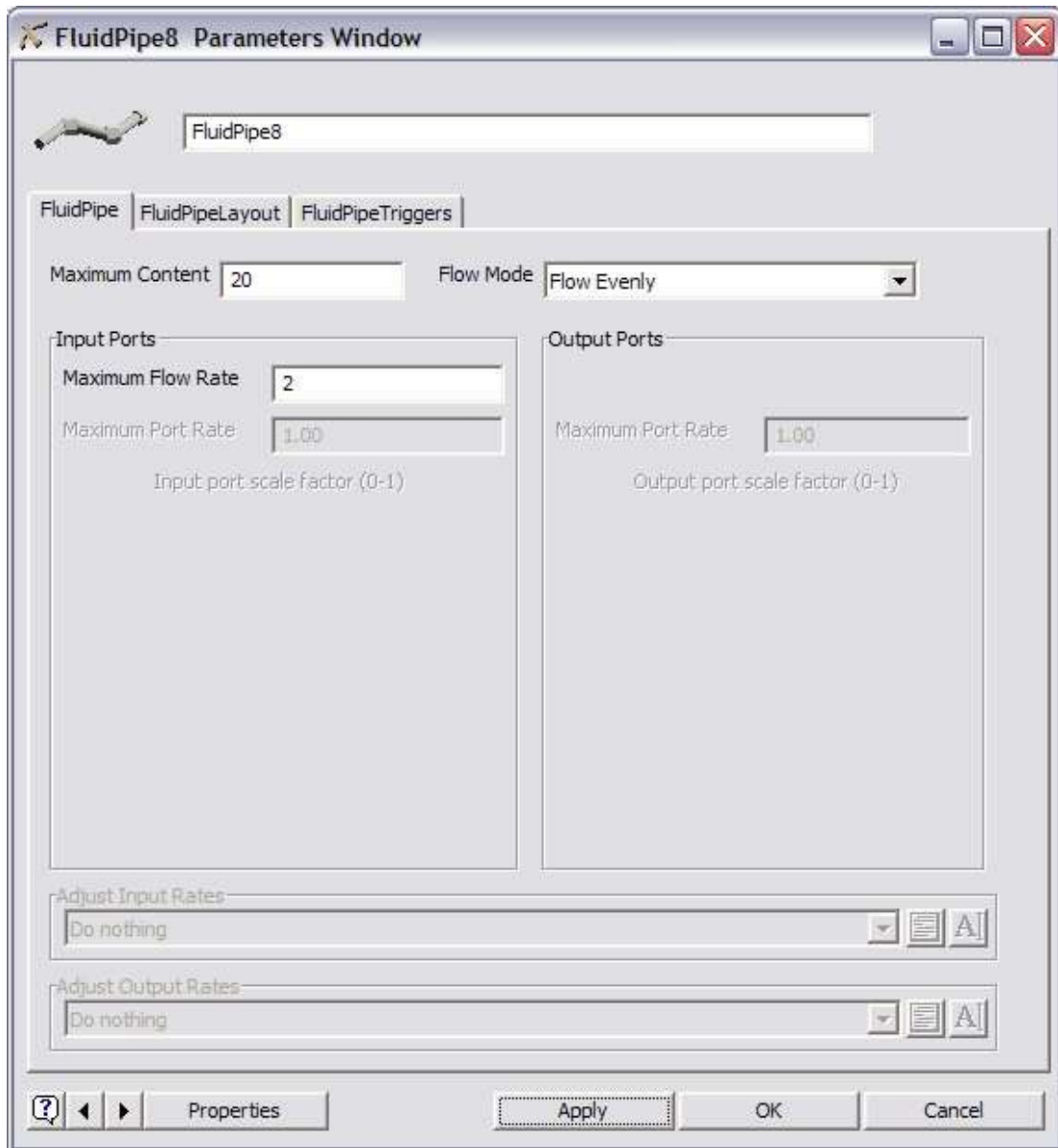


Figure 5: The content and rate values for the first Pipes.

Pipes can be configured to have multiple sections with bends between them. This does not affect the behavior of the Pipe in any way. Using the FluidPipeLayout tab in the Pipes' Parameters GUI, change the Pipes so that they appear to start near the ItemToFluid objects and end near the Tanks. The actual values in your table may be different from those in the following figure.

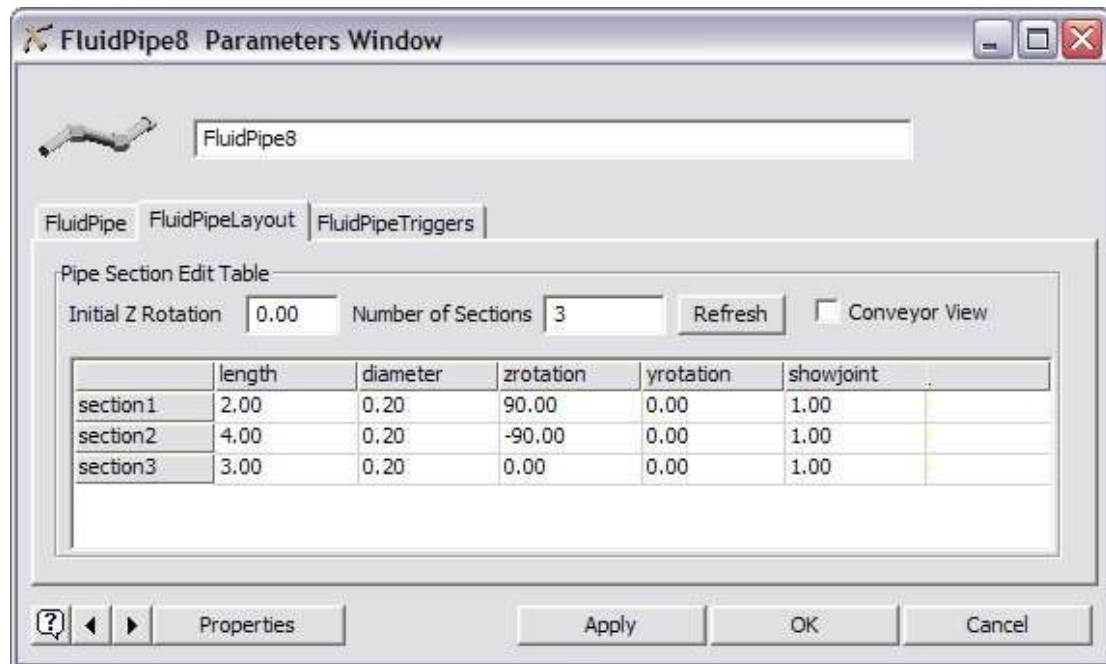


Figure 6: An example layout table for one of the first Pipes

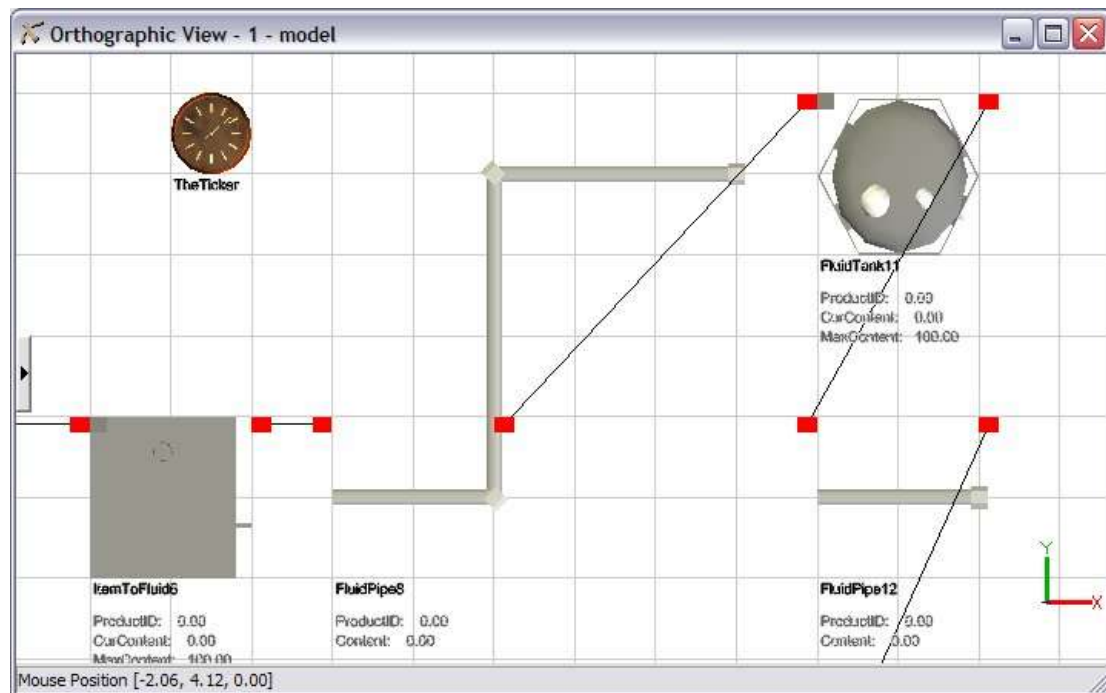


Figure 7: The results of the Layout table in Figure 6

**Step 6: Set the Tanks' rates, marks and flow logic.**

The Pipes now have a maximum output rate of 2, but the Tanks have a maximum input rate 1. If these values are left as they are, the Tank's rate will be used during the model run (because it is the lower of two values) and the Pipe will not be able to

send material downstream as fast as you have told it to. So the rate on the Tanks needs to be changed. Change the Maximum Object Input Rate and Maximum Port Input Rate to 2. From this point in the model to the end, leave the output rates of the objects at 1.

Tanks allow the modeler to set three levels that will cause triggers to fire when the content of the Tank reaches them. These values are called marks. They are edited on the FluidTankMarks tab of the Parameters GUI. For this model, the Tanks should keep their output ports closed until they have received a certain amount of material. They will then open the output ports and leave them open until the Tank becomes empty. They will always keep their input ports open. Set the low mark to be 1 and the high mark to be 45. If a mark has the value 0, the trigger for that mark will never fire.

There is a trigger that corresponds to each mark. They fire when the level in the Tank passes the mark, either by rising or falling. The triggers have a variable to tell the modeler whether the level is rising or falling. From the pick-list for the "Passing Low Mark Trigger" select "Open or Close Ports". This trigger needs to close the output ports whenever the level passes the low mark, both while rising and falling. In the code template set the Fluid Mode to "either" and the action to "closeoutput" (See figure 8). The trigger for the high mark needs to open the output ports if the level has risen to the mark. If the level is falling, it should do nothing. From the pick-list for the "Passing High Mark trigger" one again select "Open or Close Ports". This time the Fluid Mode should be "rising" and the action should be "openoutput" (see figure 9).

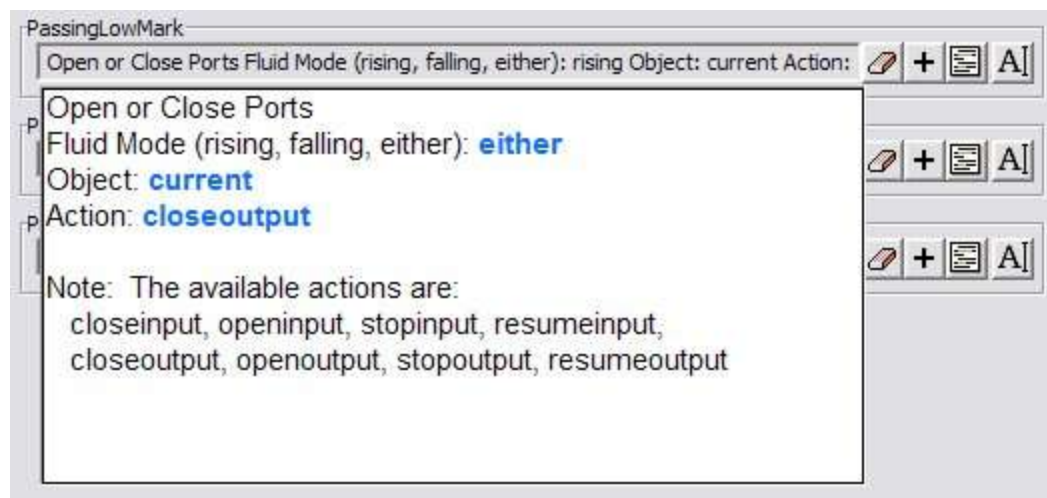


Figure 8: The Low Mark trigger for the Tanks

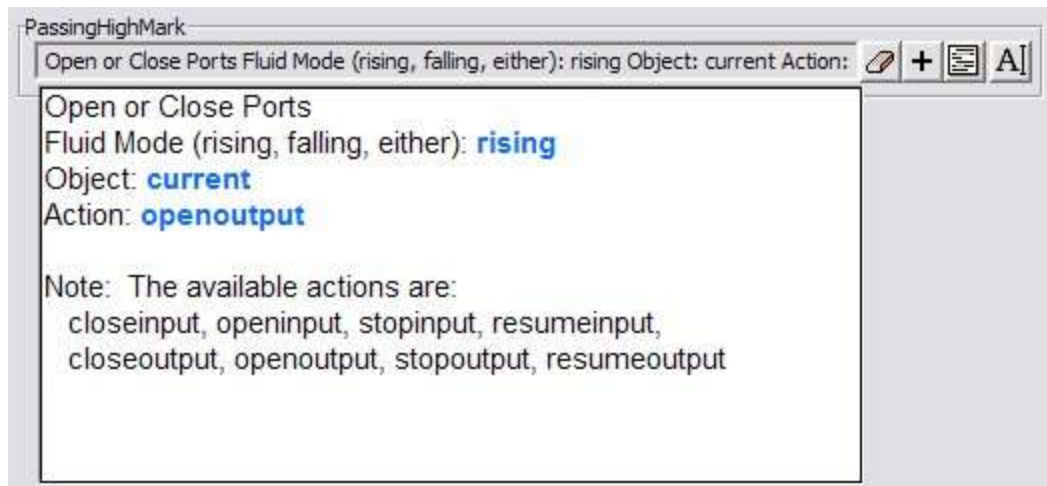


Figure 9: The High Mark trigger for the Tanks

**Step 7: Reorient and resize the next Pipes.**

Depending on where you placed the objects in your model, the Pipes that lead from the Tanks to the Mixer may need to be changed so that they point to the Mixer. Again, this is completely visual, the size and layout of the Pipes will not affect their behavior. Use the FluidPipeLayout tab to configure the Pipes in your model so that they look good to you.

The default maximum content of these Pipes is too large, however. Change it to 10 gallons. This will make sure that the material leaving the Tanks takes just a little time to get to the Mixer.

**Step 8: Set up the Mixer's step and recipe tables.**

The Mixer now needs to be configured to receive the two different materials and combine them into a new material. This is done by changing the two tables found on the FluidMixerSteps tab.

The Steps Table is used to define a series of steps that the Mixer must go through for each batch that it processes. In this model, the Step Table needs 2 steps. The description of the steps is not important, call them anything you want. The delay time for each step is executed after all the material for that step is received, and before the Mixer starts receiving material for the next step. The delay after the first step should be 0 and the delay after the second step should be 10.

The Recipe Table is used to define what materials the Mixer will receive and when it receives those materials. In this model, the Mixer will have two ingredients. Once more, the description can be anything you want because the object ignores it. The Mixer should pull 10 gallons of the first material from input port 1 during step 1. Then it should pull 20 gallons of the second material from input port 2.

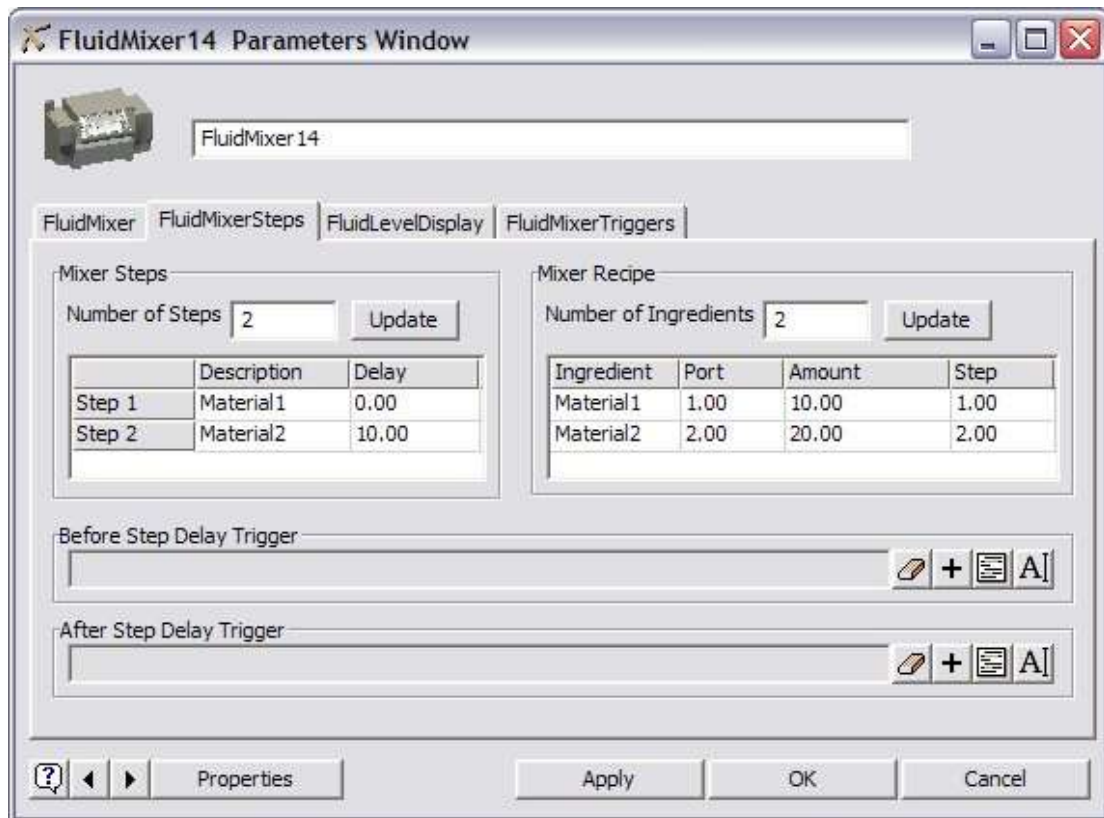


Figure 10: The Step and Recipe tables for the Mixer

The Mixer's level indicator bar is a useful tool to watch during the model run. It will display how much of each material in the Recipe Table the Mixer has received at any point in time. Unfortunately, by default it is hidden behind the Mixer's 3d shape. On the Mixer's FluidLevelDisplay tab, change the Y location of the level indicator bar to 1. This will make the bar appear in front of the Mixer so that you can see it.

#### Step 9: Examine the FluidProcessor.

The FluidProcessor's default values will work well for this model. It will receive material from port 1, process it for a certain amount of time and send it to port 1. The amount of time it spends processing is based on its maximum content and the Maximum Output Rate that the modeler defines. You can play with these values if you want to see how they affect the model.

#### Step 10: Configure the FluidToltem.

The FluidToltem near the end of the line will convert the fluid material coming from the FluidProcessor and change it into flowitems. Set the field called "Fluid per Discrete Unit" to 10 gallons. Make sure the field called "Discrete Units per Flowitem" is set to 1. These two fields will be multiplied together to determine how much material will have to be collected to create a single flowitem. In this case, 10 gallons of fluid will become 1 flowitem.

Also change the Maximum Content of the FluidToltem to 10. This tells the object that it can only collect enough material for 1 flowitem at any time. If this value is higher,

the FluidToItem may end up acting as a Queue and creating too much storage space in your model.

**Step 11: Reset and run the model.**

Once everything has been configured, press the Reset button, then press Run. This will start the model running. You should see the level indicator bars on the objects going up and down. You should also see the Pipe flashing. If a Pipe is drawn in gray, it is empty. If it is a pulsing color, material is flowitem. If it is a solid color, the material is blocked. There will also be flowitems becoming fluid and fluid becoming flowitems.

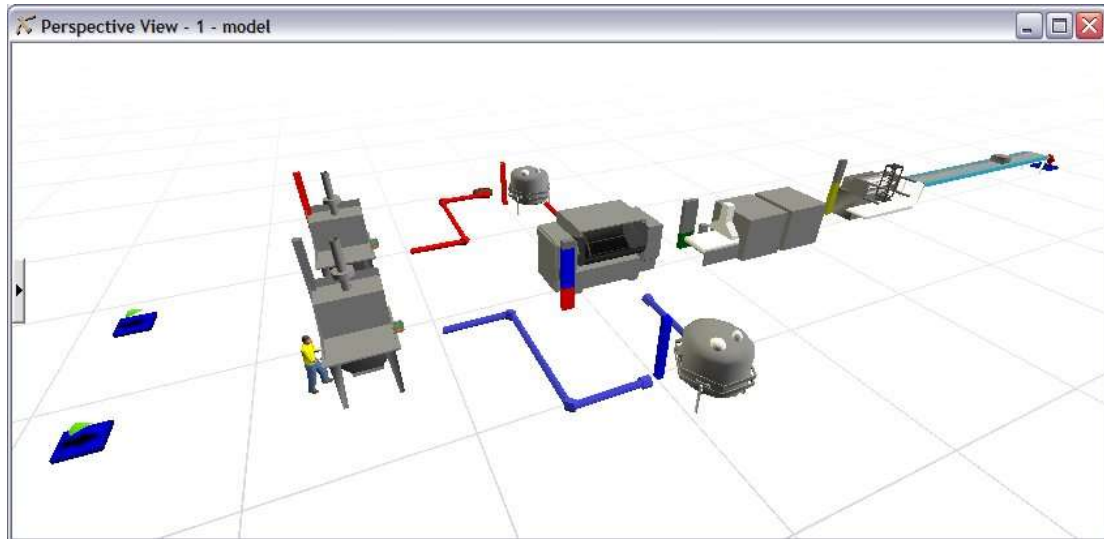


Figure 11: The finished model during a run

After completing this lesson you should have an idea of how these Fluid Objects work and some of their capabilities. There is far more that they can do that was not covered in this lesson. Read through their documentation and try other options and settings. Soon you'll be putting together larger, more detailed fluid-based models.

## Flexsim Concepts

This topic describes in detail concepts that are essential to understanding how to build models in Flexsim. You should have gone through the getting started and tutorial models provided with this user manual before reading this topic. In this topic we will not build any specific models. This allows us to focus exclusively on the concepts being discussed, instead of spending time on model building steps. We will, however, cite an example model where the concepts can be applied, and please feel free to build your own model as you go along in this topic. If you have gone through the tutorials, you should have the skills needed to build the model examples cited. If you do decide to build your model, however, I would advise that you read through this whole topic once, then go back and build the model as you go, because there are some things at the end of the topic that you'll want to understand before building the model.

In this topic I will often include snippets of code that help clarify the concept being discussed. The logic that the example code implements can be done in other ways by using the drop-down pick options, but I want to help you become more familiar with flexscript and will therefore use straight flexscript code examples. If you are still unfamiliar with flexscript, then you can skip those example snippets and move on, but I do try to give a concise description of what the code samples do, so you will hopefully be able to understand what's going on even if you are new to flexscript. If you have just finished doing the tutorials and have not learned yet how to directly edit a field's code, then please review the topic on pick lists first before reading this topic. For more information on writing flexscript code refer to the topic on writing logic in Flexsim.

### Itemtype

Itemtype is the first essential concept that we'll address. It is a value that is stored on every flowitem that travels through a Flexsim model. This value can be accessed and/or changed at any point in the flowitem's progress through the model. Every single flowitem has its own unique itemtype value, meaning that if you change the itemtype of one flowitem, it will only change the value for that specific flowitem, and the itemtype of other flowitems will not be changed. The meaning of the itemtype value is completely up to you the modeler. In general it is meant to be a value that describes a product type or category. Take for example a post office where customers come in to either have a package sent or to have copies made. In this model the flowitems, or customers, are separated into two general categories, namely those who need a package sent and those who need copies made. The itemtype value can be used to make this designation. For example, customers, or flowitems, who need a package sent can be given the itemtype value 1, and customers who need copies made can be given the itemtype value 2. In Flexsim, you will usually set the itemtype value in the Source object when the flowitem is first created. In our example let us say that 60% of arriving customers are "package" customers, and 40% are "copy" customers. To do this in Flexsim we would go to the Source's exit trigger code and add the command:

**Model building steps:** Here's how you build the model



```
setitemtype(item, bernoulli(60, 1, 2));
```

This command will randomly set the itemtype value of the flowitem that is exiting to 1 60% of the time and to 2 40% of the time. The setitemtype command sets the itemtype value of an object. It takes two parameters. The first parameter is a reference to the flowitem that you want to set the itemtype value on, and the second parameter is the value to set it to. In this example, the first parameter is item, or the flowitem that is currently exiting the Source, since we are in the Source's exit trigger (the reference "item" will be discussed in more detail later). Notice that for setitemtype's second parameter, the bernoulli command is used. This command takes 3 parameters and returns one of two possible values. The first parameter is a percentage value between 0 and 100. The second and third parameters are "success" and "failure" values, and represent the two possible values to be returned. In this case, 60% of the time the bernoulli command will return a 1 (parameter 2) and the rest of the time it will return a 2 (parameter 3). Since the bernoulli command is the second parameter of the setitemtype command, the itemtype value will be set to whatever is returned by the bernoulli command, namely the 1 or 2 value. The exit trigger of the Source is executed every time a flowitem is created and exits the Source. This means that the setitemtype command will be executed many times over the course of the simulation, and each execution will be associated with exactly one flowitem. Items exiting the Source will thus be split 60/40 for itemtypes 1 and 2 respectively. Since bernoulli is a stochastic, or random, command, the bernoulli will not always keep a perfect 60/40 ratio. You may have several consecutive customer arrivals whose itemtype value is set to 1, and vice versa. But over time the ratio will equalize out to 60/40.

Once you have initialized the itemtype value, logic in your model can then be implemented based on the itemtype value of each customer going through the model. In the example, a "package" customer may take 3 minutes to service, whereas a "copy" customer may take 5 minutes to service. In Flexsim, you would implement this difference by writing flexscript code in the process time field of a processor object. The code would look like this:

```
if(getitemtype(item)==1) return 3;  
else return 5;
```

This code basically says: if the itemtype value of the flowitem that is currently being serviced (getitemtype(item)) is equal to 1 (it is a "package" customer),



then return 3 as the process time. Otherwise (it is a "copy" customer) return 5 as the process time.

**Note on simulation time units:** Flexsim is inherently a "unitless" simulator. What this means is Flexsim time units are defined as just that: time units. They are not defined based on seconds, minutes, hours or any real life time unit. This gives you, the modeler, the flexibility to define what a Flexsim time unit means for a given model. In the example above, the code returns either 3 or 5 time units. By the fact that I said in the specification that a "package" customer takes 3 minutes and a "copy" customer takes 5 minutes, by returning 3 or 5 in the code field I have associated my model's time units with minutes. The key here is that you need to be consistent. Once you decide on which time unit to use, you need to define all times exclusively using that unit.

Again, this example can be done without writing any code, using Flexsim's pick-lists to define your logic. Nevertheless, the key concept to understand here is that every flowitem can have an itemtype value whose meaning is up to you, and that you can use the itemtype value to make decisions in your model.

**Note on Flexsim objects:** Every flowitem in a model has an itemtype. However, Flexsim objects like Sources, Queues and Processors do **not** have an itemtype.

**Note on the itemtype value:** The itemtype is a double precision floating point number. This means that the itemtype can not only hold integer values like, 1,2,3, etc., it can also hold floating point values like 1.5 or 99.9. However, the itemtype cannot hold string values like "package".

**Note on flowitem appearance:** The itemtype value will not define the visual appearance of the flowitem. This can be set by choosing the flowitem class in the Source's Parameters window, such as box, tote, or pallet.

## Labels

Labels are also a key concept to understand in building models in Flexsim. They are very similar to the itemtype value in that they store data on objects that can be used in making decisions in the model. However, there are some key differences, listed below:

- Each label has a name that is defined by you the modeler.
- Unlike itemtype, which is specific to flowitems, labels can be defined on objects as well as flowitems, for example on a Source, Queue or Processor.
- An object can have as many labels as you choose to give it.
- Labels can have number or string values, whereas the itemtype can only have a number value. Labels can even hold lists or tables of values.
- With labels, you must explicitly add the label to the object through its properties window, unlike the itemtype value, which is automatically included with every flowitem.

- When adding a label to a flowitem in the Flowitem Bin, the label is specific to that flowitem class. This means that if you add a label to the Pallet flowitem class, only flowitems that are created from that Pallet class will have that label on them.

To add labels to flowitems, go to the Flowitem Bin, select the flowitem class that is being created by your Source, and press the Properties button. Go to the labels tab and add string or number labels using the buttons at the bottom of the pane. To add labels to Flexsim objects, right click on the object in the ortho window and select the Properties option. Then go to the labels tab and add string or number labels using the buttons at the bottom of the pane. Specify each label's name in the row headers column on the left, and its value to the right of its name. For flowitems, the value you specify in the Properties window will be the default label value for all flowitems that are created, but you can change that value on each flowitem as it progresses through your model. For Flexsim objects' labels, the label's value will remain the same unless you have logic within the object that changes the label's value, in which case you will need to reset the label's value in the object's reset trigger. This lets you set the label back to an initial value when you press the Reset button to restart a model run.

Let's extend the example model mentioned above to use labels. Let's say for example that each "copy" customer that comes into the post office has a certain number of copies that need to be made, and that the service time for that customer is dependent on the number of copies needed. A customer that needs 1000 copies will take longer to service than a customer that needs 1 copy. As before, the itemtype value of each flowitem, or customer, reflects the category of customer, either "package" or "copy", but now for "copy" customers we need to add a label that tells us how many copies that customer needs. Again, to add a label to a flowitem, go to the Flowitem Bin, then select the flowitem class, go to Properties, and to the labels tab. Here we would add a number label and give it a name like "nrofcopies". As the default value we would leave it at 0 and set the value in the Source's exit trigger.

Once the label has been added in the Flowitem Bin, we can set the label's value when each flowitem exits the Source. In the example the copy customers will need a random number of copies between 1 and 1000. To implement this, you would modify the exit trigger of the Source as follows:

```
setitemtype(item, bernoulli(60,1,2));
if(getitemtype(item)==2) setlabelnum(item, "nrofcopies", duniform(1,1000));
```

As described previously, the setitemtype command sets the itemtype of the item to a 60/40 split between 1 and 2. Now we add an if statement. This if statement basically says: if the itemtype of the exiting flowitem is 2 (it is a copy customer), then set the value of the flowitem's label named "nrofcopies" to a random number between 1 and 1000. The setlabelnum command sets a label value and takes 3 parameters. The first parameter is the object whose label we want to set (item, or the flowitem that is exiting). The second parameter is the name of the label ("copies"). This parameter needs to be in quotes since it is a string parameter. The third parameter is the value to set the label to (duniform(1,1000)). The duniform command returns a value from a discrete uniform distribution. It takes 2 parameters, namely the minimum and maximum

value, and returns a random number between those two values, uniformly distributed, meaning every value between the min and max is just as likely to be returned as any other value between the min and max. The "discrete" part means that the command will only return 1,2,3, etc, as opposed to the uniform() command, which may return values like 1.5 or 2.5. Since there will never be a customer that needs 1.5 copies made, we use the duniform() command.

Note that by adding the "nrofcopies" label in the Flowitem Bin, every flowitem that is created from that flowitem class will have that "nrofcopies" label on it. Even package customers will have that label, but our logic will simply not look at the label if it is a package customer.

Now that we have set up our label and set its initial value, we can define logic to make decisions based on the value of that label in the model. For a copy customer, for example, we can change the service time based on the number of copies that the customer needs. For each copy customer, the service time can be a base of 5 minutes as before, plus an additional 5 seconds for each copy that needs to be made. To make this change, you would again go to the Processor's Process time field and change it to the following:

```
if(getitemtype(item)==1) return 3;
else return 5 + (getlabelnum(item, "nrofcopies")*(5.0/60.0));
```

As before we use an if statement to give itemtype 1 (package customers) a service time of 3 minutes. In the else portion (copy customers), though, we return the expression: 5 + (getlabelnum(item, "nrofcopies")\*(5.0/60.0)). This is the base service time of 5 minutes plus the number of copies that the customer needs (getlabelnum(item, "nrofcopies")) times five seconds (5.0/60.0). Remember that we have defined our model in minutes, so if one Flexsim time unit is equal to one minute, then five seconds is equal to 5/60 minutes or time units.

**Note on the division operator:** In the above example I use the expression 5.0/60.0 instead of 5/60. It is important to make this distinction because C++ sees the two division expressions differently. With 5/60, C++ sees this as the integer 5 divided by the integer 60. Thus, an integer divided by an integer must also be an integer, or 0. With 5.0/60.0, however, C++ sees it as the division of two floating point numbers, and thus the result is a fraction between 0 and 1. On the other hand, flexscript, which is not strongly typed like C++, actually interprets the expression 5/60 as the division of 2 floating point numbers, meaning you would be fine using 5/60 in flexscript. However, in the few situations such as this where flexscript's implementation deviates from the C++ implementation, we encourage you to write code that is cross-compatible with flexscript and C++, and thus the correct expression would be 5.0/60.0. For more information on integer vs. floating point division, refer to the topic on writing logic in Flexsim.

So again, just as with the itemtype value, we can use labels to store data on flowitems (or objects), and then we can access that data to make decisions in the model.

## Item and Current

The terms `item` and `current` are two access variables that refer to objects in Flexsim. When you edit the code of a given field you will always see at the top of the code one or more "header" statements. These statements set up your access variables for you, and usually will look something like the following:

```
treenode current = ownerobject(c);
treenode item = parnode(1);
```

In this example, the first statement is what we call a variable declaration. It declares a variable called `current`. The variable type of `current` is a `treenode`. This is a variable type that holds a reference to an object in Flexsim's tree structure. I don't want to go into too much detail on this, so in a nutshell, all data in Flexsim, including objects and flowitems, is stored as nodes in a tree structure, and the `treenode` variable type is simply a reference to a node (or object) in that tree structure. For more information on the tree structure, refer to the topic on Flexsim's tree structure. The first statement's declaration also sets the value of this variable named `current` to: `ownerobject(c)`. Now, I also don't want to go into too much detail on what the meaning of `ownerobject(c)` is because that can be a complicated side-track. The essential thing here is that you have a `treenode` variable (or object reference) called `current`, and you'll just have to trust me when I tell you that `current` will always point to the "current" object that you are editing the field for. If you go into a Source's parameters window and edit the Source's exit trigger, then in that field, `current` is a reference to that Source object. If, on the other hand, you go into a Processor object's parameters window, and open the code for the Processor's process time field, then within that field, `current` is a reference to that Processor object.

The example code also has a second statement. The statement is another declaration of a `treenode` variable, called `item` this time, that is given the value: `parnode(1)`. Again, in order not to get side-tracked, I'm not going to explain the `parnode` command, but will just say that `item` will always refer to the flowitem that is associated with a specific execution of that field or trigger. For example, if you are implementing the exit trigger of a Source, then each time the exit trigger is fired, `item` will refer to the flowitem that is exiting the Source at that specific time. Note that the `item` reference will change each time the exit trigger is executed because a new flowitem is exiting, whereas the `current` reference will be the same each time because the Source object does not change.

So these header statements set up the access variables that can be used within the code of the field. This is why in the previous examples I was able to use the word `item` in writing the commands:

```
setitemtype(item, bernoulli(60,1,2));
```

or:

```
if(getitemtype(item)==2)...
```

because I have a reference to the flowitem and the reference is named item.

Often the header statements mentioned above will vary depending on the type of field you are writing code for. For example, the header statements of an exit trigger should look like this:

```
treenode current = ownerobject(c);
treenode item = parnode(1);
int port = parval(2);
```

Here there is an additional variable declaration of an integer called port. In this case port is the output port number through which item is exiting. A reset trigger's header statements, on the other hand, will look like this:

```
treenode current = ownerobject(c);
```

Here there is only one variable declaration, namely current. There is no item declaration. The reason for this is because the reset trigger, which is executed when you press the model reset button, has no specific flowitem associated with its execution.

This user manual documents each field and its access variables in the topics and sub-topics of the pick list section.

So to review, within Flexsim's code fields you will often have access to variables called current and item. current will always reference the object whose code you are editing. item will always reference a flowitem that is associated with a specific execution of the field, for example, the item that is exiting the Source. The access variables will vary based on the type of field, but you can always find which variables are available just by looking at the header statements at the top of your code, or by referring to the user manual.

**Note on specifying the correct object when accessing labels or itemtype:** It is important to understand which object is holding a label or itemtype. For example, in the example above we use the command `getlabelnum(item, "nrofcopies")`. We do not use `getlabelnum(current, "nrofcopies")`. The reason we use item and not current is because the label is stored on the flowitem itself, and not on the Flexsim object. If you add a label to a flowitem in the flowitem bin, then item should be the reference for the `getlabelnum` command. On the other hand, if you are using a label on the object (you have added the label to the object through its properties window), then current should be the reference in the `getlabelnum` command.

## Return Values

In Flexsim there is a close interaction between the "under-the-hood" behavior of the objects in your model and the logic that you implement on those objects through code fields. Often the very reason that a code field is executed is because the Flexsim object (Processor, Source, etc.) is requesting data from you the modeler as to how it should operate. For example, the process time field of a Processor is executed because the Processor needs to know from you what its process time should be for a given flowitem. The way that you pass the correct information back to

the Processor is through the return value of that field, or in other words, by executing a return statement in your code. Let's refer back to the post office example mentioned previously. In that example we implemented the process time code as:

```
if(getitemtype(item)==1) return 3;
else return 5 + (getlabelnum(item, "nrofcopies")*(5.0/60.0));
```

The Processor will execute this field each time it receives a flowitem, just before it starts the process time. By executing this process time field, it is essentially asking you what the process time for that item should be. In this code we are using the return statement to pass back to the Processor the appropriate process time for that flowitem. Thus, the return statement is used to give back to an object the appropriate data that it needs to operate as we want it to.

Many fields do not need a returned value. For example, in the post office model the Source's exit trigger does not include a return statement. The reason for this is because with an exit trigger, the Source object is not trying to get information back from you, it is simply providing you with a spot where you can execute functionality when a flowitem exits the Source.


This user manual documents each field's required return value in the sub-topics of the pick lists section.

## Template Code

In the code edit fields you may also find seemingly weird gray text strewn throughout the code. For example, you might find the following piece of code:

```
/**By Expression*/
/** \nExpression: */
double value = /**/10/**/;
/** \n\nNote: The expression may be a constant value or the result of a command
(getitemtype(), getlabelnum(), etc).*/

return value;
```

The gray text is what is called template code. It allows you to specify what will be shown as fixed and/or editable text when the user presses the  button. It makes use of flexscript/c++'s comment syntax, so first let's give some background on commenting.

In flexscript, you can "comment out" a section of text so that the flexscript parser does not look at that text as part of the code. This allows you to add descriptive text that explains what the code does. There are two ways to make comments. The first is the one-line comment, and is done with two forward slashes: //. The example below shows a one-line comment


```
// This is my one-line comment, it ends at the end of this line
```

A one-line comment extends to the end of that line of text. The other comment is a multi-line comment. Here you signal the start of the comment with the text: `/*` and signal the end of the multi-line command with the text: `*/` as shown below:

```
/*
this is my multi-line comment
it can span as many lines
as I want it to
*/
```

Flexsim's template code mechanism uses commenting to specify template text. Remember that template text (the text that you can edit in the template drop-down) is split into fixed black text and editable blue text. In your code, to specify a section of fixed black text, use a multi-line comment but add an additional asterisk to the start tag: `/**`. By adding this extra asterisk, it signals to Flexsim's template code interpreter that this is a section of fixed black text that should show up when the user looks at the template text. In the example above, the text: `/**By Expression*/` makes it so the text: By Expression will show up in black text in the template drop-down. Now to specify the blue editable text. With blue editable text, you want the input from the user to be part of the actual code. So to start blue text, you go into a multi-line comment then immediately go out of the comment: `/**/`. You use the same tag to get out of a section of blue text. Thus, in the code above, the value 10 is made available for changing when the template text is shown:

```
double value = /**/10/**/;
```

Because these are comments, the flexscript parser only sees: `double value = 10;` but the advantage is that now you (or another modeler) can quickly change the 10 value to something else just by pressing the  button and editing the blue template text.

You may also notice that the comments will occasionally include a `\n` tag. This tag specifies a new line to be made in the fixed black template text. You can also specify a new line just by putting a new line in the comment, but often you will want your template code to take up as little space as possible so that the code itself can be viewed more easily.

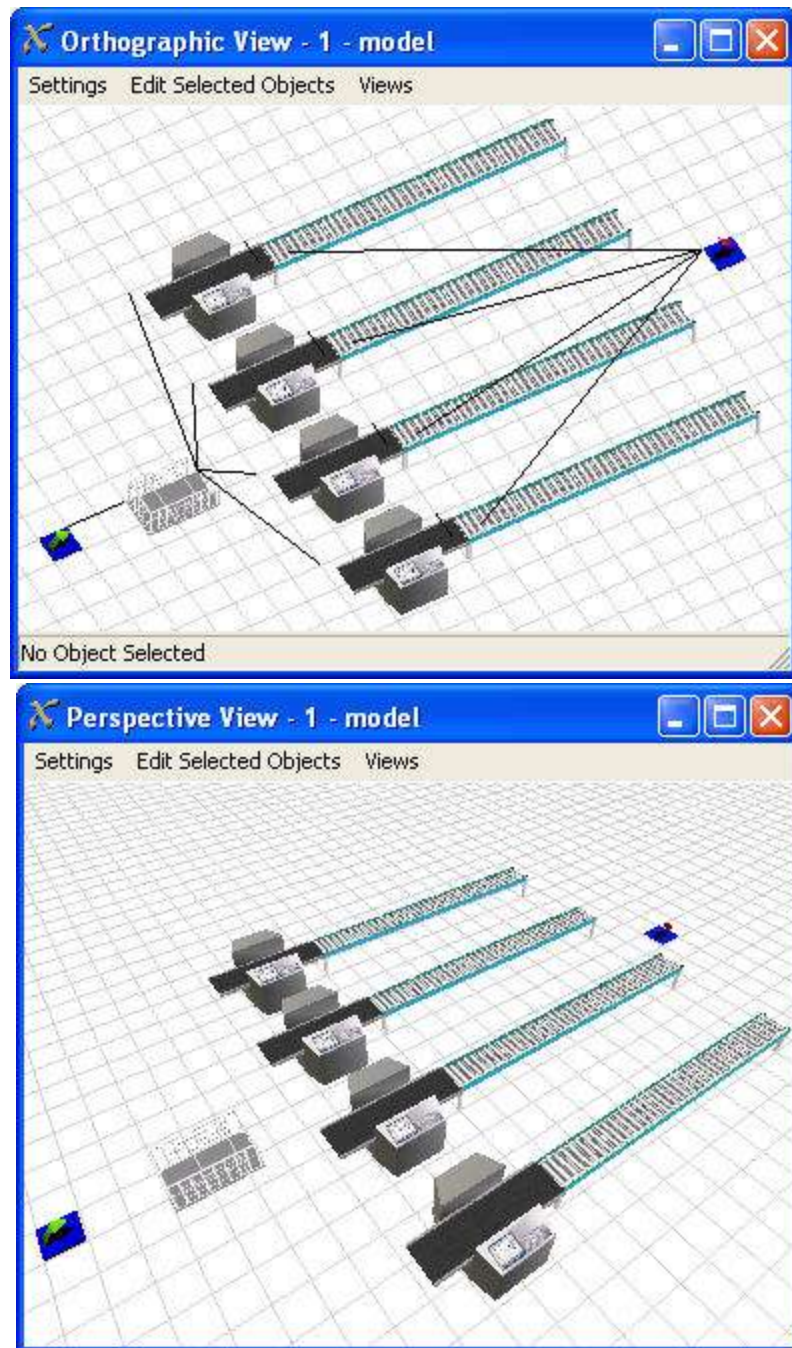




# Modeling Views

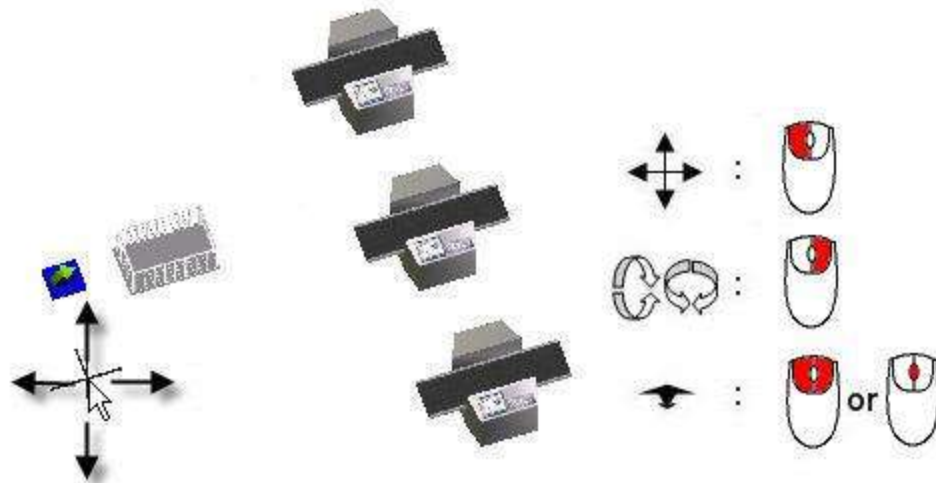
## Orthographic/Perspective View Window

The orthographic and perspective view windows allow you to view and edit your model in a 3d environment. The orthographic view uses orthographic parallel projection, and is used in the design and building phase. The perspective window uses perspective projection, giving the model a more 3D world feel. This is used once you've built the model and want to show it off.



### Moving Around in the View

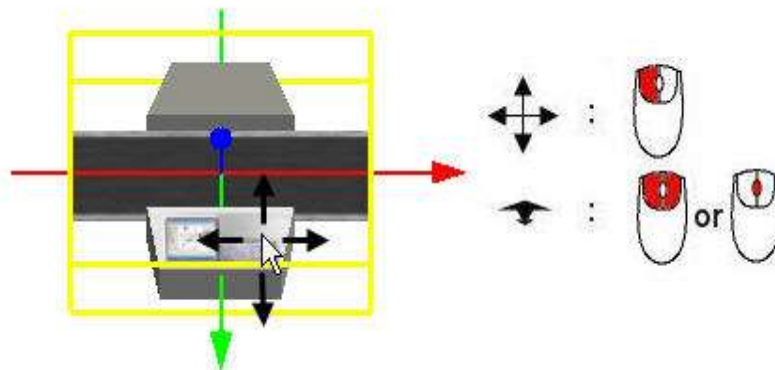
To move around in the view, click-and-hold on the floor of the model, and drag the mouse around in the view. This will translate the camera. To rotate the view, right click-and-hold on the floor of the model and drag the mouse in different directions. To zoom in and out, hold down both the right and left mouse buttons and move the mouse up and down. You can also zoom in and out by scrolling the mouse wheel.



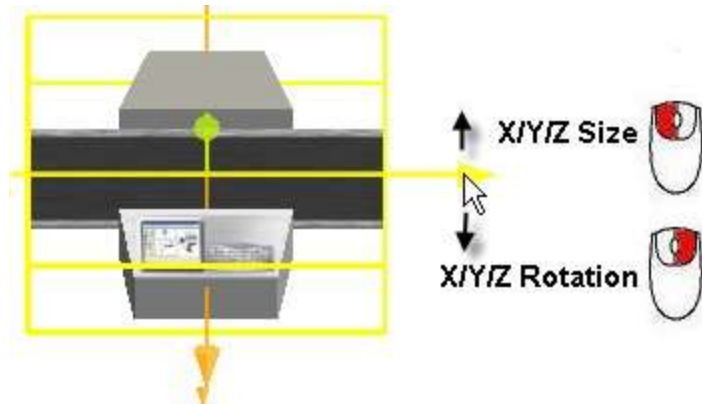
If you are working in a perspective view, you can also do a mouse guided fly through using the F7 key. Make sure the mouse is in the center of the window. Then press the F7 key. Move the mouse up and down to fly forward and backward. Move the mouse left and right to turn left and right. Once you are finished, click on the F7 button again to exit fly-through mode. It is often easier to navigate if the view is configured as first person (from the settings window).

### Moving Objects

To move an object in the X/Y plane, click-and-hold on the object and drag it to the desired location. To move the object in the z direction, click on it and scroll the mouse wheel. You can also hold both the left and right mouse buttons down on the object and drag it up and down.

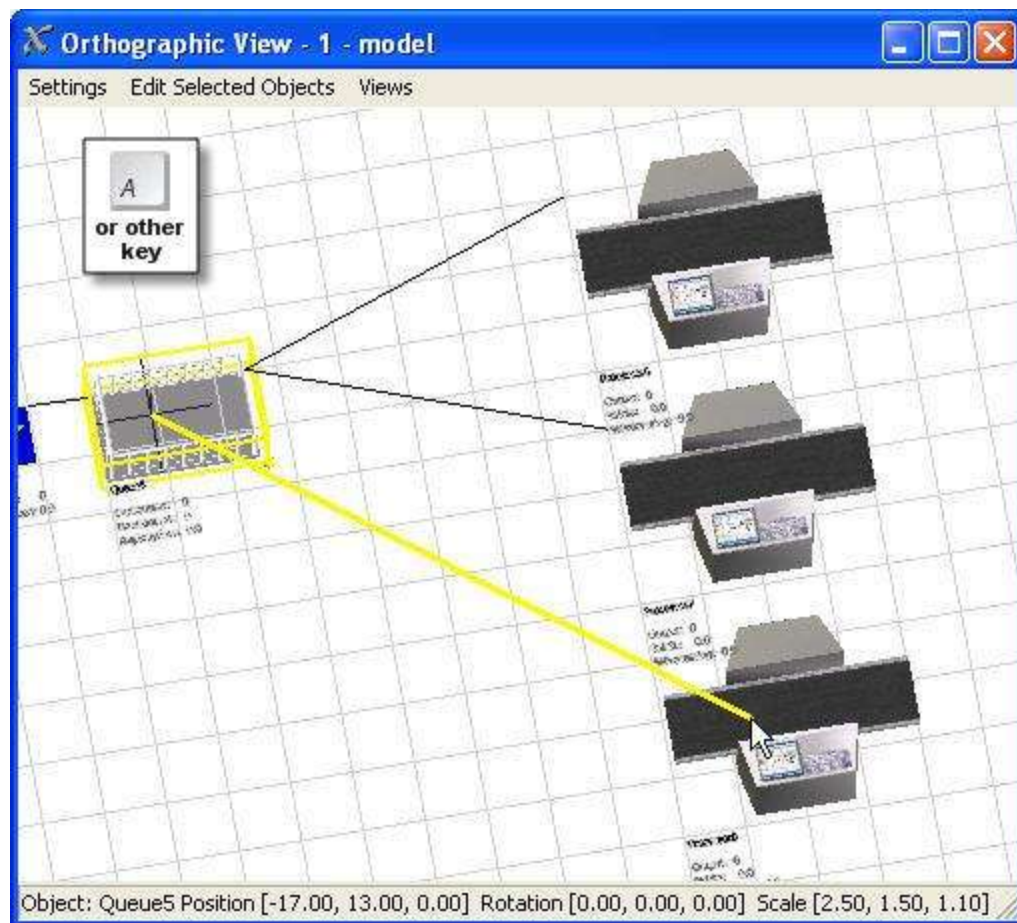


To rotate the object, select Edit Object Size/Rotation from the popup menu, then click on one of the three axis arrows with the right mouse button and drag the mouse up and down. To change the object's size, click on one of the three axis arrows with the left mouse button and drag the mouse up and down.



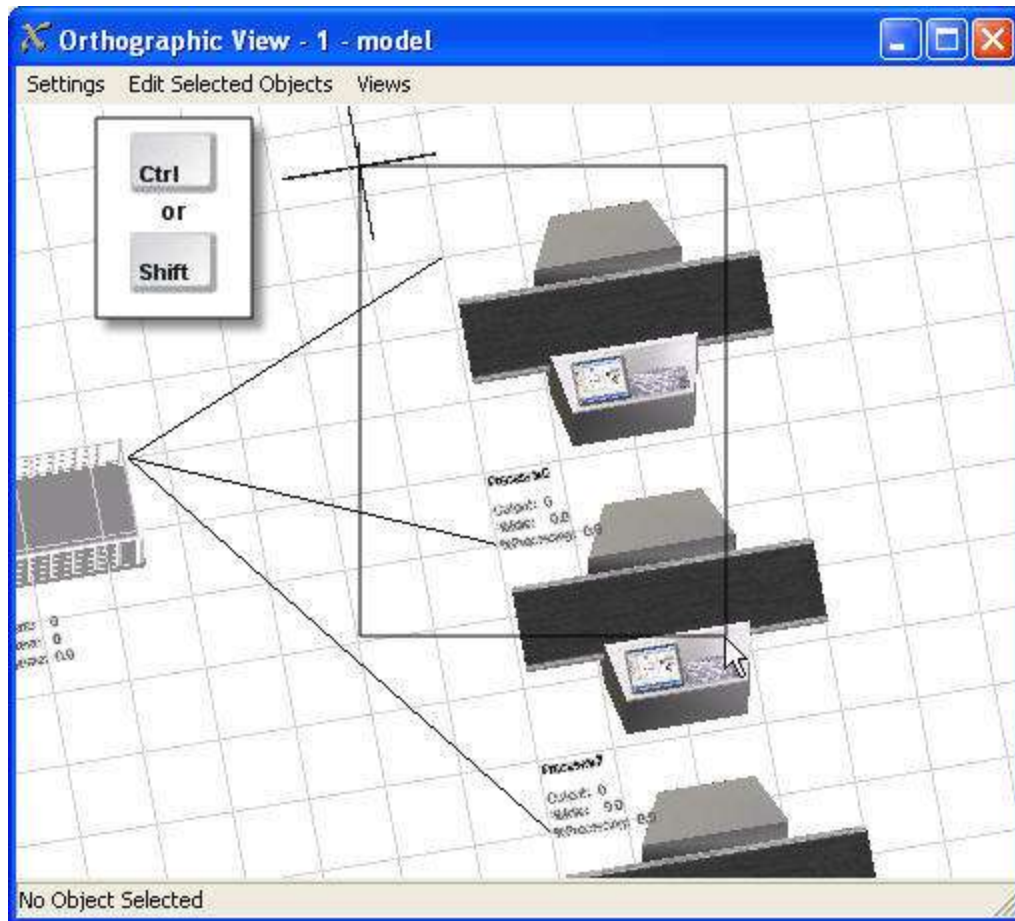
## Connecting Objects

To connect two objects in the model, hold the 'A' key down, click-and-hold on one object, drag the mouse to the other object, and release the mouse button on that object. The 'A' connect method usually connects output ports to input ports, but you can also use other key connections. These are described in detail in the keyboard interaction section.



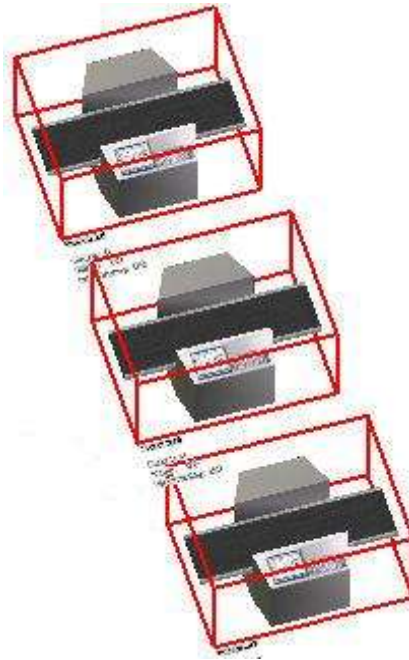
### Creating and Editing Selection Sets

You can create selection sets to have operations apply to a whole set of objects. To add object to the set, hold the Shift or Ctrl key down and drag a box around the objects that you want selected. Holding Shift down resets the selection set, while holding Ctrl down adds the objects to the selection set. You can also hold Shift or Ctrl down and click on objects, instead of dragging a box around them.





Objects in the selection set are drawn with a red wire frame around them.





Once you have created a selection set, moving, rotating and scaling one of the objects will cause the other objects in the selection set to be moved, rotated or scaled as well. You can also perform several operations on the selection set from the Edit Selected Objects menu.

## Toolbar

The orthographic/perspective window has a toolbar on the left of the window that can be shown or hidden with the  and  buttons.

- The View Settings toolbar lets you configure parameters for the view.



Click on the checkboxes to show/hide different items in the view. Press the + or - boxes to increase or decrease sizes in the view. Click on More Settings... to open the view settings window for more detail view settings.

- The Edit Selected Objects menu lets you perform many operations on the selected set of objects.
- The Views menu allows you to store preset view positions so that you can quickly go back to them when you need to.

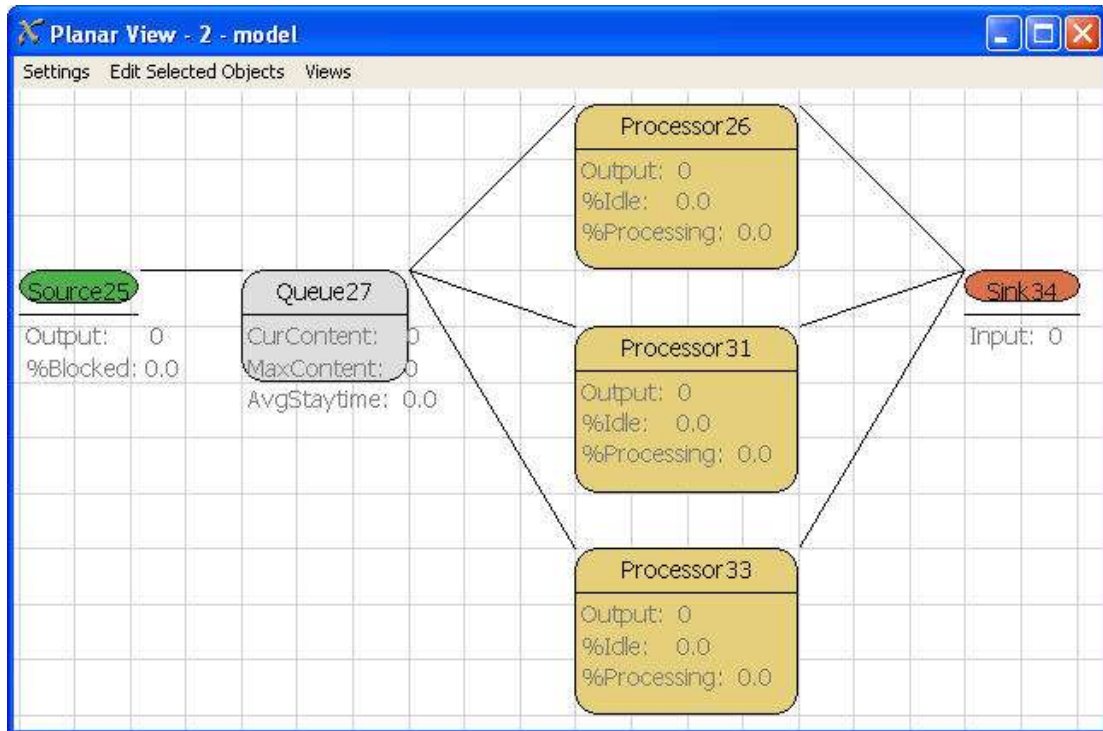
- The view also has a pop-up menu that appears when you right-click on an object in the view.

### **Editing an Object's Parameters and Properties**

To edit the parameters of an object, double click on the object or right click on it and select "Parameters" from the pop-up menu. To edit the properties of the object, right-click on it and select the "Properties" option from the pop-up menu.

## Planar View

The planar view is used to edit your model in a two dimensional view. It operates much like the orthographic and perspective views, except that you can't rotate the viewpoint. Also, instead of objects having their 3d shapes drawn, their 2D shapes are drawn.

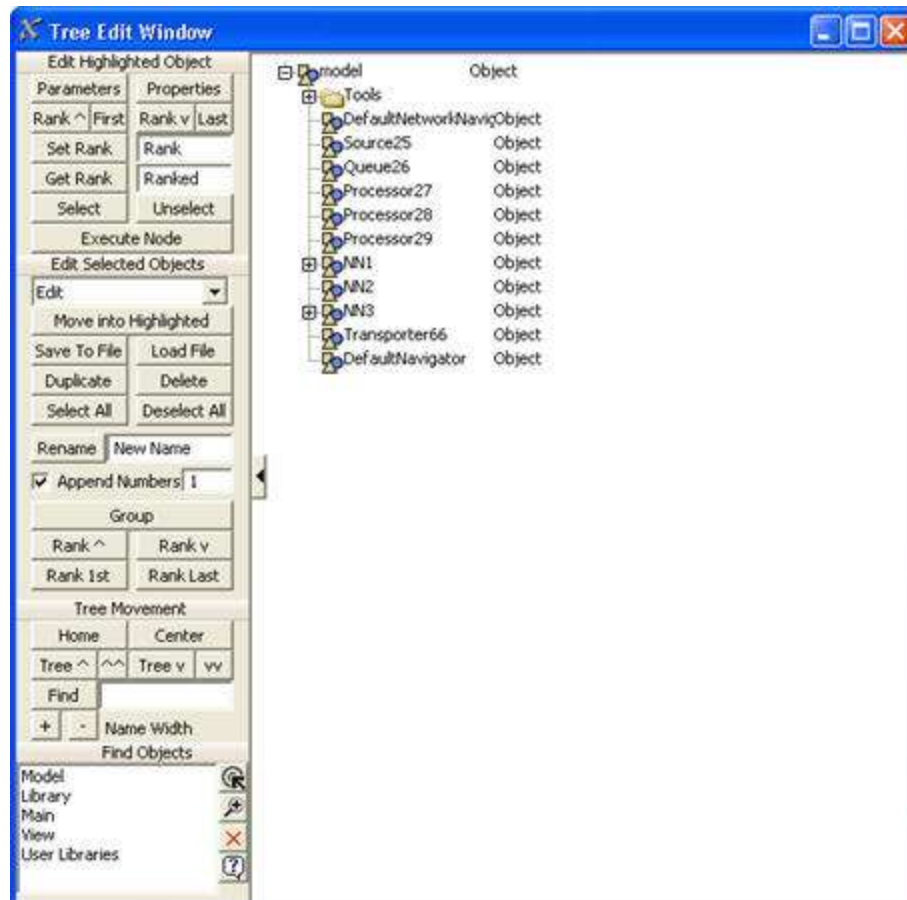




## Tree Window




The tree window lets you navigate around in the Flexsim tree structure, view and/or setting attributes on objects, writing code, and doing many other operations. To move around in the window, click the mouse on a blank area of the tree view and drag it around. You can also use the mouse wheel and page up/page down keys to scroll up and down in the tree window. For more information, refer to the Flexsim Tree Structure.

On the right is a normal tree window. On the left are several editing tools for organizing the model tree.



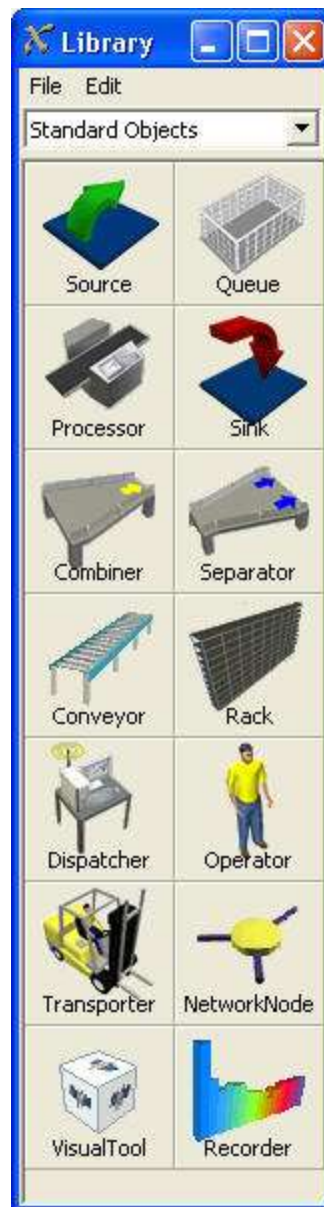
The tree window's tool panel provides several toolbars, including an Edit Highlighted Object toolbar, an Edit Selected Objects toolbar, and a Find Objects toolbar.

**Tree Movement** - This panel allows you to move the tree's viewpoint around. Press Home to go back to the top of the tree. Press Center to horizontally center the view. Press Tree ^ or Tree v to page up or page down. Press ^^ or vv to double page up or double page down. Type a name or substring of a name in the edit and press Find to find an object in the tree. Press + or - to increase or decrease the width of names in the tree view.

**Views List** - The views list at the bottom lets you quickly move between different parts of the tree. By default you can view the model, library, main, view, and user libraries trees. You can also add options to this list. The  button adds the tree's highlighted object to the list of views, and makes that object the new root of the tree view. The  adds that the tree's highlighted object to the list of views and repositions the tree to be looking at that object whenever the option is selected. The root of the tree remains the same, but the tree "zooms in" on that object. The  button removes an option from the list.

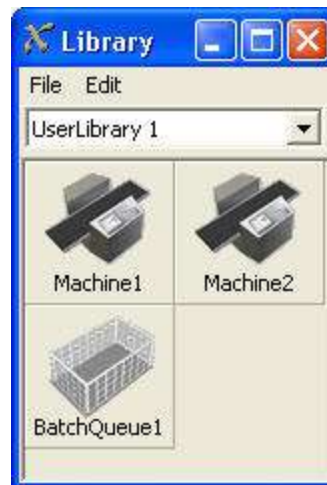
## Library Icon Grid

The library icon grid lets you drag objects into your model from Flexsim's standard library set, or from custom-made libraries. Click and hold on the object you would like to add to your model, drag it over an ortho, perspective, or planar view, and release the mouse button at the location you would like to drop the object into the model. At the top of the window you can select objects from one of three different categories: Standard Objects, which are the most used objects in the library, Fixed Resources, and Mobile Resources. If you have loaded additional libraries, these libraries will also be displayed as options in the drop-down list.

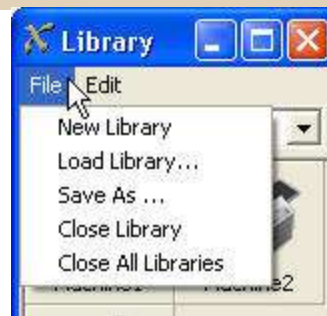


## Custom User Libraries

Flexsim lets you create and configure special libraries in addition to the standard library set. These are often referred to as user libraries. You can create custom defined functionality on objects, and then add those objects to a library for use in other parts of your model or in other models. You can save these libraries and then load them into other projects later on. You can also define a set of objects in the library to be automatically installed to your model when a new model is created or when you load the library.



### File Menu



**New Library** - This option creates a new library and adds it to the list of currently open libraries.

**Load Library** - This option loads a saved library, adding it to the list of currently open libraries. If the library contains components for automatic install, a message will appear asking you if you want to install these components. Press OK to install these components. This is explained in further detail below.

**Save As...** - This option saves the currently selected library to file.

**Close Library** - This options removes the currently selected library from the list of libraries.

**Close All Libraries** - This options removes all custom user libraries.

## Edit Menu

The edit menu lets you change currently loaded libraries.

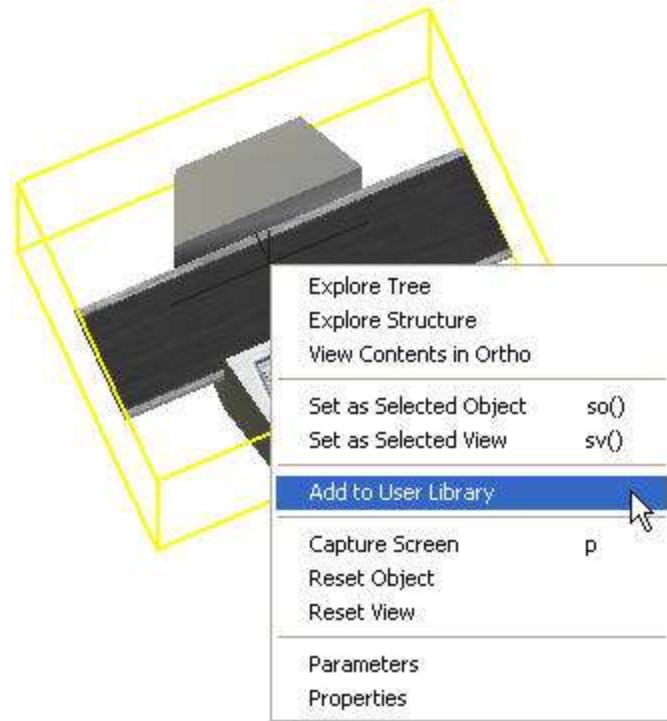


**Rename Library** - This option lets you rename the currently selected library.

**View/Edit Library Install Components** - This option opens a tree view, allowing you to view and edit the folder containing objects that are automatically installed when the library is loaded and when a new model is created. When adding an object to a library, some types of objects will give you the option of adding it to the library either as a drag-able icon or as a component for automatic install. If you choose to add it as a component for automatic install, then instead of showing up in the library's icon grid, it will be put in a special folder in the library. When the library is loaded, and when you create a new model, Flexsim checks that folder and automatically drops the objects in that folder into the model. Objects that you might want automatically installed to the model are things like GUIs, Global Tables and flowitems.

## Adding objects to a custom library

There are several types of objects that you can add to a user library. You will most commonly add a standard object like a Processor or Queue whose parameters you have modified to fit a custom modeling situation. You could also start from scratch with a BasicFR or BasicTE object, implement your custom behavior, and then add it to a library. Another possibility is to use a VisualTool as a container of a sub-model, and add the whole sub-model to the library. To add these types of objects to a user library, right click on the object in an orthographic or perspective view and select the Add to User Library option.

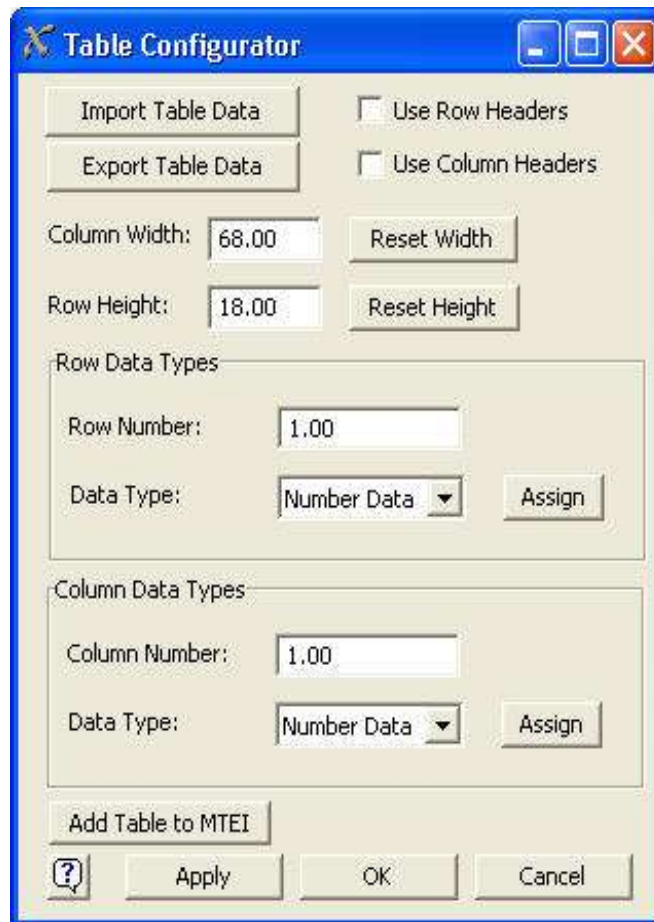


This will add the object to the currently selected library. If there are no custom libraries loaded, then a new library will be created, and the object will be added to that new library.

You can also add GUIs, Global Tables, flowitems, and user commands to a custom library. You can access this capability from the windows in which you edit these respective objects. The figure below shows a menu option in the GUI editor to add the GUI to a library.

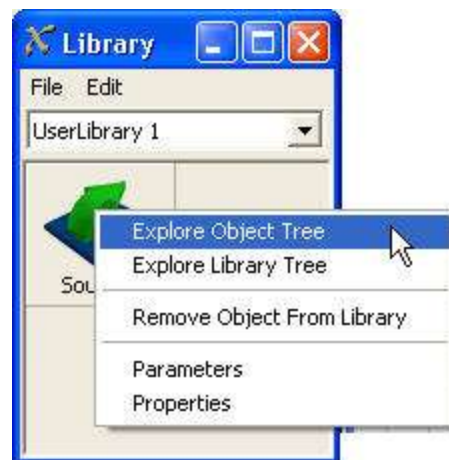


The figure below shows the advanced page of a Global Table. The two buttons at the bottom allow you to add the table to a user library.



**Note on adding an object to a library:** When an object is added to a library, a copy of the object is created and put in the library. This means that once it is added in the library it has no more linkage to the original object. It is a copy and can be changed on its own, separate from the original object.

Once you have added objects to the library, you can edit or remove these objects by right-clicking on them in the library window. The following menu will appear.



**Explore Object Tree** - This option lets you explore the object in a tree window.

**Explore Library Tree** - This option lets you explore the whole user library in a tree window.

**Remove Object From Library** - This option removes the object from the user library.

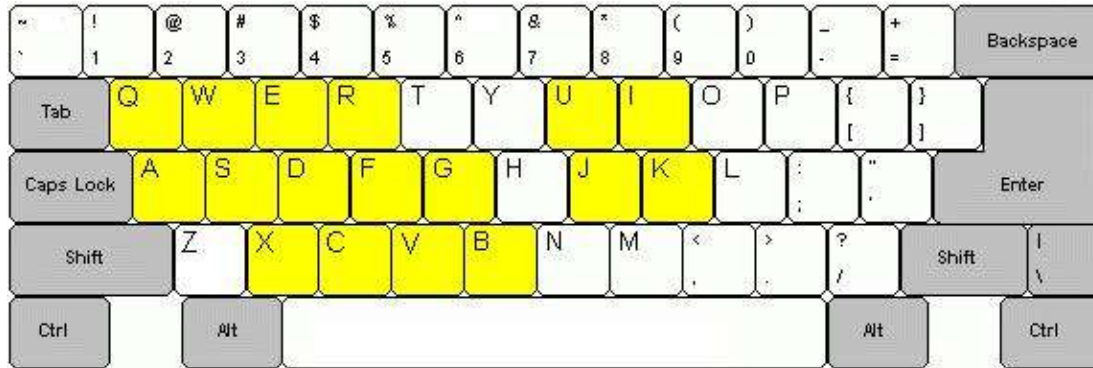
**Parameters** - This option opens the parameters window of the object.

**Properties** - This option opens the properties window of the object.



## Keyboard Interaction

When you are working in an orthographic or perspective view, you will use several keys on the keyboard to build, customize, and get information from the model. The figure below shows the keyboard layout. Keys that are highlighted in yellow have meaning when interacting with Flexsim.



**Note on key presses:** The ortho or perspective view must be the active window in order for key presses to work properly. You should first activate the window by clicking on the title bar before doing any key presses. Otherwise, the key presses will not work until the second time you try them.

### A, J: context sensitive connect

The A key is used to connect two objects depending on the type of objects. Hold down the A key, click on one object, holding the mouse button down, and drag to another object, releasing the mouse button on that object. Usually this connects the output ports of one object to the input ports of another object. For NetworkNodes, however, the A key connects a NetworkNode to TaskExecuters as travelers, to FixedResources as travel gateways, and to other NetworkNodes as travel paths. You can also use the J key if you are left handed. If you connect two objects with the A key, and don't see any changes, first make sure the view settings do not hide connections. If still no change is apparent, then those objects are probably not supposed to be connected with the A key.

### Q, U: context sensitive disconnect

The Q key is used to disconnect two objects depending on the type of objects. Hold down the Q key, click on one object, holding the mouse button down, and drag to another object, releasing the mouse button on that object. Usually this disconnects the output ports of one object from the input ports of another object. For NetworkNodes, however, the Q key disconnects a NetworkNode from TaskExecuters as travelers, from FixedResources as travel gateways, and sets one-way of a travel path connection to "no connection" (red). You can also use the U key if you are left handed.

### S, K: central port connect

The S key is used to connect central ports of two objects. Central ports are used for referencing purposes, using the `centerobject()` command. Hold down the S key, click

on one object, holding the mouse button down, and drag to another object, releasing the mouse button on that object. You can also use the K key if you are left handed.

### **W, I: central port disconnect**

The W key is used to disconnect central ports of two objects. Hold down the W key, click on one object, holding the mouse button down, and drag to another object, releasing the mouse button on that object. You can also use the I key if you are left handed.

### **D: context sensitive connect**

The D key is a second key for context sensitive connecting. The `NetworkNode` and the `TrafficControl` both implement this connection.

### **E: context sensitive disconnect**

The E key is a second key for context sensitive disconnecting. The `NetworkNode` implements this connection.

### **X: context sensitive click/toggle**

The X key is used to change an object or view information on the object, dependent on the type of object. Hold the X key down, and click on the object. The `NetworkNode` will toggle the whole network through different viewing modes. The X key also creates new spline points on a network path. Racks will also toggle through different viewing modes. A conveyor will reposition downstream conveyors to be flush with conveyor end points.

### **B: context sensitive click/toggle**

The B key is an additional key used to change an object or view information on the object, dependent on the type of object. Hold the B key down, and click on the object. The `NetworkNode` will toggle the whole network through different viewing modes. The `TrafficControl` also uses the B key.

### **V: view input/output port connections**

The V key is used to view an object's input/output port connections. Hold the V key down, and click on an object, holding both the V key and the mouse button down. If the mouse button is released first, then the information will disappear, but if the V key is released first, the information will persist.

### **C: view central port connections**

The C key is used to view an object's central port connections. Hold the C key down, and click on an object, holding both the V key and the mouse button down. If the mouse button is released first, then the information will disappear, but if the C key is released first, the information will persist.

### **F: create library objects**

The F key is used to quickly create library objects. Select an object in the library icon grid by clicking on it. Then click in the ortho/perspective view, and press and hold the F key down. Then click in the ortho view in the location you would like to create the object. The object will be created at that position.

**R: create and connect library objects**

The R key is like the F key, except it also connects consecutively created objects with an A connection.

**G: fast properties switching**

The G key lets you switch the focus of a parameters or properties window to different objects in the model. The ortho view remembers the last parameters/properties window that was opened. If you click in the ortho view to activate it, then hold the G key down and click on an object, the last opened properties window will update its focus to the new object.

## Menus and Settings

### View Window Settings for Ortho/Perspective View



This dialog box can be opened by selecting the Settings | More Settings... menu option on the Ortho or Presp view windows. It lets you configure the orthographic or perspective view's visual display settings. These settings apply only to the window that opened the dialog. They will not change any other views.

#### General

These parameters affect the general appearance of the view window.

#### Connections

**Show Connections** - If this box is checked, the ports and port connection lines will be displayed in the view window. Hiding these connections often makes it easier to see what is happening in the model. If a model is slowing down, often un-checking this box will help speed it up.

**Connector Size** - This number sets how large the port connectors are on the object.

**Connector Color** - This value sets the color of the connector lines in the view.

#### Grid

**Show Grid** - If this box is checked, the grid will be drawn in the view window.

**Snap to Grid** - If this box is checked, objects will automatically move to the nearest grid line when they are moved in the model. This is useful for placing objects in precise locations. Resizing of objects will also snap to the grid if this is checked.

**Grid Size** - This value sets how far apart the lines on the grid are in the view window. If this number is too small, the lines will be too close together and may slow down the model on some computers.

**Grid Fog** - This value lets you have the view's grid fade into the background color as it gets further from the viewpoint. Usually this is only useful on a perspective view. Set the value between 0 and 1, 0 meaning no fade, 1 meaning full fade.

**Grid Line Width** - This value sets the width of the view's grid lines.

**Grid Line Color** - This value sets the color of the grid lines.

## Names

**Show Names** - If this box is checked, the names and basic statistics of the objects will be drawn in the view window. Showing names can make it easier to see what is happening in the model as it runs, but can also decrease the view's refresh rate for large models.

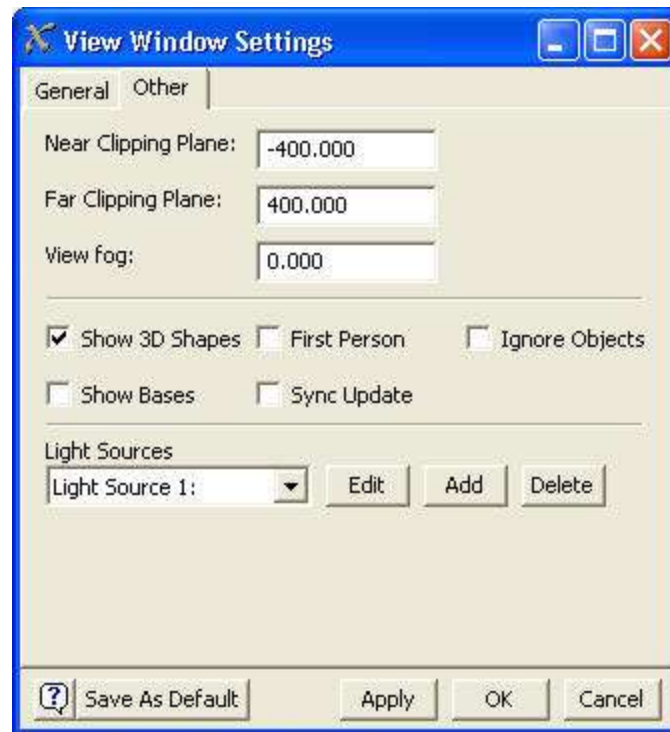
**Name Size** - This number defines how large the name of the object will be drawn in this view window. The basic statistics that are drawn beneath each object will also be affected the same amount.

**Name Style** - This pull-down list lets you choose where the name of the object will be drawn. Choose either below the object or in the center of the object.

## Background Color

**Background Color** - This option lets you select the color of the view window's background from a standard Windows color-choosing dialog box.

## Other Settings



**Near Clipping Plane** - This value sets the value for the nearest objects that will be drawn.

**Far Clipping Plane** - This value sets the value for the farthest objects that will be drawn.

**View Fog** - This value sets the view's fog value. View fog causes objects that are far away from the camera position to fade into the background color. Set the value between 0 and 1, 0 meaning no fog, and 1 meaning complete fog. This is usually only applicable to a perspective window.

**Show 3D Shapes** - If this box is checked, the 3d shapes (.3ds files or .wrl files) for all the objects in the model will be drawn in the view window. Some objects do not have 3d files associated with them, they are generally drawn directly with OpenGL. These objects will not be affected by this box.

**Show Bases** - If this box is checked, the 2D bases of the objects will be shown in the view window. Hiding bases generally makes the 3D views look better.

**First Person** - If this box is checked, the view window's mouse controls will be in first person mode. This means that the view will rotate around the user's view point, and not around a point in the middle of the screen. This mode is most useful when navigating in fly-through mode.

**Sync Update** - If this box is checked, all open view windows will be updated at the same time. If it is not checked, some windows may not be updated until an action is

completed in a different window. Checking this box may cause the program to run a little slower.

**Ignore Objects** - If this box is checked, the user will not be able to click on any objects in the view window. This is useful for navigating around a model that is completed, as the user will not be able to accidentally move any objects.

## Light Sources

These controls allow a user to add, edit and create light sources for the view window

**Light Sources** - The pull-down list contains all of the light sources that are currently in the view window.

**Edit** - This button opens the Light Source Editor dialog box for the light currently showing in the pull-down list.

**Add** - This button will create a new light source in the view window.

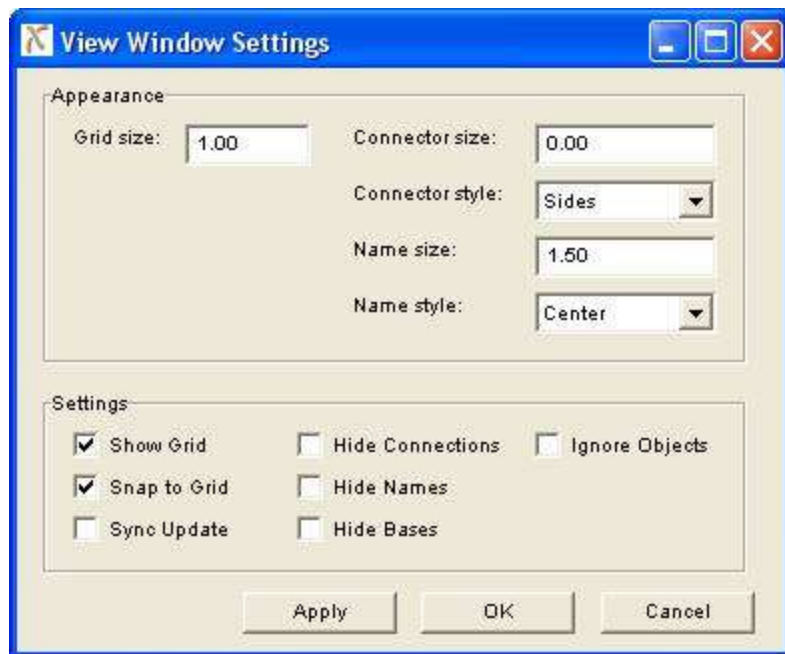
**Delete** - This button will delete from the view window the light source currently showing in the pull-down list. There must always be at least one light source in the model.

## Save As Default Button

The "Save as Default" button lets you save this view and its configuration as the default ortho/perspective view to be opened when you press the Ortho or Persp button on Flexsim's main toolbar. Note that this only applies to the current model you are working on. If you create a new model, default settings will revert back to the original.

---

## View Window Settings for Planar View



This dialog box can be opened by selecting the Settings menu option on the Planar view window. These settings apply only to the window that opened the dialog. They will not change any other views.

### Appearance

These parameters affect the appearance of the view window.

**Grid Size** - This value sets how far apart the lines on the grid are in the view window. If this number is too small, the lines will be too close together and may slow down the model on some computers.

**Connector Size** - This number sets how large the port connectors are on the object.

**Connector Style** - This pull-down list allows the user to set whether the add/delete port buttons will all be drawn above the object, or is they will be drawn on different sides of the object.

**Name Size** - This number defines how large the name of the object will be drawn in this view window. The basic statistics that are drawn beneath each object will also be affected the same amount.

**Name Style** - This pull-down list allows the user to choose where the name of the object will be drawn. Choose either below the object or in the center of the object.



## Settings

These parameters affect various visual settings of the view window.

**Show Grid** - If this box is checked, the grid will be drawn in the view window.

**Snap to Grid** - If this box is checked, objects will automatically move to the nearest grid line when they are moved in the model. This is useful for placing objects in precise locations. Resizing of objects will also snap to the grid if this is checked.

**Sync Update** - If this box is checked, all open view windows will be updated at the same time. If it is not checked, some windows may not be updated until an action is completed in a different window. Checking this box may cause the program to run a little slower.

**Hide Connections** - If this box is checked, the ports and port connection lines will not be displayed in the view window. This often makes it easier to see what is happening in the model. If a model is slowing down, often checking this box will help speed it up.

**Hide Names** - If this box is checked, the names and basic statistics of the objects will not be drawn in the view window. This object makes it easier to see what is happening in the model as it runs.

**Hide Bases** - If this box is checked, the 2D bases of the objects will not be shown in the view window. This generally makes the 3D views look better.

**Ignore Objects** - If this box is checked, the user will not be able to click on any objects in the view window. This is useful for navigating around a model that is completed, as the user will not be able to accidentally move any objects.

## Views Toolbar

The Views toolbar allows you to capture the view's current camera position and add it to a list of views.

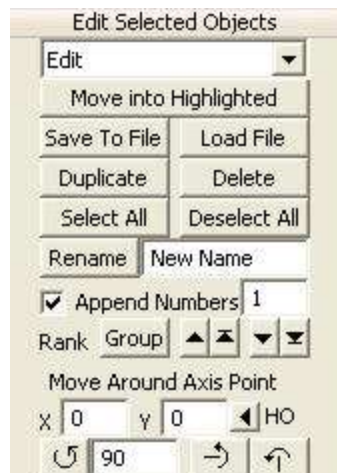


Press Capture to capture the current view. When a view position is captured it is added to a list in the drop-down box. Select Delete, and the selected view will be deleted. Type the name of the view and press Set Name to change the name of the view.

**Note:** There are two globally saved lists of captured views in a given model, the orthographic view list and the perspective view list. Hence all orthographic views will have a different list than the perspective views.

## Edit Selected Objects Toolbar

The Edit Selected Objects Toolbar can be found in the toolbar of any view window (Planar, Ortho, Perspective, or Tree). It offers several options that are performed on the currently selected set of objects in that view window. To select a group of objects, the shift or control key is held down while the user clicks on objects. Objects in the selection set will have a red box drawn around them. The currently highlighted object (the last object you clicked on) will have a yellow box drawn around it.



**Move into highlighted object** - This option moves the selected objects (the ones with the red box) into the highlighted object (the one with the yellow box). This allows the highlighted object to be used as a container.

**Save to file** - The selected objects will be saved to a file with a .t extension that can later be reopened in Flexsim. This allows users to save and import parts of models as needed.

**Load from file** - This object loads a .t file into the currently highlighted object. The highlighted object then becomes a container for the imported objects.

**Duplicate** - This option creates an identical copy in the model of the selection set. All port connections are kept intact.




**Delete** - This option deletes the selected objects.

**Select All** - This option adds all objects in the model to the selection set.

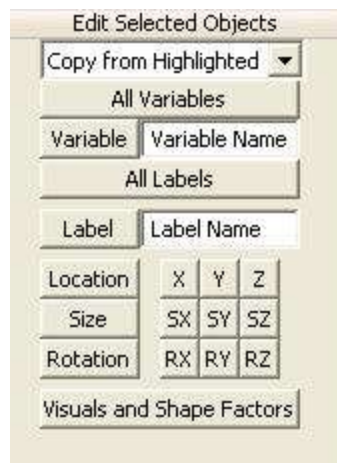
**Deselect All** - This option takes all objects in the model out of the selection set.

**Rename** - Enter a name and a starting number and press the rename button to rename all selected objects with appended ascending numbers.

**Rank** - This series of buttons lets you group the objects together in the tree, as well as move them up and down.

**Move Around Axis Point** - This set of controls lets you manipulate the locations and rotations of the selected object set. All operations are done around an axis location in the model. You can either enter this in, or highlight an object in the model and click the "< HO" button to get that object's location. Enter an angle of rotation in degrees and press the  button to rotate the selected set around the axis point. Click the  or  buttons to flip the selected objects over the axis point either vertically or horizontally, respectively.

**Copy from Highlighted** - These options copy information from the highlighted object (the one with the yellow box) to all of the objects in the selection set (the ones with the red box). If C++ functions are copied (such as variables), the model will need to be recompiled before running it again.



The following data can be copied:

- Single variable or label. Type in the name of the variable or label to be copied and press the "Variable" or "Label" button.
- All variables, including all C++ functions (such as triggers).
- All labels
- Spatial values. These buttons allow you to copy spatial values from the highlighted object. You can copy the whole x/y/z size/location/rotation with one operation, or copy individual x, y, or z attributes.
- Visuals and Shape factors. This allows you to copy shape factors from the highlighted object in the same manner you can copy spatial values. It also copies visual information such as 3D shapes, 2D shapes and color.

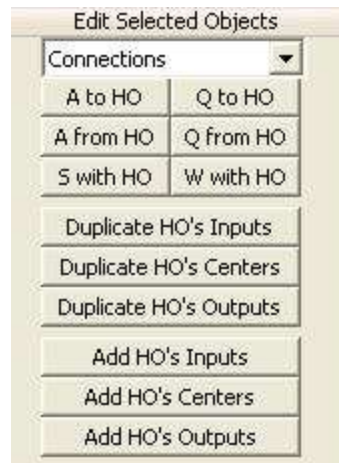
**Switches** - These options allow you to turn on and off various settings on the selected objects. For all of the objects in the selection set, the chosen setting will be reversed (toggled). That is, if it was on it will be turned off, and if it is off it will be turned on.



The available settings to toggle are:

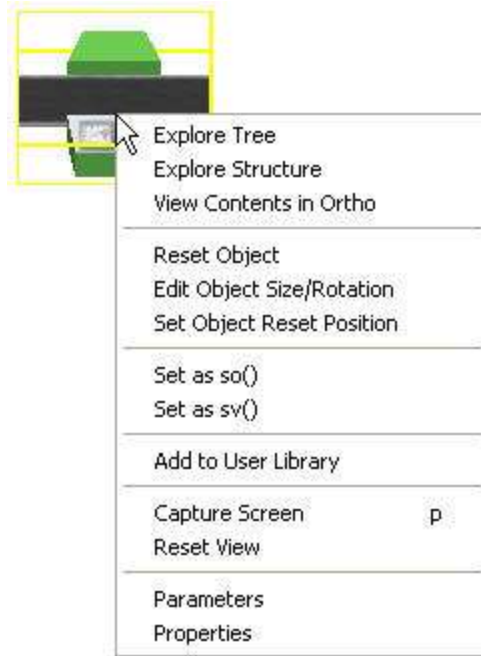
- Display the objects' names and labels.
- Display the objects' ports and port connections.
- Display the objects' 3D shapes. Note that objects drawn with OpenGL will not be hidden.
- Display the objects' 2D shapes (bases).
- Display the content of the objects. If this is off, objects inside the main object will not be seen.
- Activate stats collection for the objects.
- Protect the objects. This locks the objects' location, size and rotation.

**Connections** - This sub-menu allows you to make drag connections between the highlighted object and the selection set.



- A connect to highlighted object - this option makes an 'A' drag connection from all objects in the selection set to the highlighted object.
- A connect from highlighted object - this option makes an 'A' drag connection from the highlighted object to all objects in the selection set.
- S connect with highlighted object - this option makes an 'S' drag connection from all objects in the selection set to the highlighted object.
- Q disconnect from highlighted object - this option makes a 'Q' drag connection from the highlighted object to all objects in the selection set.
- Q disconnect to highlighted object - this option makes a 'Q' drag connection from all objects in the selection set to the highlighted object.
- W connect with highlighted object - this option makes a 'W' drag connection from the highlighted object to all objects in the selection set.
- Copy highlighted object's connections - this option creates copies the highlighted object's input port, output port, and central port connections to all objects in the selection set.

## Ortho/Perspective Popup Menu



**Explore Tree** - This option brings up a tree window that explores the object as a tree. If no object is selected, tree window will appear exploring the model.

**Explore Structure** - This options brings up a tree window exploring the tree structure of the orthographic window itself.

**Reset Object** - This resets the x/y/z rotation and the z location of the object to 0.

**Edit Object Size/Rotation** - This shows the object's x/y/z axes. Click and drag on of the axes to change size. Right-click and drag to change rotation.

**Set Object Reset Position** - This saves the object's current position so that on every reset, the object will go back to that position.

**View Contents in Ortho** - This option brings up a new orthographic window that shows the contents of the object selected. Usually you will only do this for a VisualTool object that acts as a container of other objects. If no object is selected, then a new orthographic model view will open.

**Set as Selected Object so()** - This object designates the object as so(). You will usually use this option for writing code in the script window.

**Set as Selected View sv()** - This selects the ortho window as the selected view. You will usually use this option for writing code in the script window.

**Add to User Library** - This option adds the selected object to the currently active user library.

**Capture Screen** - This option captures the current screen, saving it as a bitmap file in the prints folder of flexsim. This can also be done by selecting on the active view and pressing the 'P' key.

**Reset View** - This returns the camera position of the view to its initial position.

**Parameters** - This option opens the object's parameters window.

**Properties** - This option opens the object's properties window.

---



## Light Source Editor



**X Position** - This number is the position along the X axis of the model where this light source is located. This number is relative to the position of each object.

**Y Position** - This number is the position along the Y axis of the model where this light source is located. This number is relative to the position of each object.

**Z Position** - This number is the position along the Z axis of the model where this light source is located. This number is relative to the position of each object.

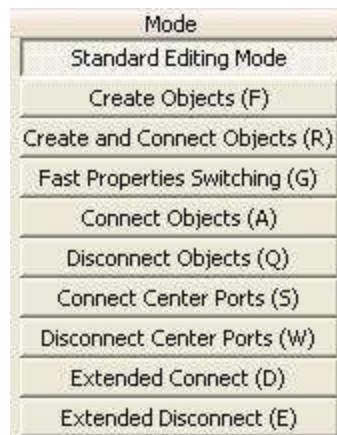
**Color** - This allows the user to choose the color of the light from this light source from a standard Windows color-choosing dialog box.

**As point source** - If this box is checked, the light generated by this light source appears to be coming from a specific point, relative to the camera. If it is not checked, the light appears to be coming from a given direction only.

## Model View Toolbars

### Mode Toolbar

The mode toolbar lets you toggle between different modes in your ortho/perspective view. You enter a mode by pressing the appropriate mode button, and then can exit the mode either by right-clicking in the view, or by pressing a different mode button.



**Standard Editing Mode** - This mode is the standard mode for your view. Here you can move objects around, size them, etc.

**Create Objects** - Once you are in this mode, you can simply click on an object in the library, and then each time you click the mouse's left button in your ortho view, a new instance of that object will be created. You can also enter or exit this mode by pressing or releasing the F key.

**Create and Connect Objects** - This mode is similar to the previous mode, except that when a new object is created, it will be connected with the previous object that was created ('A'). You can also enter or exit this mode by pressing or releasing the R key.

**Fast Properties Switching** - This mode allows you to quickly switch between the Properties/Parameters of several objects without having to close and open new windows. Once you are in this mode, open an object's Parameters or Properties window, and then click on a different object in the model, and the window will update to the new object's Parameters/Properties. You can also enter or exit this mode by pressing or releasing the G key.

**Note on Fast Properties Switching** - When you switch between Parameters/Properties of different objects, the previous Properties/Parameters are NOT inherently applied before the switch, so if you are making changes, you will need to press the Apply button to apply the changes.

**Note on getting out of Fast Properties Switching** - If you are in this mode, a right-click will NOT get you out of the mode. Since you need a right-click to open a Properties

window, exiting the mode with right-click is disabled. You will need to press another mode button in order exit the mode.

**Connecting/Disconnecting Objects** - These modes allow you to connect and disconnect objects without using the usual click-hold-drag-release method of connecting. Just click on one object, then click on another object, and they will be connected. There are modes for A,Q,S,W,D, and E connecting. You can also enter or exit this mode by pressing or releasing the appropriate key.

---

## Travel Networks Toolbar

This toolbar lets you edit the default connecting settings for connecting Network Nodes in your model.



You can select default connecting as two way or one way, passing or nonpassing, straight or curved. This affects what type of connection will be created when you do an 'A' drag connect between two network nodes.

## Model Layouts Toolbar

This toolbar lets you create and edit certain layouts for your model. For example, you may want to save a layout that represents a left-to-right flow chart of the steps in your model, where model flow proceeds always from left to right. Then you may want to save another layout that represents an actual factory layout. This toolbar lets you do that.



Press Add Layout or Delete Layout to add or delete a layout. By default, when you add a layout, it will save the current layout of the model. Enter the name of the layout and press Set Name to change the name of the layout. Press Set to save the layout base on the current locations of objects in the model. Select a layout in the drop-down box, and the model will immediately go to that layout.

## Edit Highlighted Object Toolbar



This panel allows you to edit the object that is *highlighted*. *Highlighted objects have a yellow box around them.*

**Parameters/Properties** - To open the Parameters or Properties window for the object, click on the Parameters or Properties button.

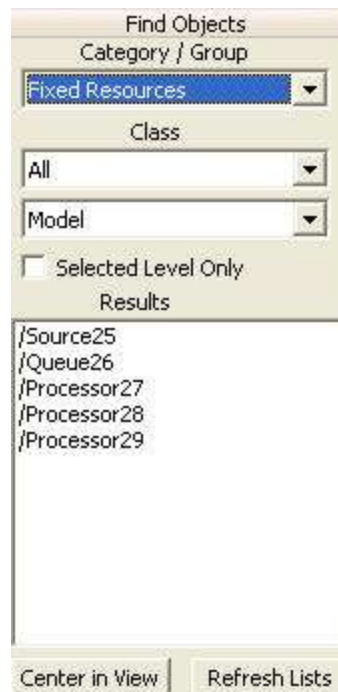
**Ranking** - Click on the two up arrows to either move the object one up in the tree or to the front of the tree. Press one of the two down arrows to either move the object one down in the tree, or move it to the end of the tree. Enter a rank number and press Set Rank to set the rank of the object. Press Get Rank, and the object's rank will be written to the text to the right of the button.

**Select/Unselect** - Press Select to add the object to the selection set. Press Unselect to remove the object from the selection set.

**Context Click** - The last three buttons simulate clicking on the object while holding a key down. Refer to keyboard interaction for more information.

## Find Objects Toolbar

This toolbar lets you quickly find objects in your model based on the type of object.



You can select the library category, or the class type, or a specific layer in the model that you want to find the objects in. When you select an option from the drop-down, the list at the bottom will be re-populated. Select an object and press Center in View to go to that object in the Ortho/Perspective/Tree view.

## Groups Toolbar

The groups toolbar lets you create and edit different groups in your model.

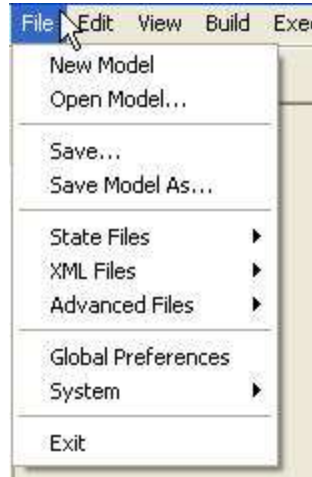


Press Add Group or Delete Group to add or delete a group. Enter the name and press Set Name to change the name of the group. Select object in the model using the Ctrl or Shift lasso method, then press Set or Add to set or add those objects to the group. Press Select to select the objects in the group. Press Select Only to select the objects in the group and unselect all other objects in the model.



# Main Menu and Toolbar

## File Menu



**New Model** - This option clears the current model so that a new one can be created. It does not affect the view layout or library. A warning will be displayed that if you proceed you will lose all objects in your model.

**Open Model...** - This option allows the user to choose a Flexsim Model File (.fsm extension) to edit. Any changes that have been made to the current model that have not been saved will be lost.

**Save...** - This option saves the current Model File (.fsm extension). Any changes that have been made to the current model will be saved.

**Save Model As...** - This option allows the user to save the model to a file. The file that is created has the extension .fsm. The entire contents of the model tree will be saved.

**State Files** - This option allows the user to save the state (current model run) of the model, or load a saved state model to continue the run. Saving state is helpful when your model is in the middle of a simulation run and you want to save it in its current state (all flowitems remain where they are and resources remain in their current state of operation), and then later load the model's state and be able to continue running the simulation from that point.

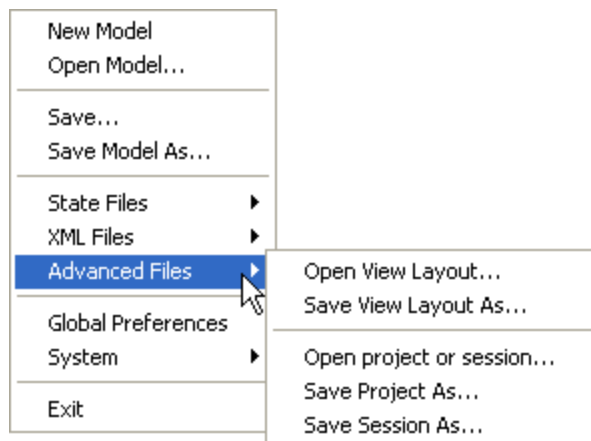


**Save State** - This option saves the current state of the model.

**Load State** - This option loads and compiles a saved state model.

**XML Files** - This submenu allows you to open and save files in XML format instead of the standard Flexsim binary format. You can open model, project, view layout, or session files with this option.

**Advanced Files** - This option allows the user to load or save a View Layout, Project, or Session (both view layout and project).



**Open View Layout** - This option allows the user to choose a Flexsim View Layout File (.fsv extension) to open. This will change the layout of the windows in the program. It will not affect the model, or library.

**Save View Layout** - This option allows the user to save a Flexsim View Layout File (.fsv extension). A view layout will save the view tree.

**Open a project or session** - This option allows the user to choose a Flexsim Project or Session File (.fsp or .fss extension) to open. Whenever you open a project or session, you will need to compile the entire session from the View menu, instead of pressing the default Compile button. The default Compile

button only compile code in the model, whereas the Compile Entire Session option in View|System Maintenance will compile both the library and the model.

**Save Project As...** - This option allows the user to save a project file (.fsp). A project file will save the main tree which includes the system, library, and model..

**Save Session As...** - This option allows the user to choose a Flexsim Session File (.fss extension) to save, or open. This will replace the current view layout, model, library, and other global information. A session saves both the project and the view into one file.

**Global Preferences** - This option brings up the preferences window, allowing you to set preferences in Flexsim. You can also configure text highlighting properties here.

**System** - This option allows you to manually reload media, as well as disconnect any DLLs that have been linked to DLL toggled nodes.

**Exit** - This option will close Flexsim without saving any files. If the user wishes to save any changes, the appropriate file(s) should be saved before this option is selected.

## Edit Menu



**Lock Splines** - This option will lock splines for all network paths that are currently in the model. Locking splines will considerably increase the run speed of a simulation, but you will not be able to change the paths of the network. This will be reset to unlocked whenever you compile the model.

**Unlock Splines** - This option will unlock the splines for network paths. Once the spline is unlocked it can be edited graphically with the mouse.

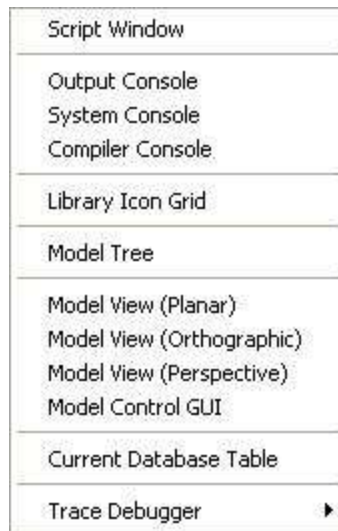
**Set Spline Tension** - This option brings up a dialog box to set the tension of splines in the model. This value should be between 0 and 1. A value of 0 will cause all splines to be totally straight passing directly through the nodes, and a value of 1 maximizes the curvature of the splines.

**Resize and Rotate Objects** - Check this option to have object axes shown in ortho and perspective views for sizing and rotating.

**Set Number Precision** - This option brings up a dialog box to set the number precision. This is the number of decimal points to be shown for numbers in the current Flexsim session.

**Find Replace** - This option brings up the Find and Replace dialog.

## View Menu



**Scripting Window** - This option opens the scripting window. The scripting window is an interactive window to execute Flexscript commands manually.

**Output Console** - This option opens a window where output information is shown. You can print your own information to the output console using the commands: `pt()`, `pr()`, `pd()`, `pf()`, etc. For more information on these commands, refer to the command summary.

**System Console** - This option opens a window where information about the status of the program engine is printed. If this window is open, the model may move sluggishly or slowly.

**Compiler Console** - This option opens a window where information is printed while the model is compiled. This shows the status of each step of the compilation process.

**Command Hints** - This option opens the command hints window, which lists available commands and the parameters that they require. It is not as detailed a list as the Command Summary window, which is available through the toolbar.

**Attribute Hints** - This option opens the attribute hints window, which lists available object attributes and a brief description of the meaning of each of them.

**Completion Hints** - This option opens a completion hints window that lists the possible commands that a user could be typing. This list is updated as the user types.

**Library Icon Grid** - This option opens the Library Icon Grid with an icon grid of the objects that can be dragged into the model.

**Model Tree** - This option opens a tree window that shows the model folder. This shows all of the objects that are in the model. The tree can be manipulated from this view.

**Model View (Planar)** - This option opens a planar model view that shows the model in a two dimensions. Objects' 3D shapes are not drawn in this view. The view cannot be rotated and objects are not shown as rotated. If the ortho and perspective views are causing the model to run too slowly, this view may speed it up.

**Model View (Orthographic)** - This option opens a new orthographic view window of the model.

**Model View (Perspective)** - This option opens a new perspective view of the model.

**Current Database Table** - This option brings up a window that shows the currently active database table that was opened or queried with the `dbopen()`, `dbchangetable()`, `dbsqlquery()` commands. Refer to the command summary for more information on these commands.

## Trace Debugger

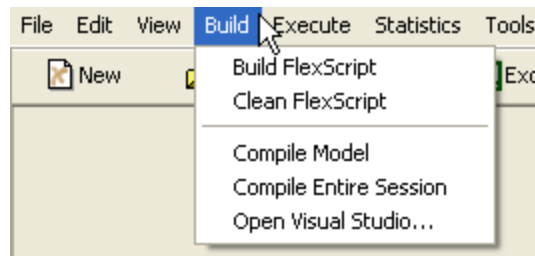
This sub-menu contains the information to adjust trace settings, turn the trace on, and open the trace console.

**Settings...** - This option brings up the Trace Debugger, allowing you to select event types to be displayed in the trace window.

**Trace On** - This option toggles the trace on or off. Trace information is displayed in the trace console. Also, if the trace is on, then the System Console will print more detailed information for exceptions.

**Trace Console** - This option opens a window where information is printed about events that occur on objects selected in the model. For the information to be printed "Trace On" must be checked.

## Build Menu



**Build Flexscript** - Builds all flexscript code.

**Clean Flexscript** - Clears out all flexscript built code.

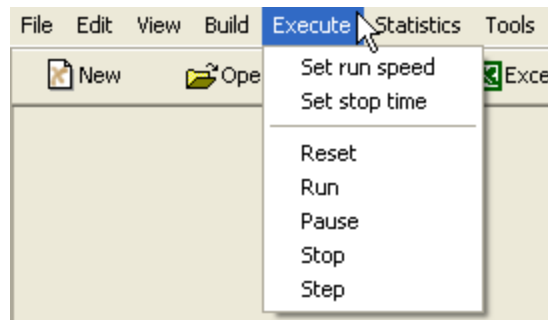
**Compile Model** - Compiles all c++ code in the model.

**Compile Entire Session** - Compiles all c++ code in the main tree. You will not need to do this unless you are developing your own application with Flexsim.

**Open Visual Studio...** - Opens Microsoft Visual Studio.

**Make all Code C++/Flexscript** - There are two options to make all code either C++ or Flexscript. We provide these options so that modelers can have both the ease of use of Flexscript (code works immediately when editing in Flexscript, without having to compile) as well as the run-speed of C++ (since it is compiled, it runs much faster than Flexscript). While in the model building phase you can use Flexscript, so that your code is interpreted immediately after you write it. Then, once your model is ready to run, you can choose the Build|Make all code C++ option, compile, and run to get the speed of C++.

## Execute Menu



**Set run speed** - This option brings up a dialog box that lets you set the run speed of the model.

**Set stop time** - This option brings up a dialog box that lets you set the stop time of the model. The model is stop with the very first event occurring after the designated time.

**Reset Model** - This option resets the current model file and prepares it to run. Same as selecting the Run button on the simulation run panel.

**Run Model** - This option runs the current model file. Same as selecting the Run button on the simulation run panel.

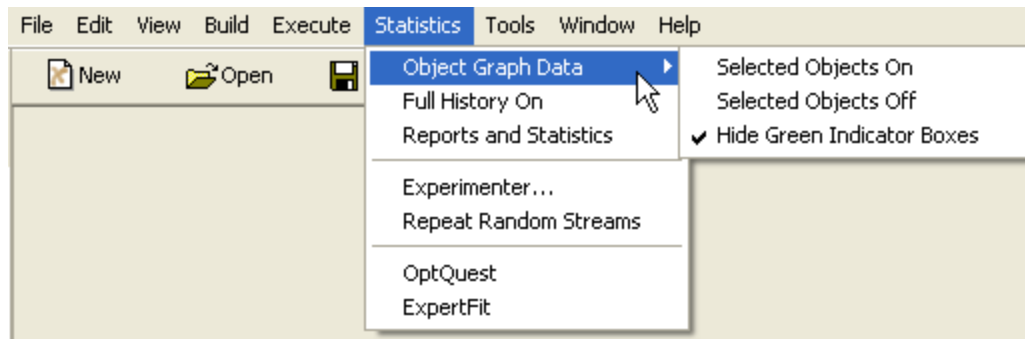
**Pause Model** - This option pause the current model. Same as selecting the Pause button on the simulation run panel.

**Stop Model** - This option stops the current model. Same as selecting the Stop button on the simulation run panel.

**Step one event** - This option runs the model to the next event. At the next event the model will stop. Same as selecting the Step button on the simulation run panel.



## Stats Menu



**Object Graph Data** - This option opens a submenu to turn on and off the real-time content and staytime graph data collecting in Flexsim.

**Selected Objects On** - This option will turn on the local stats for all selected objects (those currently indicated with a red bounding box).

**Selected Objects Off** - This option will turn off the local stats for all selected objects (those currently indicated with a red bounding box).

**Hide Green Indicator Boxes** - If this option is checked, the green boxes that appear around objects that are collecting statistics will be hidden.

**Full History On** - This option turns full history collection on or off. If on, Flexsim will record into a database any product movement or state changes. This database can then be exported to Flexsim's charting application for further analysis.

**Standard Report** - This option brings up a dialog box to create a standard output report. Standard reports are exported in comma separated value (csv) file format, and will automatically be opened and viewed in Microsoft Excel or whichever program on your computer has been set as the default for opening csv files.

**State Report** - This option creates and opens a state report for all *selected objects*. *Selected objects are objects that appear with a red box around.* in the model. The state report is exported in comma separated value (csv) file format, and will automatically be opened and viewed in Microsoft Excel or whichever program on your computer has been set as the default for opening csv files.

**Model Documentation** - This option opens the model documentation dialog and is used to save information about your model to an external text file.

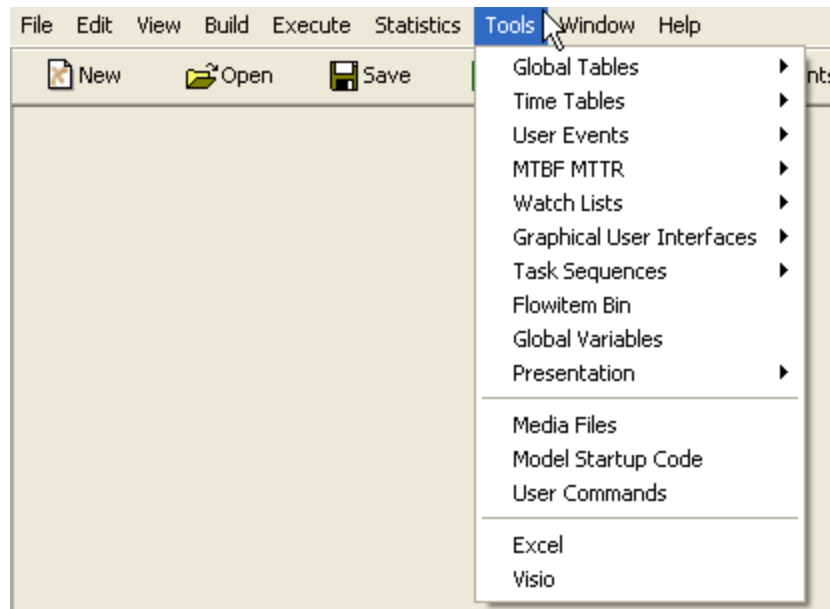
**ExpertFit** - This option will start the ExpertFit software developed by Averill Law & Associates. A complete user manual is available online within the ExpertFit software. ExpertFit is a program used to determine the best statistical probability distribution to match your input data.

**Experimenter** - This option brings up the Simulation Experiment Control input window. This is the same as pressing the Experimenter button on the lower right of the main window. The Experimenter is used to perform multi-run multi-scenario analyses of your model.

**Repeat Random Streams** - This option repeats the random streams every time you reset and run the model. To make each run different unselect this option.

**OptQuest** - This option opens the Optimizer editor. The optimizer is used to find the optimum values for a set of user-defined decision variables so as to either minimize or maximize a user-defined objective function within the constraints defined by the user. The OptQuest application developed by OptTeK Systems, Inc. is fully integrated into Flexsim, but requires a special license sold as an add-on to the basic Flexsim software license. Please contact Flexsim Software Products, Inc. for a license.

## Tools Menu



**Global Tables** - This sub-menu provides functionality for adding, deleting and editing Global Tables in your model.

**Time Tables** - This sub-menu provides functionality for adding, deleting and editing Time Tables in your model.

**User Events** - This sub-menu provides functionality for adding, deleting and editing User Events in your model.

**MTBF MTTR** - This sub-menu provides functionality for adding, deleting and editing MTBF MTTR objects in your model.

**Watch Lists** - This sub-menu provides functionality for adding, deleting and editing Watch Lists in your model.

**Graphical User Interfaces** - This sub-menu provides functionality for adding, deleting and editing Graphical User Interfaces in your model.

**Task Sequences** - This sub-menu provides functionality for adding, deleting and editing User Task Sequences in your model.

**FlowItem Bin...** - This option opens the FlowItem Bin that allows the user to edit the master FlowItem list for the model. This is the same as pressing the FlowItem button on the toolbar.

**Global Variables** -

**Presentation** - The presentation menu provides options for creating a presentation using Flexsim.

**Sky Box** - This option adds a Sky Box to the model and opens the Sky Box editor. Sky boxes look best in the perspective window.

**AVI Maker** - This option opens the AVI Maker.

**Presentation Builder** - This option opens a special perspective view called the Presentation Builder that allows you to develop fly through presentations.

**Excel...** - This option opens the Excel Interface window that allows the user to define the parameters for Excel imports and exports. This option is the same as pressing the Excel button on the toolbar.

**Visio...** - This option opens the Visio Import window.

## Window Menu

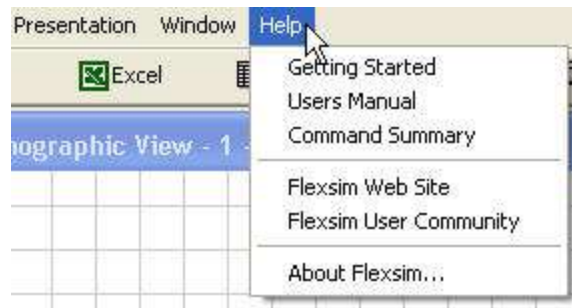


**Cascade** - This option will cascade the open windows for easy selection.

**Close All** - This option closes all of the active windows in Flexsim. This can be useful when a user has opened more windows than they can effectively work in.

**Other options** - The other options are the windows that are currently open in Flexsim. When a user selects one of these options, that window is brought to the front of the screen and is made the active window.

## Help Menu



**Getting Started** - This option opens the Flexsim Getting Started help section.

**Users Manual** - This option opens the Flexsim User Manual.

**Command Summary** - This option opens the command summary.

**Flexsim Web Site** - This option opens the Flexsim web site in a web browser.

**Flexsim User Community** - This option opens the Flexsim User Community web site in a web browser.

**About Flexsim...** - This opens a splash screen giving you information about Flexsim. It displays the version of Flexsim currently running, the user that this copy of Flexsim is registered to, graphics card info, and contact information.

## Flexsim Simulation Software

Copyright © 2001-2004 Flexsim Software Products, Inc. Orem, Utah  
All Rights Reserved

### Licensed to: Development User

User: AnthonyJohnson, Computer: ANTHONY

OpenGL: GeForce FX 5700 Ultra/AGP/SSE/3DNOW!, NVIDIA Corporation, 1.4.1

Engine Version: 3.0, 0.062 Built on 17-Dec-04

Library Version: 3.0\_prelease 7-Jan-2004 15:24


GUI Version: 3.0\_prelease 07-Jan-2004 15:24


Contact: support@flexsim.com, Tel: 801-224-6914





## Flexsim Toolbar





 **New:** This button closes the current model and allows you to begin building a new one. This button does not change the current project or view layout. If you wish to use a different project, use the "Open project or session" option in the main menu. To use a different view layout, use the "Open View Layout" option.


 **Open:** This button allows you to open a previously saved model (.fsm file). The model that is currently open in Flexsim will be discarded and the one you choose will be loaded. This option will not change the current project or view layout. If the you wish to open a different project, use the "Open project or session" option in the main menu. To open a different view layout, use the "Open View Layout" option.


 **Save:** This button allows you to save the current model. A dialog box appears confirming the current file that is being edited.


 **Excel:** This button opens the excel interface dialog box.

 **Flowitems:** This button opens the Flow Item Bin editor. In this window, the user can edit the types of flowitems currently available in the Flowitem Bin. Attributes of the flowitems, such as size or shape, can be changed. New flowitem classes can also be created, and unused ones can be deleted.


 **Hints:** This button opens the command hints window that provides hints on Flexscript/C++ commands.


 **Tree:** Pressing this button opens a tree view of the model. This is useful to see all of the objects that are currently in the model, even those that cannot be seen in the ortho and perspective view windows. Objects' attributes can also be edited here; however it is recommended that object parameters always be changed through the objects' parameters dialog windows, and not through the tree. This button should generally only be used by advanced users.

 **Ortho:** This button opens an orthographic view window. In this type of window, the objects do not appear smaller as they get farther from the user's viewpoint. This view is often used to build the model initially, so that all of the objects can easily be placed in the correct locations.

 **Persp:** This button opens a perspective or virtual reality view window. In this type of window, objects appear smaller as they get farther from the user's viewpoint. This view is most often used after the model has been laid out initially, so that the user can see how the model would look in reality.

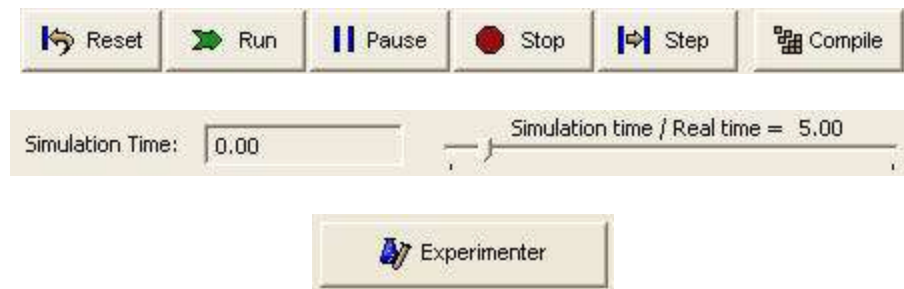


 **Script:** This button brings up a script editor. Here you can execute flexscript commands to change or get information from your model.

 **Control:** This button opens the model's control gui. This is a window that you can create yourself to do operations that are specific to your model. You can create this gui using Flexsim's GUI Builder.

## Simulation Run Panel

This panel is used to control a model's run. Most of the functions are available through the Execute menu in the main menu. The panel is found at the bottom of the main window in Flexsim.



**Simulation Time** - This displays the current model time. This value does not have any units attached to it and can be considered any unit of time (seconds, days, etc).

**Simulation Time/Real Time** - This slider defines the number of model time units that Flexsim will try to calculate per second of real time. The actual result may fall short of this value if the model requires too much processing at each event.

### Buttons



This option resets the model. The OnReset function is called for each object in the model. This should always be selected before running a model.



This option starts the model running. The model clock will continuously advance until the model is stopped, or there are no more events that need to happen.



This options stops the model running without updating the states of the objects in the model so that you can still query the length of time each object has been in its current state. The model is not reset. It can be started again from the exact point in time that it was stopped.



This option stops the model while it is running. It also updates the states of all objects in the model. The model is not reset. It can be started again from the exact point in time that it was stopped.



This option sets the model clock to the time of the next event that needs to occur. That event then happens. This allows users to step through the execution

of a model event by event. When many events are scheduled to occur at the same time, there may be no visible changes in the model when this option is selected.



This option resets and compiles the model



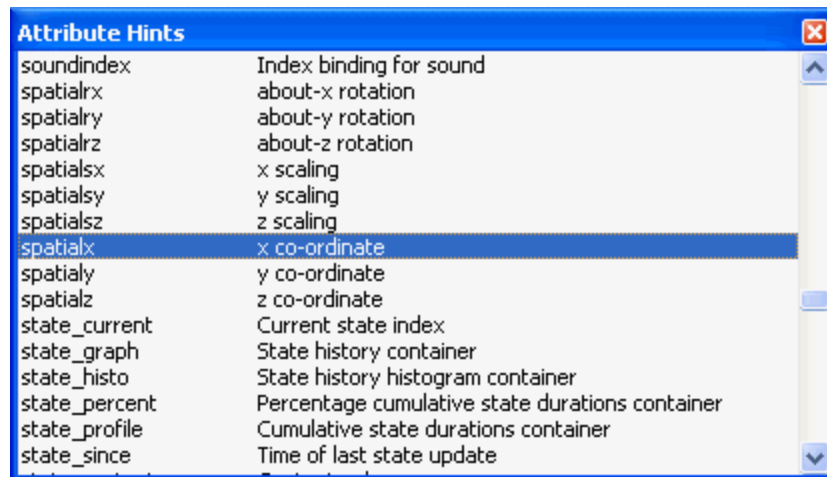
This button opens the Simulation Experiment Control dialog window. The Experimenter is used to run multiple iterations of a model as a single scenario or multiple scenarios as defined by the user.



# General Windows

## Attribute Hints

The attribute hints window shows the list of special Flexsim attributes and their meaning. Click inside the window and type the name of the attribute you would like to know about, and the window will scroll to that attribute.



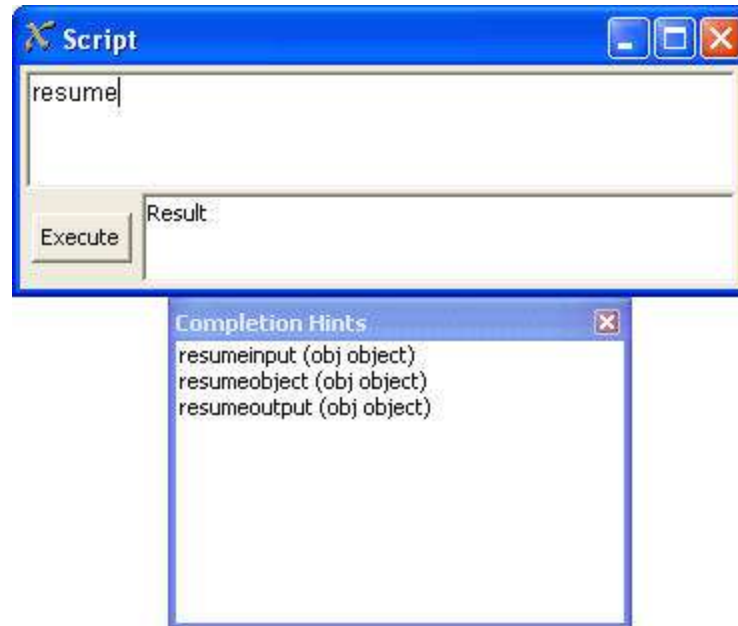
## Command Hints

In this window you can get quick hints on commands and what parameters they take. Click inside the window and type the name of the command you would like to know about, and the window will scroll to that command. Also, if this window is open while you are typing code in another window, it will automatically scroll to commands that you are typing in your code window.

Command Hints		
setcameravolume	view	(obj view, num w, num h, num n, num f, num mag, num fov, num op, num fp)
setchecked	view	(obj view, num value)
setcollisioncheck	objects_data	(obj taskexecutor, num checkstate[, num checkinterval])
setcolor	objects_data	(obj object, num red, num green, num blue)
setcurrent	objects_functions	(thing)
seteventtime	exec_events	(str EventName, num time, num timemode[1/2], num createevent[1/0])
setframe	objects_data	(obj object, num framenum)
setitem	objects_functions	(thing)
setitemtype	objects_data	(obj object, num value)
setlabelnum	objects_data	(obj object, str/num label, num value)
setlabelstr	objects_data_string	(obj object, str/num label, str value)
setloc	objects_data	(obj object, num x, num y, num z)
setmessageloopsleep	other	(num interval, num sleeptime)
setname	objects_data_string	(obj object, str value)
setnodename	objects_data_string	(obj/node object, str value)
setnoderum	objects_data	(node thenode, num value)

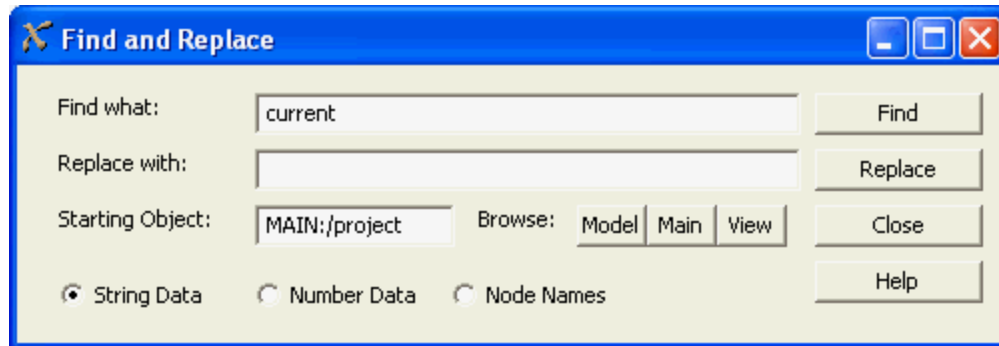
## Completion Hints

The completion hints window lets you get hints for commands while you are typing code. As you type code in a script editor or in a code window, the completion hints window will offer hints for the commands you type.



## Find Replace

The find/replace dialog is used to search for and replace text or numbers in the project. Type the text or number you want



**Find what** - Type here the text or number you would like to find, and then click the Find button.

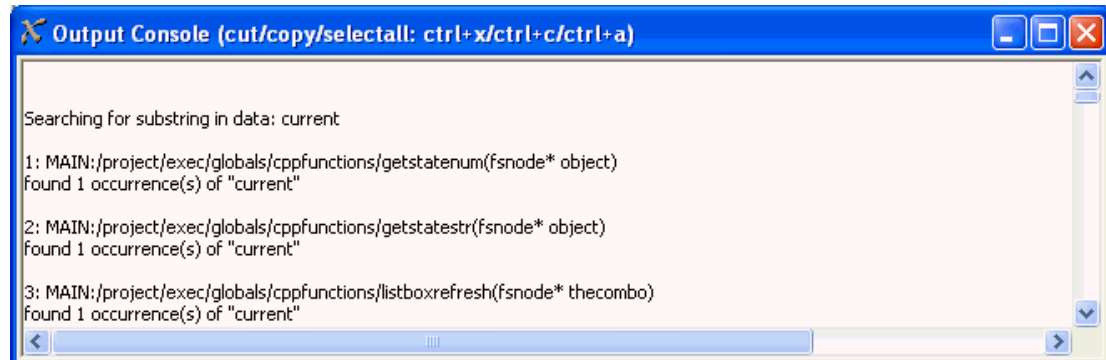
**Replace with** - Enter the text you want to replace, and then click the Replace button.

**Starting Object** - This is a path reference to a node from which to start the search. You can type the path in explicitly or browse for the node using one of the three browse buttons on the right, which will bring up a tree browse dialog.

**String Data/Number Data/Node Names** - Select one of these radio buttons to select which types of data to search in the tree.

## Output

Once you click on the Find or Replace button, the output console will appear, reporting information on the locations of data that was found and/or replaced.



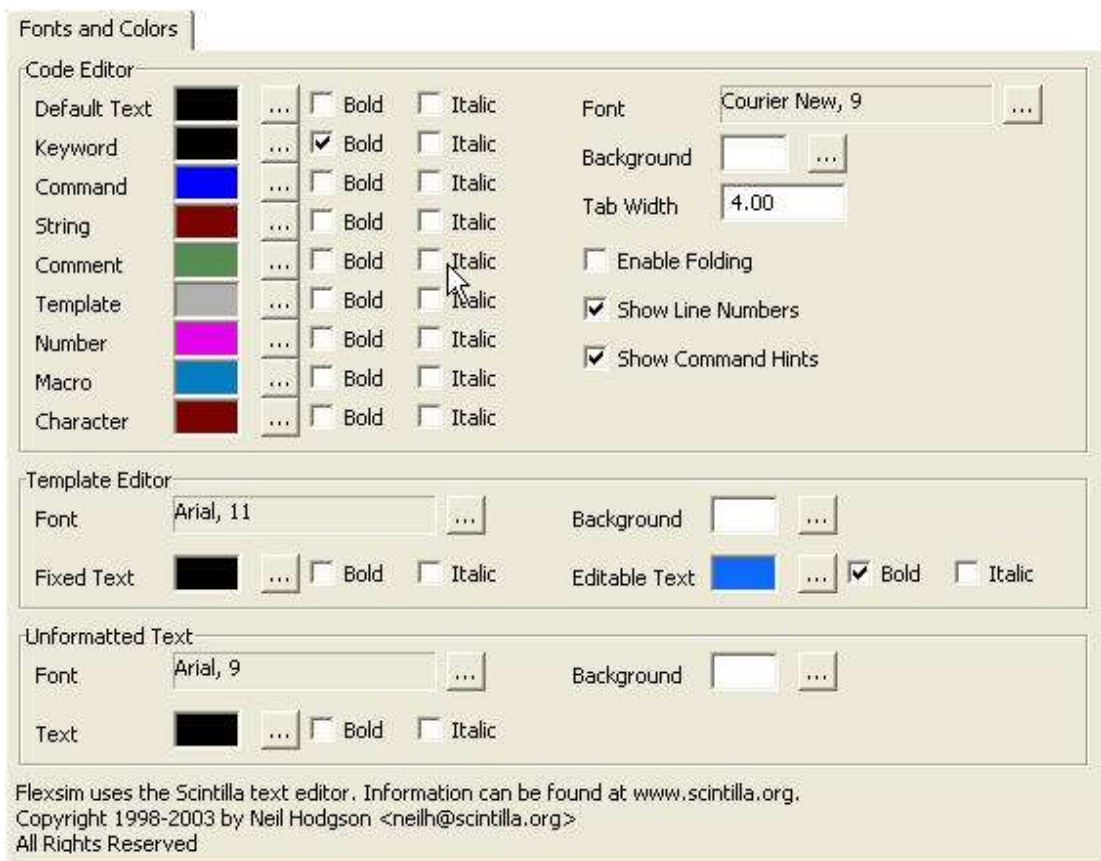


## Preferences Window

The preferences dialog window specifies default user preferences for Flexsim. These preferences are saved across several models. The dialog window is split into three tabs.

### Fonts and Colors

The Fonts and Colors tab specifies syntax highlighting and other settings used in Flexsim's implementation of the Scintilla code editor. You can also specify settings and colors for the template editor, as well as for unformatted text (which is used in multi-line unformatted text controls like the user description portion of the Properties window).



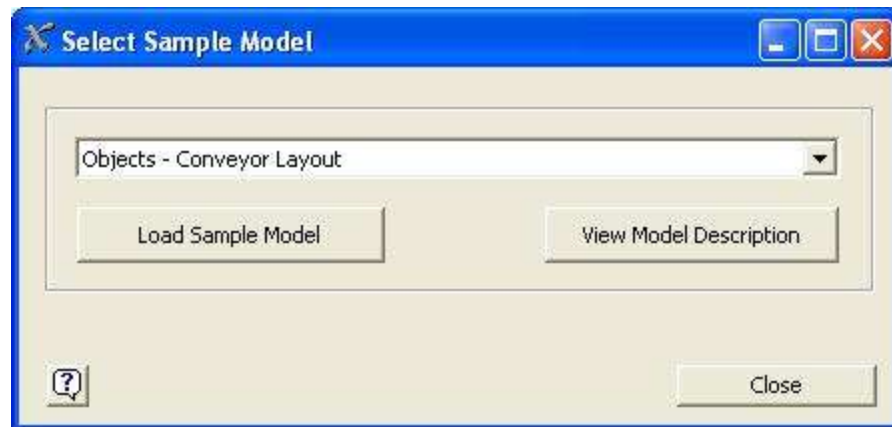
For more information on the Scintilla code editor, go to [www.scintilla.org](http://www.scintilla.org). The Scintilla text editor is under copyright 1998-2003 by Neil Hodgson <[neilh@scintilla.org](mailto:neilh@scintilla.org)>. All Rights Reserved.

**Font Style** - This specifies the font and size of text. Click on the "..." button on the right to select a font and size.

**Coloring** - This panel shows the colors that are used when highlighting code. Click on the button to the right of a color to change the color for a given code type.

**Highlighting** - In this panel you can specify which types of code are highlighted and which types are not highlighted.

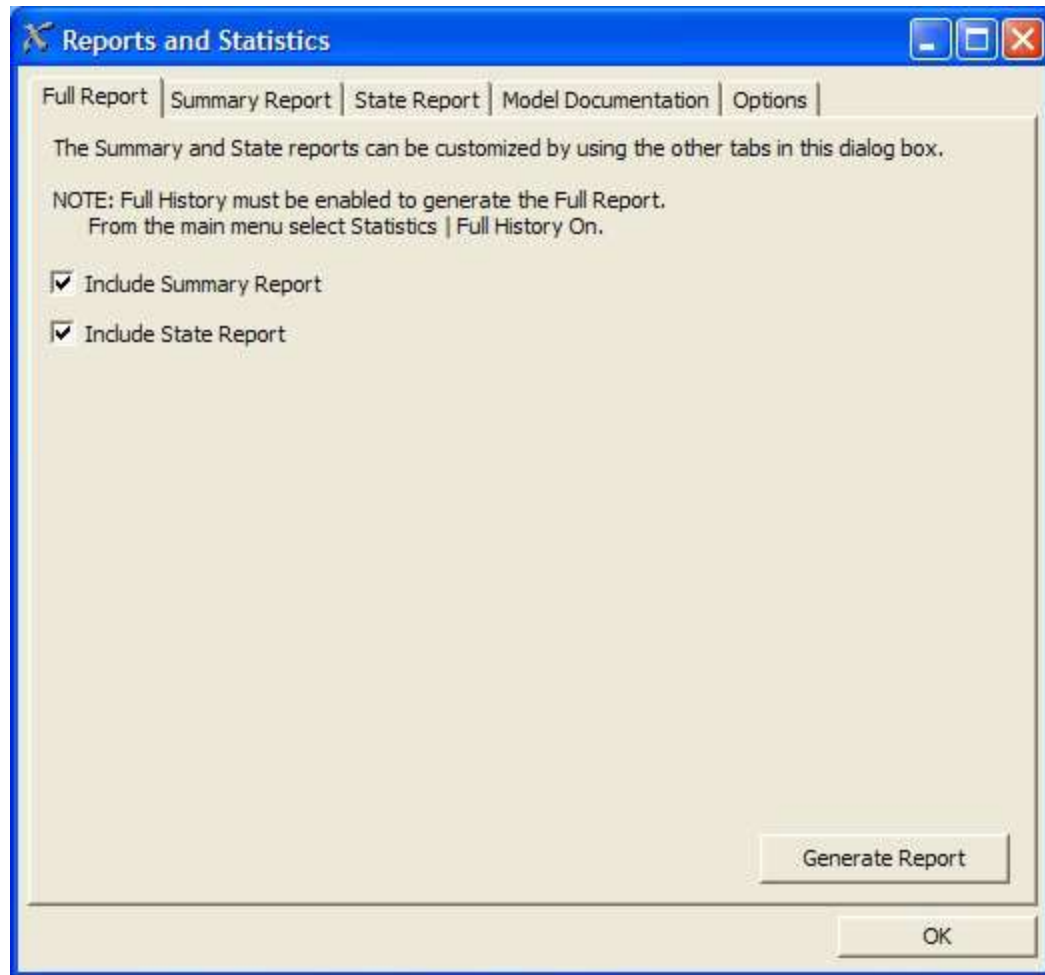
## Sample Models



The sample model dialog lets you load sample models as well as descriptions of the models. Select one of the sample models from the list, and click on the Load Sample Model button. Click on the View Model Description button to see a description of the model.

## Reports and Statistics

The Reports and Statistics dialog box allows the modeler to create various reports based on statistics gathered during a model run. These reports can include information about flowitem throughput, staytime, state history and other data that the modeler can select or customize. This dialog box also allows the modeler to create a document containing the most important details about objects in the model.



### Full Report Tab

The full report tab allows you to generate a full report of the model's run that will be saved in a database and opened with Flexsim's charting and reporting application. This database can only be generated if Full History was enabled while the model was running. If it was not enabled, then you will be shown a message informing you that no data was collected. To create the database and open it with the charting application, press the Generate Report button. The classes that appear the full report can be customized by using the Options tab.

**Include Summary Report** - If this is checked, the Summary Report will be included in the database that is created. The charting application can only display the first 32 columns of the report. If you need to see more than that, generate the report using

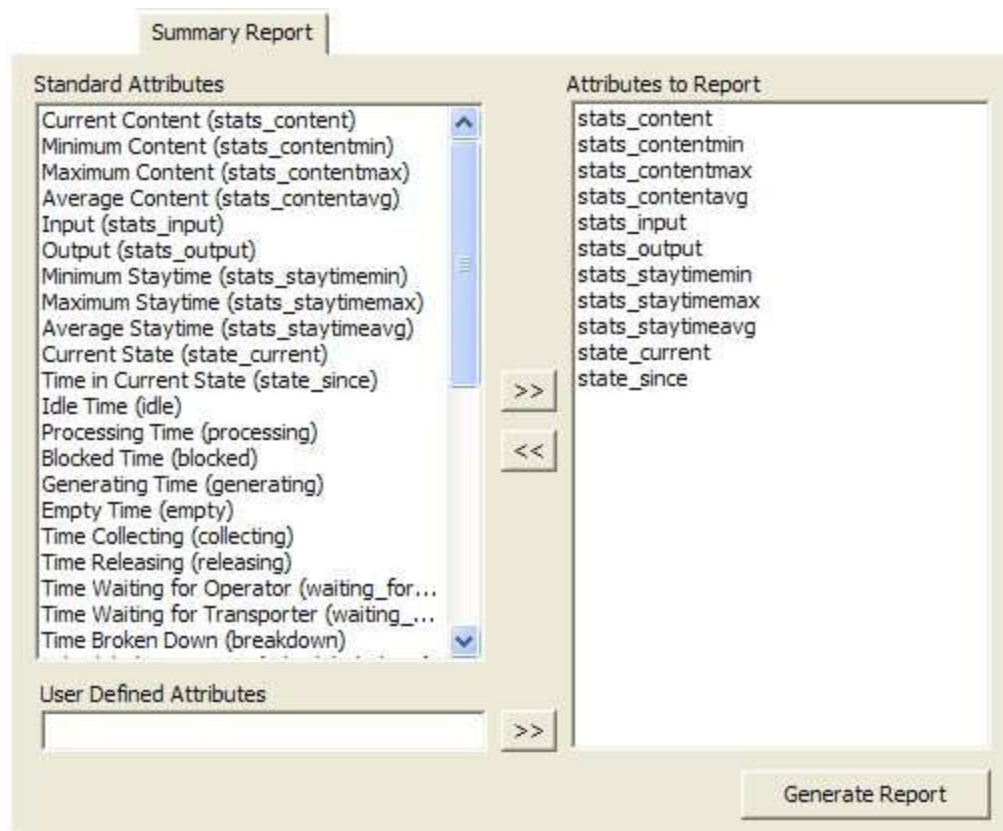
the Summary Report tab. The classes that appear the report can be customized by using the Options tab. The attributes and other values that are displayed can be changed by using the Summary Report tab.

**Include State Report** - If this is checked, the State Report will be included in the database that is created. The charting application can only display the first 32 columns of the report. If you need to see more than that, generate the report using the State Report tab. Generally you will want fewer states in the report. In that case, you can remove or change them using the State Report tab.



**Generate Report** - Press this button to generate the report.


### Summary Report Tab

The standard report tab allows you to create a report of your model. Flexsim reports on a list of attributes that you define for the model. Once you have created the list of attributes that you want to report, click on the Generate Report button. The report is then created and exported to Microsoft Excel.



**Standard Attributes** - This list lets you select standard attributes like content, staytime, state variable, etc. Press the **>>** button at the top to add the selected attribute to the list of attributes to report.

**User Defined Attributes** - Here you can type in the name of a label or variable that you want reported, then click the  button at the bottom to add your own attributes to the report list. For example, if one or more Queues have a label called "lastreditem" and you want a report of all such labels and their values, then type "lastreditem" in the field and press the  button.

**Attributes to Report** - This is the list of attributes that will be reported. To remove an item from the list, select it and press the  button.



**Generate Report** - Press this button to generate the report.



## State Report Tab

The state report tab allows you to create a report of the time the objects in your model spent in individual states. This can be reported either as an exact time or as a percentage of the model run time. Once you have created the list of states that you want to report, click on the Generate Report button. The report is then created and exported to Microsoft Excel.



**Display values as percentages** - If this box is checked the report will display the percentage of the total run time that the objects spent in each state. If it is not checked, the report will display the exact amount of time the objects spent in each state.

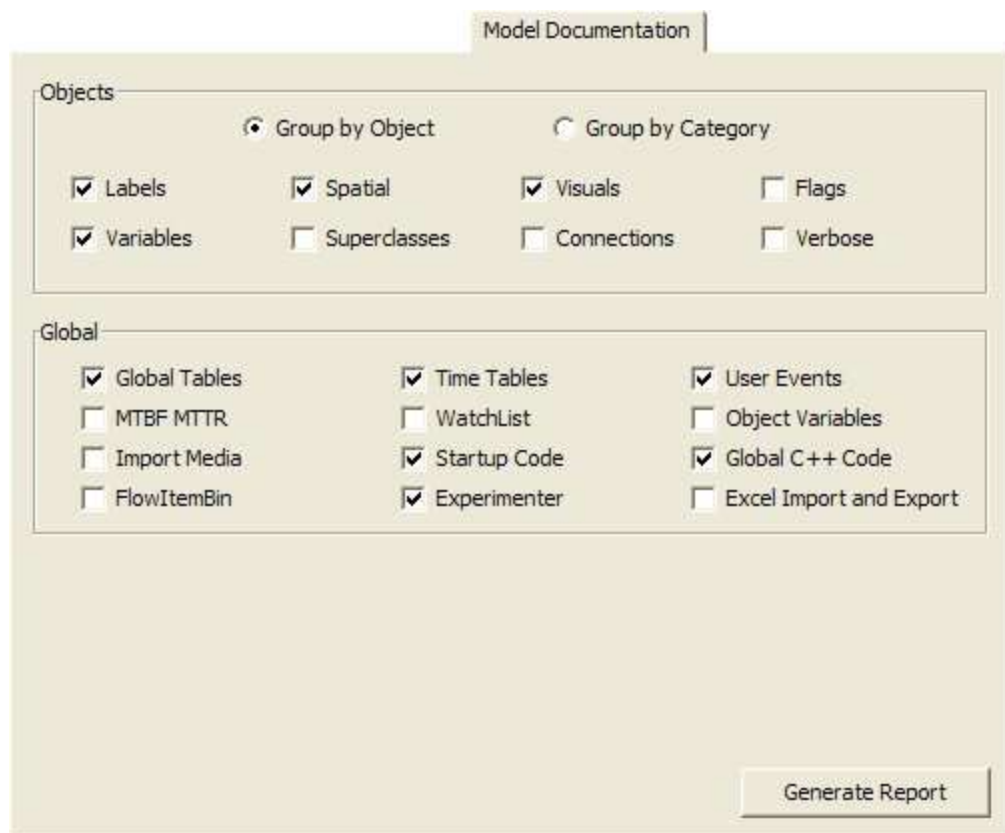
**Available States** - These are the states available in Flexsim that can be included in the report. To place a state in the States to Report column, highlight the state you are interested in and press the  button. You can place all of the available states in the States to Report column by pressing the  button.

**States to Report** - The time the objects spent in these states will be displayed in the report. The time for all of these states will be reported for all objects, even if the object was never in some of the states. To remove a state from this list, highlight it and press the  button. To remove all of the states from this list press the  button.

**Generate Report** - Press this button to generate the report.

### Model Documentation Tab

The model documentation tab lets you create a (.txt) document that reports information on your model. Check the appropriate boxes that you want to be documented, and then click the Generate Report button to create the file.



Model Documentation

Objects

☒ Group by Object ☐ Group by Category

☒ Labels ☒ Spatial ☒ Visuals ☐ Flags

☒ Variables ☐ Superclasses ☐ Connections ☐ Verbose

Global

☒ Global Tables ☒ Time Tables ☒ User Events

☐ MTBF MTTR ☐ WatchList ☐ Object Variables

☐ Import Media ☒ Startup Code ☒ Global C++ Code

☐ FlowItemBin ☒ Experimenter ☐ Excel Import and Export

Generate Report

**Objects** - These options allow the modeler to select which attributes of the model objects should be included in the report. If "Group by Object" is selected, all of the selected attributes for each object will be together in the final report. If "Group by Category" is selected, then all of the values for each attribute will be together in the report. If "Verbose" is selected, then any code fields (triggers, process time, etc) will

be documented with the full text of the field. If "Verbose" is not checked, then only the text that appears a template window will be recorded in the resulting document.

**Global** - These options allow the modeler to select which global objects and features they would like included in the report.

**Generate Report** - Press this button to generate the report.

## Options Tab

The options tab allows you to select the classes of objects that will be displayed in the reports.



**Available Classes** - These are all of the classes in Flexsim whose instances can be included in reports. These are the classes that appear the Library Icon Grid. You can add a class to the Classes to Report list by highlighting the class you are interested in and pressing the **>>** button. You can add all of the classes to the Classes to Report list by pressing the **>>>** button.

**Classes to Report** - All of the objects in the model that are instances of any of these classes will be included in any reports that are generated. Any instances of classes that are not in this list will not be included in reports. To remove a class from this list, highlight the class you want to remove and press the **<<** button. You can remove all of the classes from the list by pressing the **<<<** button.



## Trace Debugger

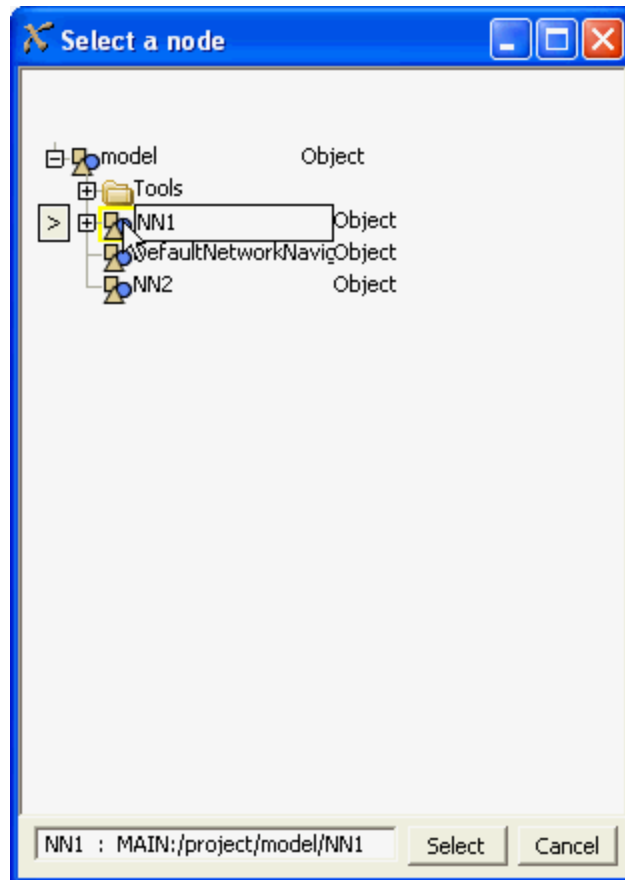
The trace debugger allows you to get information on when certain events happen, and at what times. Enter the start and end times for tracing, and check the boxes for events you want trace. OnTimerEvent happens when a created event of an object is executed. OnReceive happens when an object receives a flowitem. OnSend happens when an object sends a flowitem downstream. OnMessage happens when a message is sent to an object.



## Tree Browse Dialog

This dialog allows you to browse for a node in the tree. The purpose of selecting a node is dependent on the context of the situation. Sometimes you will use this window to select a start node for a find/replace operation. Sometimes you will use this window to select an experiment variable in the Experimenter.

Select the node you want by clicking on it in the tree view, then click on the Select button. The window will then close and return to the original calling window.



## Database Table View

This table view is accessed from the View menu. It shows the currently active database table. Database tables are opened using the `dbopen()` command. For more information, refer to the command summary.

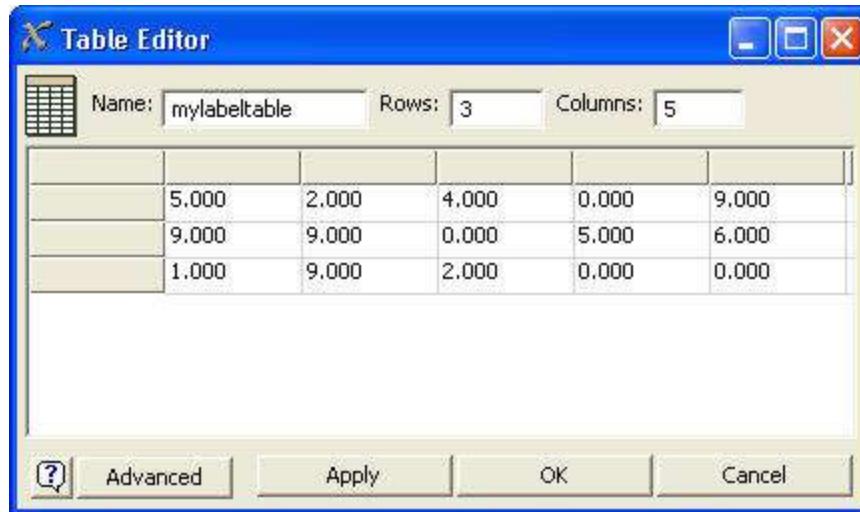


The screenshot shows a window titled "Current Database Table" with a blue title bar. Inside the window is a table with 6 columns labeled Col1 through Col6. The table contains 5 rows of data. Below the table is a large empty space. At the bottom of the window, there is a "Refresh" button and an "OK" button. A help icon (?) is also visible next to the "Refresh" button.

	Col1	Col2	Col3	Col4	Col5	Col6
	1.00	2.00	3.00	4.00	5.00	6.00
	0.00	9.00	8.00	7.00	6.00	5.00
	12.00	23.00	34.00	45.00	56.00	67.00
	0.00	0.00	0.00	0.00	0.00	0.00

## Table Editor

The table editor is used to edit a simple table in Flexsim.



**Name** - This is the table's name. It should be memorable and describe the table's function. The commands to read and write to them access them by name.

**Rows** - This is the number of rows in the table. After changing it, press "Apply" to update the table on-screen. Any new rows that were created can now be edited.

**Columns** - This is the number of columns in the table. After changing it, press "Apply" to update the table on-screen. Any new columns that were created can now be edited.

**Advanced** - This option brings up a table configurator window, allowing you to customize the tables for specific uses.

### Editing the Table

To edit a cell in the table, click on the cell and type the data in the cell. Press the arrow keys to navigate between cells. Cells hold numbers by default, but can be set to hold string data by right-clicking on the cell and selecting "Insert>Add String Data."

# Object Windows

## Object Parameters and Properties Windows

In Flexsim there are two types of windows for configuring object attributes. These two windows are referred to as the Properties and Parameters windows.

### Object Properties Window

The properties window allows you to configure attributes that are common across all objects in the library. Hence the properties window looks the same for all objects. Examples of attributes common to all objects are color, 3D shape, location, rotation, size, labels, port connections, and statistics. Each properties window has four tab pages in it. They are listed as follows.

General Properties Page  
Visual Properties Page  
Labels Property Page  
Statistics Properties Page

To bring up an object's properties page, right click on the object and select the "Properties" option from the pop-up menu.

### Object Parameters Window

The parameters window allows you to configure attributes that are dependent on the type of object you are editing. For example, a Queue has a maximum content attribute, whereas a Combiner has a component list for receiving flowitems. The parameters window of the Queue is therefore different from the parameters window of the Combiner. However, there are often commonalities between parameters windows. For example, both the Queue and the Combiner can have a send-to strategy. Objects that have commonalities will usually contain portions of their parameters windows that are the same so that you can more quickly learn how to use these windows.

To bring up an object's properties page, double click on the object or right click on the object and select the "Parameters" option from the pop-up menu.

Parameters windows are made up of a set of tab pages, depending on the object. Many pages are shared by several objects. This reference provides documentation for each parameters page, as well as links to each object that uses the page. The parameters pages are listed as follows.

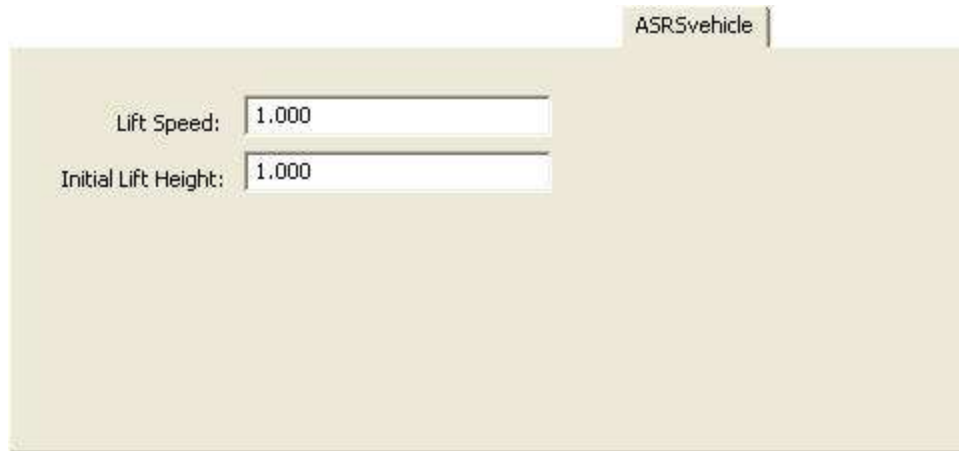
ASRSvehicle Page  
BasicFR Advanced Page  
BasicTE Page  
Collision Page  
Combiner Page  
Conveyor Page  
ConveyorLayout Page  
Crane Page  
Dispatcher Page

## Flexsim User Guide

Elevator Page  
FixedResource Page  
Flow Page  
FlowNode Page  
MergeSort Page  
MultiProcessor Page  
NetworkNode Connection Page  
NetworkNode Main Page  
Object Parameters and Properties Windows  
Object Trigger Functions  
Operators Page  
Photo Eyes  
Processor Page  
ProcessTimes Page  
Queue Page  
Rack Page  
Rack Size Table  
Recorder Parameters Wizard  
Reservoir Page  
Robot Page  
Separator Page  
Source Page  
TaskExecutor Page  
Traffic Control Page  
Transporter Page  
VisualTool Page

## Parameters Pages

### ASRSvehicle Tab Page



The screenshot shows a software window titled "ASRSvehicle". Inside the window, there are two input fields. The first is labeled "Lift Speed:" and contains the value "1.000". The second is labeled "Initial Lift Height:" and also contains the value "1.000". The background of the window is a light beige color.

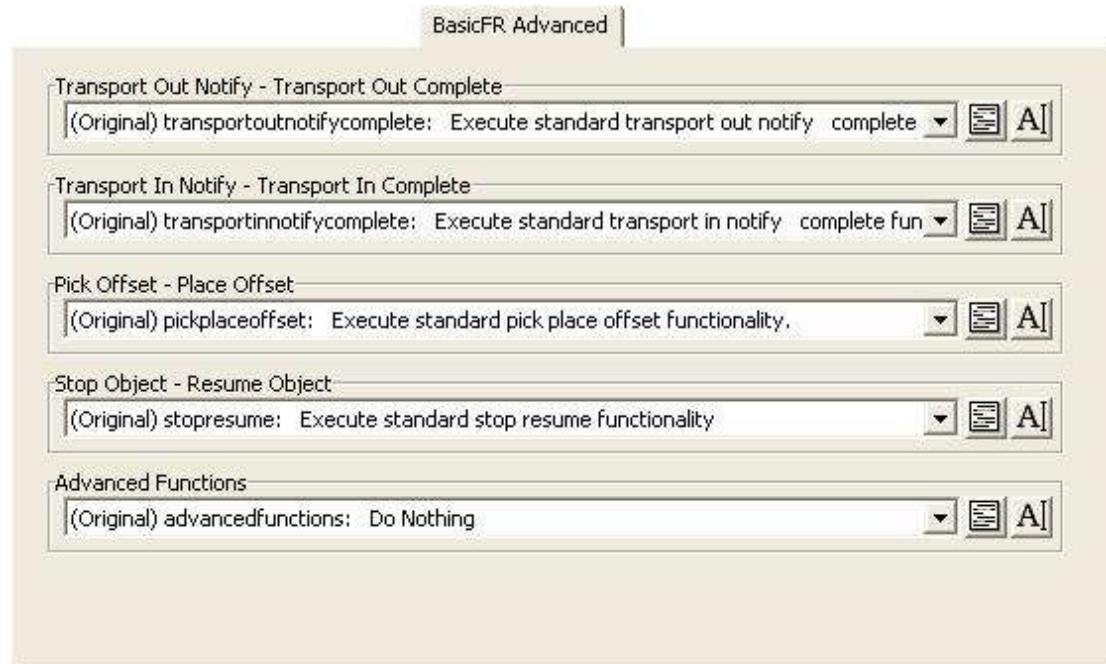
**Lift speed** – This number is how fast the lift on the ASRSvehicle moves up and down.

**Initial lift height** – This number defines the reset z location of the ASRSvehicle's platform. The platform returns to this height when the model resets. It also returns to this height when it is not traveling the offset of a load/unload task.

### This Page is Used By

ASRSvehicle

## BasicFR Advanced Page



**Transport Out Notify - Transport Out Complete** - This pick list is fired at two different times. First is when the object is notified that a flowitem is going to exit the object using a transport. This function is referred to as transport out notify. The second time it is fired is when the transport has arrived, finished its load time, and is about to move the flowitem out. This is called transport out complete. A variable is passed into this function telling you which operation is applicable. In this field you can manage things like the `nrofransportsout` variable, as well as how to screen further output/input to the object.

Access variables for the transport out notify - transport out complete function are as follows:

**current:** the current object

**notifyoperation:** this variable is 1 or 0. 1 means it is a transport out notify operation, 0 means it is a transport out complete operation.

**item:** this is a reference to the item that is going to leave this object

**port:** this is the output port number through which the item will exit

**transporter:** For a transport out complete operation, this is a reference to the transporter that is picking the item up.

**nrofransportsoutnode:** This is a reference to the object's `nrofransportsout` variable. The value of this node should be incremented by 1 at a notify operation, and decremented by 1 at a complete operation. You can also query the value of this node to know how many items are waiting for a transport to pick them up.

**Transport In Notify - Transport In Complete** - This pick list is fired at two different times. First is when the object is notified that a flowitem is going to enter the object using a



transport. This function is referred to as transport in notify. The second time it is fired is when the transport has arrived, finished its unload time, and is about to move the flowitem into the object. This is called transport in complete. A variable is passed into this function telling you which operation is applicable. In this field you can manage things like the `nroftansportsin` variable, as well as how to screen further output/input to the object. For example, you may want to allow more than one flowitem to be in transit to the object at the same time. In such a case, you could call `receiveitem` when you are notified of an upcoming entry within this field.

Access variables for the transport in notify - transport in complete function are as follows:

**current:** the current object

**notifyoperation:** this variable is 1 or 0. 1 means it is a transport in notify operation, 0 means it is a transport in complete operation.

**item:** this is a reference to the item that is going to enter this object

**port:** this is the input port number through which the item will enter

**transporter:** For a transport in complete operation, this is a reference to the transporter that is dropping the item off.

**nroftansportsinnode:** This is a reference to the object's `nroftansportsin` variable. The value of this node should be incremented by 1 at a notify operation, and decremented by 1 at a complete operation. You can also query the value of this node to know how many items are still in transit to this object.

**Note on the transport out/in complete function return value:** In the notify and complete operations of both transport in and transport out, if the function returns a value of 0, the object will assume that nothing was done and will execute its own default logic. If this function returns a 1, the object will assume the proper variable management was done and will do nothing. If this function returns a -1 and the operation is a complete operation, the object will again assume proper variable management, but in addition, it will notify the transporter that it is not ready to receive the flowitem. The transporter then must wait until it is notified that it can resume its operation. The reason you may need to use this is in case this object has been stopped using `stopobject()`. If so, you may not want any flowitems coming in or going out. If this is the case, then you will need to save off a reference to the transporter using the `savestoppedtransportin()` or `savestoppedtransportout()` function, and then return -1. Then, when it is ok for the transporter to resume its operation (usually from this object's `resumeobject` function) you will need to call `resumetransportsin()` and `resumetransportsout()` to notify all stopped transports that they may resume their operation.

**Pick Offset - Place Offset** - This pick list is fired at two different times. First is when a transport object is attempting to place, or unload, a flowitem into this object. This is called place offset. It should return an offset location for the transport to offset to before placing the item. The second time this is called is when a transport is about to pick, or load, an item from this object. This is called pick offset. It should again return an offset location for the transport to offset to before picking the product up.

Access variables for the pick offset - place offset function are as follows:

**current:** the current object

**pickoperation:** this variable is 1 or 0. 1 means it is a pick operation, 0 means it is a place operation.

**item:** this is a reference to the item that is being picked or placed

**otherobject:** this is a reference to the object that is picking or placing the item

**xvalnode, yvalnode, zvalnode:** these parameters are references to nodes whose values should be set in this function, and will represent the offset location returned by this function. For example, if I want the picking/placing object to offset 10 in the x direction, 0 in the y direction, and 5 in the z direction, I would set xvalnode to 10 using the setnodenum() command, yvalnode to 0, and zvalnode to 5.

**Note on the pick/place offset return value:** If you are implementing your own pick/place logic here, and do not want the default logic to be executed, you must return 1 from this function. If the object gets a return value of 0 from this function, it will assume that nothing was done, and will execute its own default logic.

**Stop Object - Resume Object** - This pick list is fired at two different times. First is when the stopobject() command is called for this object. Second is when the resumeobject command is called for this object. This field should define a strategy for how the object will "stop", and how it will "resume". This field should also manage data for remembering how many stop requests have been made, what the state of the object was before it was stopped, etc.

Access variables for the stop object - resume object function are as follows:

**current:** the current object

**stopoperation:** this value is either a 1 or 0. 1 means the stopobject() command is being called on this object. 0 means the resumeobject() command is being called on this object.

**stopstate:** this value is only applicable for a stop operation. It specifies the requested state passed into the stopobject() command.

**nrofstopnode:** this is a reference to this object's nrofstops variable. If this is a stop operation, the value of this node should be incremented by 1 within this function. If it is a resume operation, the value of this node should be decremented by 1 within this function. Also, you will get the value of this node to know how many stopobject() commands have been called on this object. When the object is stopped for the first time (the value of the node goes from 0 to 1), you should execute logic specifying how to stop this object. When the object is resumed for the final time (the value of the node goes from 1 to 0), you should execute logic specifying how to resume this object.

**timeoflaststopnode:** this is a reference to this object's timeoflaststop variable. When this object is stopped for the first time, this node should be set to the current time, so you can know the amount of time the object was stopped when it is finally resumed.

**statebeforestopnode:** this is a reference to this object's statebeforestopped variable. When this object is stopped for the first time, this node should be set to the object's current state, so that you can know which state to go back to when the object is resumed.

**Note on the stop/resume object return value:** If you are implementing your own stop/resume logic here, and do not want the default logic to be executed, you must return 1 from this function. If the object gets a return value of 0 from this function, it will assume that nothing was done, and will execute its own default logic.

**Advanced Functions** - This pick list is fired for several different notifications or functions that are called on the object. For the most part, you will not need to implement any logic for these notifications, but they are nonetheless made accessible to you. The return value of this function should either be a 1 or a 0. If 0, the object will execute the default functionality associated with the given notification. If 1, the object not do any default functionality, but assumes that the function has overridden the default.

The type of notification called for the advanced function is passed in as parval(1), or msgtype. This parameter can be one of several values. These values are listed as follows:

**ADV\_FUNC\_CLASSTYPE:** This is a request to get the type of class of the object. The classtype should be returned as an integer with certain bits set high. You can construct this value using the bitwise OR operator: | and several classtype macros. For example, a FixedResource's classtype is: `CLASSTYPE_FLEXSIMOBJECT | CLASSTYPE_FIXEDRESOURCE`.

**ADV\_FUNC\_DRAGCONNECTION:** This function is called when a keyboard key is held down, and the user clicks and drags from this object to another object. In this case, the object to which the mouse was dragged is passed in as parnode(2), the ascii value of the key that was clicked is passed in as parval(3), and the classtype value of the object is passed in as parval(4).

**ADV\_FUNC\_CLICK:** This function is called when the object is clicked on. Here a reference to the view in which it was clicked is passed in as parnode(2), and the click code is passed in as parval(3). Possible click codes are: `DOUBLE_CLICK`, `LEFT_PRESS`, `LEFT_RELEASE`, `RIGHT_PRESS`, `RIGHT_RELEASE`.

**ADV\_FUNC\_KEYEDCLICK:** This function is called when a key on the keyboard is held down and the object is clicked on. Here the view is passing as parnode(2), the click code is passed in as parval(3), and the ascii value of the pressed key is passed in as parval(4). Possible click codes are: `DOUBLE_CLICK`, `LEFT_PRESS`, `LEFT_RELEASE`, `RIGHT_PRESS`, `RIGHT_RELEASE`.

### This Page is Used By

BasicFR

## BasicTE Page

The screenshot shows the 'BasicTE' configuration window with the following settings:

- OnBeginOffset:** (Original) onbeginoffset: Do Nothing
- OnUpdateOffset:** (Original) onupdateoffset: Do Nothing
- OnFinishOffset:** (Original) onfinishoffset: Do Nothing
- Pick Offset - Place Offset:** (Original) pickplaceoffset: Execute standard pick.place offset functionality.
- Stop Object - Resume Object:** (Original) stopresume: Execute standard stop resume functionality
- Advanced Functions:** (Original) advancedfunctions: Do Nothing

**Note:** For more information on offset travel, refer to the TaskExecuter's documentation on offset travel.

**OnBeginOffset** - This pick list is fired at the beginning of an offset travel operation. An x,y,z offset location is passed into this function. From this x,y,z offset location, the object should figure out how it will travel the specified offset, and then return the amount of time it will take to make the travel operation.

Access variables for the OnBeginOffset function are as follows:

- current:** the current object
- x:** the requested x offset. This is an offset distance from the x center of the object.
- y:** the requested y offset. This is an offset distance from the y center of the object.
- z:** the requested z offset. This is an offset distance from the z base of the object.
- item:** if the offset operation has an involved item, this is a reference to the item
- endspeed:** this is the requested end speed for the offset travel operation
- maxspeed:** this is the value of this object's maximum speed variable
- acceleration:** this is the value of this object's acceleration variable
- deceleration:** this is the value of this object's deceleration variable
- lastupdatespeed:** this is the value of this object's lastupdatespeed variable. It is the end speed of the object's last travel operation.
- rotatewhiletraveling:** this is the value of this object's rotatewhiletraveling

variable. It is a 1 or 0, and specifies whether the user wants the object to rotate while traveling.

**OnUpdateOffset** - This pick list is fired before the view is refreshed. Here is where the object updates its location based on its current offset operation.

**OnFinishOffset** - This pick list is fired when the object has finished its offset operation. This should update the location of the object to its final offset location.

Access variables for the OnUpdateOffset function and the OnFinishOffset function are defined as follows:

**current:** the current object

**offsettingnow:** this is the value of this object offsettingnow variable. If the object is currently doing an offset operation, this will be 1, otherwise 0. The offsettingnow value is automatically set to 1 when OnBeginOffset is called, and then set back to 0 when OnFinishOffset is called.

**offsettingtotaltime:** this is the value of this object offsettotaltime variable. It tells the total time this object returned from its OnBeginOffset function.

**maxspeed:** this is the value of this object's maximum speed variable

**acceleration:** this is the value of this object's acceleration variable

**deceleration:** this is the value of this object's deceleration variable

**lastupdatedspeed:** this is the value of this object's lastupdatedspeed variable. It is the end speed of the object's last travel operation.

**rotatewhiletraveling:** this is the value of this object's rotatewhiletraveling variable. It is a 1 or 0, and specifies whether the user wants the object to rotate while traveling.

**curloadunloadtime:** if the offset operation is a load or unload operation, then this value is the load/unload time for the operation. Note that if there is a non-zero load/unload time, then this time will be executed before the OnFinishOffset trigger is fired. This means that in your update offset function, you may need to query whether you're in the offset part of the operation, or in the load/unload part of the operation. Usually this will not matter, though, because the travel operations will finish, and during the remaining load/unload time, the update function will automatically set the object's location to the final destination location. You will really only need to use this if you are going to do some animation/movement during the load/unload time, as the ASRSvehicle and Elevator objects do.

**Pick Offset - Place Offset** - This pick list is fired at two different times. First is when another transport object is attempting to place, or unload, a flowitem into this object. This is called place offset. It should return an offset location for the transport to offset to before placing the item. The second time this is called is when a transport is about to pick, or load, an item from this object. This is called pick offset. It should again return an offset location for the transport to offset to before picking the product up. Note that this function does not fire when this object is attempting to load/unload to or from another object, but rather when another object is attempting to load/unload to or from this object.

Access variables for the pick offset - place offset function are as follows:

**current:** the current object

**pickoperation:** this variable is 1 or 0. 1 means it is a pick operation, 0 means it is a place operation.

**item:** this is a reference to the item that is being picked or placed

**otherobject:** this is a reference to the object that is picking or placing the item

**xvalnode, yvalnode, zvalnode:** these parameters are references to nodes whose values should be set in this function, and will represent the offset location returned by this function. For example, if I want the picking/placing object to offset 10 in the x direction, 0 in the y direction, and 5 in the z direction, I would set xvalnode to 10 using the setnodenum() command, yvalnode to 0, and zvalnode to 5.

**Note on the pick/place offset return value:** If you are implementing your own pick/place logic here, and do not want the default logic to be executed, you must return 1 from this function. If the object gets a return value of 0 from this function, it will assume that nothing was done, and will execute its own default logic.

**Stop Object - Resume Object** - This pick list is fired at two different times. First is when the stopobject() command is called for this object. Second is when the resumeobject command is called for this object. This field should define a strategy for how the object will "stop", and how it will "resume". This field should also manage data for remembering how many stop requests have been made, what the state of the object was before it was stopped, etc.

Access variables for the stop object - resume object function are as follows:

**current:** the current object

**stopoperation:** this value is either a 1 or 0. 1 means the stopobject() command is being called on this object. 0 means the resumeobject() command is being called on this object.

**stopstate:** this value is only applicable for a stop operation. It specifies the requested state passed into the stopobject() command.

**nrofstopnode:** this is a reference to this object's nrofstops variable. If this is a stop operation, the value of this node should be incremented by 1 within this function. If it is a resume operation, the value of this node should be decremented by 1 within this function. Also, you will get the value of this node to know how many stopobject() commands have been called on this object. When the object is stopped for the first time (the value of the node goes from 0 to 1), you should execute logic specifying how to stop this object. When the object is resumed for the final time (the value of the node goes from 1 to 0), you should execute logic specifying how to resume this object.

**timeoflaststopnode:** this is a reference to this object's timeoflaststop variable. When this object is stopped for the first time, this node should be set to the current time, so you can know the amount of time the object was stopped when it is finally resumed.

**statebeforestopnode:** this is a reference to this object's statebeforestopped variable. When this object is stopped for the first time, this node should be set to the object's current state, so that you can know which state to go back to when the object is resumed.

**Note on the stop/resume object return value:** If you are implementing your own stop/resume logic here, and do not want the default logic to be executed, you must return 1 from this function. If the object gets a return value of 0 from this function, it will assume that nothing was done, and will execute its own default logic.

**Advanced Functions** - This pick list is fired for several different notifications or functions that are called on the object. For the most part, you will not need to implement any logic for these notifications, but they are nonetheless made accessible to you. The return value of this function should either be a 1 or a 0. If 0, the object will execute the default functionality associated with the given notification. If 1, the object not do any default functionality, but assumes that the function has overridden the default.

The type of notification called for the advanced function is passed in as parval(1), or msgtype. This parameter can be one of several values. These values are listed as follows:

**ADV\_FUNC\_CLASSTYPE:** This is a request to get the type of class of the object. The classtype should be returned as an integer with certain bits set high. You can construct this value using the bitwise OR operator: | and several classtype macros. For example, a FixedResource's classtype is: `CLASSTYPE_FLEXSIMOBJECT | CLASSTYPE_FIXEDRESOURCE`.

**ADV\_FUNC\_DRAGCONNECTION:** This function is called when a keyboard key is held down, and the user clicks and drags from this object to another object. In this case, the object to which the mouse was dragged is passed in as parnode(2), the ascii value of the key that was clicked is passed in as parval(3), and the classtype value of the object is passed in as parval(4).

**ADV\_FUNC\_CLICK:** This function is called when the object is clicked on. Here a reference to the view in which it was clicked is passed in as parnode(2), and the click code is passed in as parval(3). Possible click codes are: `DOUBLE_CLICK`, `LEFT_PRESS`, `LEFT_RELEASE`, `RIGHT_PRESS`, `RIGHT_RELEASE`.

**ADV\_FUNC\_KEYEDCLICK:** This function is called when a key on the keyboard is held down and the object is clicked on. Here the view is passing as parnode(2), the click code is passed in as parval(3), and the ascii value of the pressed key is passed in as parval(4). Possible click codes are: `DOUBLE_CLICK`, `LEFT_PRESS`, `LEFT_RELEASE`, `RIGHT_PRESS`, `RIGHT_RELEASE`.

**This Page is Used By**

BasicTE

## Collision Tab Page

For more information on collision detection functionality, refer to the TaskExecutor.

The screenshot shows the 'Collision' configuration window. At the top, the 'Collision' tab is selected. Below it, there's a text field for 'Time between Collision Checks' with the value '1.000'. To the right are two checkboxes: 'Checking Collisions' and 'Draw Spheres', both currently unchecked. The main area is divided into two sections. The top section, 'Collision Spheres', contains a table with three columns and two rows. To the right of the table are three buttons: 'Add Sphere', 'Delete Sphere', and 'Advanced...'. The bottom section, 'Collision Members', is split into two panes. The left pane, labeled 'Model', contains a list with one item, 'TaskExecutorFlowItem'. The right pane, labeled 'Collision Members', is empty. Between the two panes are two buttons: '>>' and '<<'. At the bottom of the window is a 'HandleCollision' dropdown menu showing '(Original) collisiontrigger: Print a message to the output console detailing the collision.' To the right of the dropdown are a help icon and the text 'Alt'.

**Time between Collision Checks** - The simulation time that passes between this object's collision checks. This does not specify the time between its collision members' collision checks.

**Draw Spheres** - Check this box if you want the collision spheres visible around the CollisionObject.

**Checking Collisions** - Check this box to turn on collision detection.

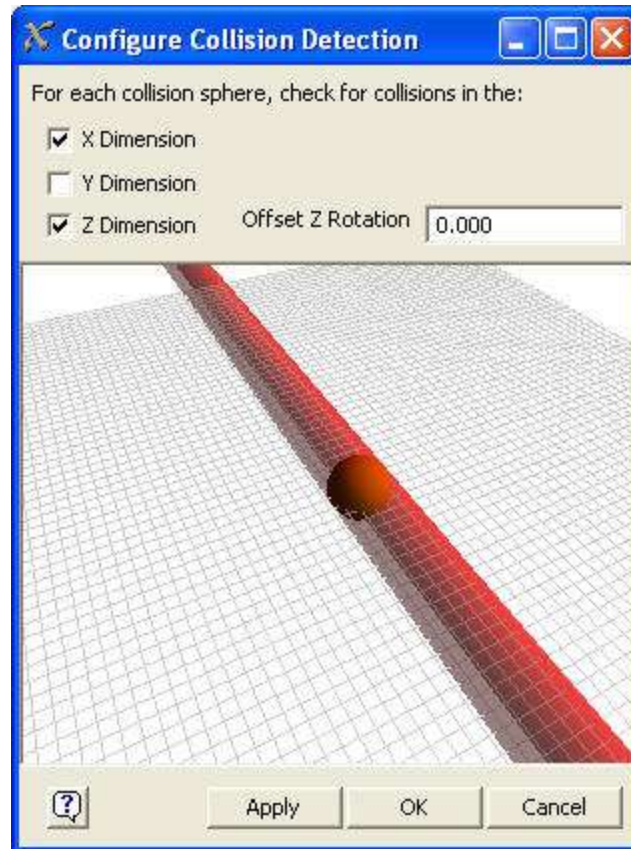
**Collision Spheres** - This table is used to define one or more collision spheres on the object.

**Add Sphere** - Select this button to add a new sphere to the object. Define the size and position of the sphere in the table.

**Delete Sphere** - Select this button to delete the last sphere in the table.



**Advanced...** - This button allows you to optimize for collision checking speed by configuring the TaskExecutor to exclude certain axes when checking its spheres for collisions. This lets you cover more checking area with less spheres. Pressing the button opens the window below. Uncheck the X, Y, and/or Z Dimensions to exclude certain axes in the check. The result for your configuration is drawn in the view. A transparent cylinder or plane covers areas that will cause a collision given the configuration you've chosen. You can also enter an offset rotation, like 45 degrees, if you want to check for collisions on an axis that is not parallel with the normal axis. Note that the X, Y, and Z dimensions are according to the global coordinate system, and not according to the individual object's coordinate system.



**Collision Members** - On the left is a list of model objects that can be added to the TaskExecutor's collision members. On the right is the list of collision members for the object. To add a member from the model list to the TaskExecutor's member list, select the object you want by clicking on it, and then click on the ">>" button. To delete an object from the member list, select the one you want, and click on the "<<" button.

**HandleCollisions** - The pick list allows the user to define what happens when a collision takes place. See Collision Trigger pick list.

### This Page is Used By

TaskExecutor  
Transporter

## Flexsim User Guide

Operator  
Crane  
ASRSvehicle  
Robot  
Elevator

---

## Combiner Tab Page

	Target Quantity
From Input Port 2	1.000
From Input Port 3	1.000
From Input Port 4	1.000

**Pack/Join/Batch** - Selects the mode that the Combiner is operating in.

**Convey Items Across Combiner Length** - If checked, flowitems will travel across the Combiner during their process time.

**Components list** - This table is used to define how many of each type of flowitem the combiner will collect before sending the completed collection downstream. The combiner will use the flowitem that arrives through input port one as the container object and will only accept one of them. Each row in the table represents arrivals from input ports numbered two and above. If you make additional connections while this window is open, you will need to close the parameters window and reopen it in order for your changes to register.

**TargetQuantity** - The required number of flowitems to be received through the associated input port for each operation.

## This Page is Used By

Combiner

## Conveyor Tab Page

The screenshot shows the 'Conveyor' tab in the Flexsim software interface. It is divided into two main sections: 'Operation' and 'Visual'.

**Operation Section:**

- ☒ Accumulating
- Speed: 1.00
- Maximum Content: 1000.00
- Spacing Value: 1.00
- Spacing Rule: Item Size (dropdown)
- Spacing Orientation: Item X Size (dropdown)
- Orient Z: 0.00
- Orient Y: 0.00
- Virtual Length: 0.00
- ☐ Scale Product Size with Virtual Length
- ☐ Notify Upstream of Blocked Length

**Visual Section:**

- Texture: fs3d\RollerConveyor.bmp (with a browse button)
- Texture Length: 2.00
- Product Z Offset: 0.00
- ☒ Side skirt follows contour of conveyor (not floor)
- Side skirt dimension: 0.20
- ☐ Leg base relative to conveyor (not floor)
- Leg base dimension: 0.00

### Operation

These parameters define how the conveyor functions.

**Accumulating** - If this box is not checked, the conveyor is non-accumulating. In a non-accumulating conveyor, the entire conveyor stops if a flowitem reaches the end and cannot exit. If accumulating, the flowitems will continue traveling the length of the conveyor until they run into a stopped flowitem or reach the end. The distance between accumulated flowitems is determined by the spacing rule.

**Speed** - This number defines the speed that flowitems travel at as they move down the conveyor.

**Maximum Content** - This number sets a limit on how many flowitems can be on the conveyor at one time. The maximum number of flowitems on a conveyor is usually determined by the length of the conveyor and the size of the flowitems, but this allows you to screen it even more.

**Spacing Value** - This number is used in different ways, as you define in the Spacing Rule.

**Spacing Rule** - This defines how much space the conveyor leaves between flowitems that are on it.

**Item Length** - A flowitem will stop moving when it reaches the flowitem in front of it.

**Item Length + Spacing Value** - A flowitem will stop when it is a certain distance from the back of the flowitem in front of it. That distance is defined by the spacing value.

**Spacing Value** - A flowitem will stop when its front edge is a certain distance away from the front edge of the flowitem in front of it. That distance is defined by the spacing value.

**Spacing Orientation** - If the Spacing Rule includes the item's size, this field determines which dimension of an item defines its size on the conveyor. Options are x,y, or z size.

**Orient Z, Orient Y** - These fields are for visual purposes and do not affect the operation of the conveyor. They are used in conjunction with the Spacing Orientation field to orient the flow items correctly. They define a rotation in degrees around the z and y axes of the conveyor for flow items to be rotated. For example, if in the Spacing Orientation field Item Y Size is chosen instead of the default Item X Size, you will want to specify the Orient Z field as either 90 or 270, so that the y dimension of the flow item aligns with the length of the conveyor, instead of the x size. If Item Z Size is chosen, then you will want the Orient Y field to be either 90 or 270.

**Virtual Length** - This field allows you to define a specific length for the conveyor, different from the actual length of its layout. If 0 is input (default), the conveyor's normal layout length will be used. Otherwise, the value in this field will be used as the conveyor's length. This field is used if you want to simulation a very long conveyor without having it traverse a huge distance in your model, or if you want to specify an exact distance.

**Scale Product with User Length** - This is for visual purposes and does not affect the operation of the conveyor. If you have defined a very large Virtual Length, this field allows you to scale the size of the products that flow down the conveyor so that they don't overlap. For example, if the conveyor's layout length is 10, but you've specified a virtual length of 100, then items with a size of 1 would only take up 1/100th of the conveyor's virtual length. This is 0.1 units in the real layout, but since a flow item is 1 unit, it overlaps with other flow items. If this box is checked, then the size of the flow item will be shrunk to 0.1 units.

**Notify Upstream of Blocked Length** - This tells the conveyor to notify upstream conveyors when products on the conveyor extend past the conveyor's leading edge.

## Visual

These parameters define how the conveyor looks in the model. They do not affect the conveyor's operational functionality. These parameters in combination give you unlimited flexibility in the look of your conveyor. Often configuring these settings is done by trial and error, but you should be able to learn quickly how each parameter affects the conveyor's look.

**Texture** - This file will be drawn on the conveyor to change its basic appearance. The texture is divided into three parts. The left third of the texture is drawn onto the left side skirt, the middle third is drawn on the top, and the right third is drawn on the right side skirt.

**Texture Length** - This number defines the length along the plane of the conveyor to stretch one copy of the texture before repeating the texture.

**Product Z Offset** - Changing this number will change how far above or below the conveyor the flowitems are drawn. A value of 0 puts the items directly on the conveyor. A negative value will put the items below the conveyor's plane.

**Side skirt follows contour of conveyor (not floor)** - If this box is checked, the side of the conveyor will be drawn starting a certain distance away from the conveyor. If it is not checked, it is drawn starting a certain distance away from the floor. Either way, the sides connect to the conveyor's contour.

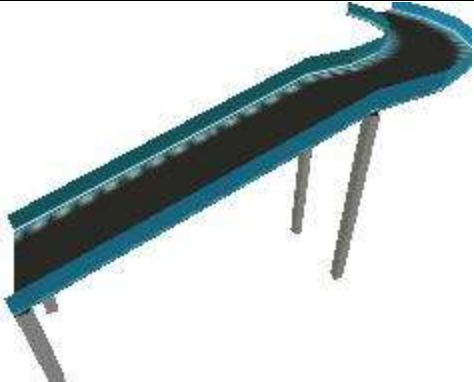
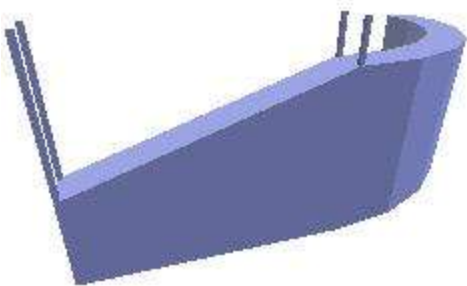
**Side Skirt Dimension** - This value is used to determine how long the skirt is.

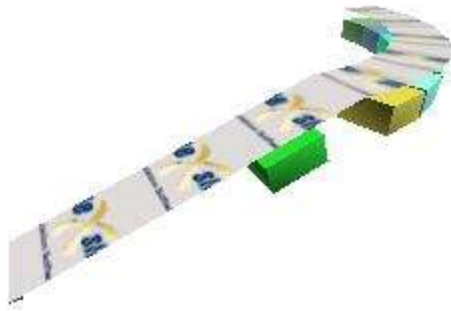
**Leg base relative to conveyor** - If this box is checked, the legs of the conveyor will be drawn starting a certain distance away from the conveyor. If it is not checked, they are drawn starting a certain distance away from the floor. Either way, the sides connect to the conveyor's contour.

**Leg base dimension** - This value is used to determine where the legs of the conveyor begin.

## Visual Examples

Here are some examples of different looks you can give your conveyor. They might not be completely practical, but they are there to show the flexibility and give you ideas. Attached with each picture is a small explanation of what was changed from the default visualization of the conveyor.

 <p><b>Belt conveyor with side rail</b>  Side Skirt Dimension = -0.2  Texture = fs3d/BeltConveyor.bmp</p>	 <p><b>Blue hanging conveyor with big side skirt</b>  Texture = &lt;blank&gt;  Skirt follows contour unchecked  Skirt dimension = 0  Leg base dimension = 2.5  color changed to blue</p>
--	--

**Hanging tape conveyor with Flexsim texture**

Texture = flexsim.bmp  
Product Z Offset = -0.5  
Side Skirt Dimension = 0  
Leg base relative checked  
Leg base dimension = 0  
Texture Length = 1

**Conveyor with only side skirts**

Custom texture: myflexsim.jpg with  
myflexsim.tpg in same directory  
Product Z Offset = -0.5

**This Page is Used By**

Conveyor  
MergeSort

## Layout Tab Page

Layout

Conveyor Section Edit Table

Initial Z Rotation:  Number of Sections:

type: 1=straight, 2=curved

	type	length	rise	angle	radius	nroflegs
section1	1.000	10.000	0.000	90.000	5.000	2.000
section2	2.000	10.000	0.000	90.000	3.000	2.000
section3	1.000	10.000	2.000	90.000	5.000	2.000

< >

**Initial Z Rotation** - This number sets the rotation of the conveyor, or the direction of the first section of the conveyor. This is the same as editing the Z rotation in the Properties window. Be aware that you may have problems editing this value if you have both the Parameters and Properties windows open at the same time, since hitting the apply/ok button on one of the windows may overwrite what you have changed in the other window.

**Number of Sections** - This field defines the number of sections that will be in the conveyor. Each section corresponds to one row in the table. All information about a section is stored in that section's row of the table. To change the number of sections, enter the desired number and hit the apply button. The table will then be updated accordingly.

### The Section Table

This table stores all of the information about the conveyor's sections. Each row represents one section. The conveyor is customized by editing this table. Several of the fields are computed automatically and should not be edited by the user. The fields that the user can safely edit are described here.

**Type** - This number defines what type of section this row is. The valid values are 1 for a straight section, and 2 for a curved section.

**Length** - This number defines how long the section is if it is a straight section. The value is ignored if it is a curved section.

**Rise** - This value defines the difference in height between the start of the segment and the end. For example, if this value is 3, the segment will end 3 units higher than it began.



**Angle** - This number defines the angle in degrees that the segment will turn if it is curved. It can be positive or negative and can be greater than 360. This value is ignored if the section is a straight section.

**Radius** - This number defines the radius of the turn in the curved section. The radius is measured from the center of the turn to the center of the conveyor. This value is ignored if the section is a straight section.

**NrofLegs** - This number defines the number of legs that will be drawn for this section. They will be drawn differently depending on the settings made in the Conveyor tab-page.

All other fields in the table should not be edited. They can, however, give you feedback on the sections you edit. For example, the `seclength` column shows the total length of the section, given the parameters that you have specified in the other fields. The `startlength` column shows the total length of the conveyor up to but not including that section.

### This Page is Used By

Conveyor  
MergeSort

---

## Crane Tab Page

Crane

Travel Sequence

L>XY>D

X : Move Gantry  
Y : Move Trolley  
L : Lift Hoist  
D : Drop Hoist  
> : Separate Travel Operations

Speeds

	Max_Speed	Acceleration	Deceleration
Gantry	2.00	1.00	1.00
Trolley	2.00	1.00	1.00
Hoist_Lift	2.00	1.00	1.00
Hoist_Drop	2.00	1.00	1.00

Lift Height  Lift Radius

Frame X Location  Frame X Size

Frame Y Location  Frame Y Size

Frame Z Location  Frame Z Size

**Travel Sequence** - Here you can specify the order in which the crane performs travel operations. Refer to the crane specifications for more information.

**Speed Table** - Here you specify the max speed, acceleration and deceleration for each of the 4 operations the crane will do. Note that these operations only apply to offset travel. If the crane is connected to a network, then when it is on the network, only the normal maxspeed, acceleration and deceleration specified in the TaskExecutor page will be used.

**Lift Height** - Here you define how high the crane will lift to get to its lift height.

**Lift Radius** - Specify a radius within which the crane will not do a lift operation.

**Frame X/Y/Z Location** - These numbers define the location of the crane's frame. Note that this is different than the cranes actual x/y/z location. The crane's x/y/z location describes the location of the moving part of the crane. The frame will be stationary throughout the simulation, while the actual x/y/z location of the crane changes as the crane travels.

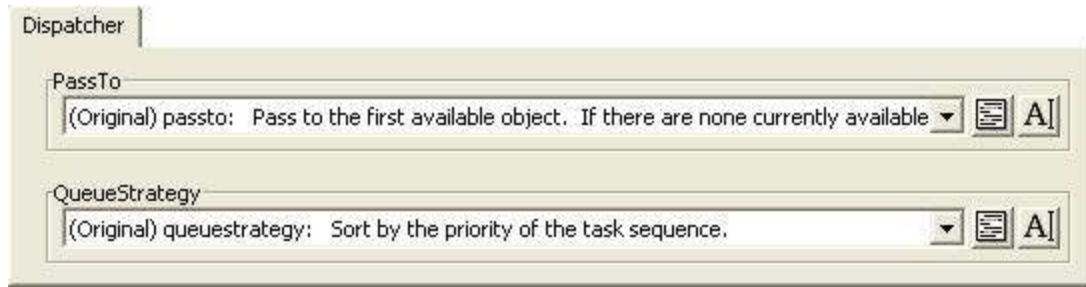
**Frame X/Y/Z Size** - These numbers define the size of the crane's frame. Note that this is different than the cranes actual x/y/z size. The crane's x/y/z size describes the size of the moving part of the crane. If you do not want a frame to be drawn at all, then specify these values as 0.

## This Page is Used By

Crane



## Dispatcher Tab Page



**Pass To** - This pick list returns the output port number that the task sequence should be dispatched to. If 0 is returned, then the task sequence will be queued up using the below mentioned queue strategy, and then will be dispatched to the first available mobile resource. If -1 is returned, then the Dispatcher will do absolutely nothing. In such a case you would use the `movetasksequence()` and `dispatchtasksequence()` commands to execute dispatching logic yourself. See Pass To pick list.

**Queue Strategy** - This pick list returns a "priority" value for the task sequence that is used to rank it in the object's task sequence queue. By default, it will simply return the priority value given to the task sequence when it was created, but the user can also customize task sequence priorities in this field. See Queue Strategy pick list.

### This Page is Used By

- Dispatcher
- TaskExecutor
- Transporter
- Operator
- Crane
- ASRSvehicle
- Robot
- Elevator

## Elevator Tab Page



The screenshot shows a software interface with a tab labeled "Elevator". Below the tab, there are two input fields. The first field is labeled "Elevator Z Size" and contains the value "7.000". The second field is labeled "Elevator Z Location" and contains the value "0.000".

**Elevator Z Size** - This value sets the z size of the elevator's frame, or the size of its vertical poles. Note that this is not the same as the actual z size of the elevator object. These poles are for visualization only. If the elevator must travel to a level outside this range, the elevator platform will be drawn away from the poles.

**Elevator Z Location** - This value sets the z location of the elevator's frame, or the location of its vertical poles. Note that this is not the same as the actual z location of the elevator object, but rather the location of its frame.

## This Page is Used By

Elevator

**FixedResource Page**

Labels

newnumberlabel	0.000	
mylabel	0.000	
otherlabel	0.000	

Add Number Label    Add Text Label

This page allows you to add labels to the FixedResource. It is exactly the same as the labels page in the general properties tab.

## Flow Tab Page

Flow

Output:

☐ Continuously Evaluate Sendto

Send To Port:  
 (Original) sendtoport: First Available (open all ports)

☒ Use Transport      Priority: 0.000      ☐ Preempt

Request Transport From:  
 (Original) transportdispatcher: The object connected to center port number 1

Input:

☒ Pull      ☐ Continuously Evaluate Pull Requirement

Pull From Port:  
 (Original) receivefromport: Pull from any port.

Pull Requirement:  
 (Original) pullrequirement: No specific requirement. (Always returns true.)

For detailed information on this functionality, refer to the FixedResource.

## Output

These parameters determine how the object sends flowitems downstream.

**Continuously Evaluate Sendto** - If checked, the Send To Port will be re-evaluated every time a downstream object becomes available. It's important to note that this is only executed when the downstream object **becomes** available. It does not continuously evaluate just because a downstream object is already available. If you want to manually force a re-evaluation at some other time than when a downstream object becomes available, then you can do so by calling the `openoutput()` command on this object.

**Send To Port** - This pick list returns the output port number connected to the object that the flowitem should be moved to. If 0 is returned, all outputs are opened and the flowitem is moved to the first downstream object that is able to receive it. See Send To Port pick list.

**Use Transport** - If this box is checked, the object will request a transport to move the flowitem downstream. If it is not checked, the flowitem will be moved automatically.

**Priority** - This parameter is only visible if "use transport" is checked. This value sets the priority of the task sequence that will be sent to the transporter or dispatcher. Transporters and dispatchers generally sort task sequences so that sequences with higher priorities will be performed first. Task sequences with the same priority will be performed in the order that they were received.

**Preempt** - This parameter is only visible if "use transport" is checked. If this box is checked, the task sequences sent to the transporter will automatically preempt whatever the transporter is doing at the time. This may cause the transporter to perform tasks that would normally not be allowed, such as carrying more flowitems than its capacity.

**Request Transport From** - This pick list is only visible if "use transport" is checked. This function returns a reference to the Dispatcher or Transporter that will be used to move the flowitem. See Transport Dispatcher pick list.

## Input

These parameters define how an object pulls flowitems from upstream objects.

**Pull** - If this box is checked, the object will pull flowitems from upstream objects. The upstream objects should open all their output ports to allow the object to pull the flowitems it needs.

**Continuously Evaluate Pull Requirement** - This parameter is only visible if "pull" is checked. If checked, the object will re-evaluate the pull requirement for all released flowitems upstream every time a new flow item is released. This is much like the Continuously Evaluate Sendto check box, in that you may need to explicitly call `openinput()` if you want to manually trigger the re-evaluation of the pull requirement.

**Pull from port** - This parameter is only visible if "Pull" is checked. This pick list returns the input port number connected to the object that the next flowitem is to be pulled from. This field is evaluated only on reset of the model and when the pulling object becomes ready to receive its next flowitem. For a Processor with a capacity of 1, this means that the "Pull From Port" field will only be evaluated once right after each flowitem exits the Processor. For a Conveyor, this field will be evaluated after a flowitem enters the conveyor and travels its product length. Opening and closing ports does not trigger this field to re-evaluate. See Receive From Port pick list.

**Pull requirement** - This parameter is only visible if "Pull" is checked. This pick list needs to return either a true or a false (1 or 0). This field is evaluated when considering whether or not to pull in a particular flowitem from the upstream object that was defined by the "Pull from port" field. This field will only be evaluated for flowitems that are in the "ready" state (i.e. `FRSTATE_READY`) meaning the flowitems are ready to leave the upstream object. Basically, the "Pull Requirement" field is evaluated for every "ready" flowitem immediately after the "Pull from port" field gets evaluated. The field is evaluated again for each new flowitem that later becomes ready in the upstream object. If the "Continuously Evaluate Pull Requirement" box is checked, then when a new flowitem becomes ready, the "Pull Requirement" will be re-evaluated for all the "ready" flowitems at that time, and in the order of their rank. See Pull Requirement pick list.

## This Page is Used By

Source  
Queue  
Processor  
Combiner

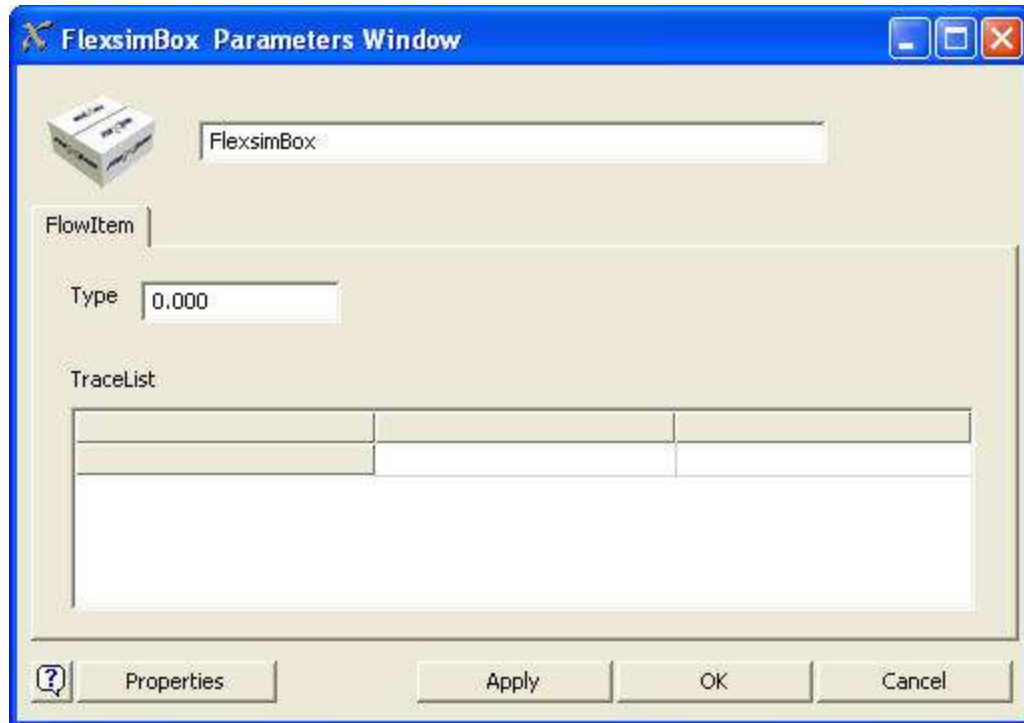


Separator  
MultiProcessor  
Conveyor  
Rack  
Reservoir  
FlowNode  
MergeSort

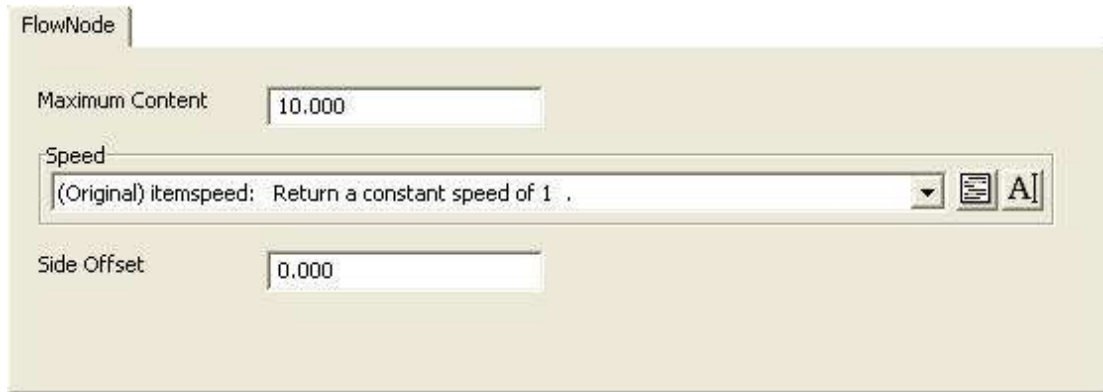


## Flowitem Page

This parameters page allows you to view and change the itemtype of the flowitem. It also lets you view the trace list of the flowitem. The tracelist is only recorded if you tell each object that it enters to write to its trace list, through the entry or exit trigger.



## FlowNode Tab Page



FlowNode

Maximum Content 10,000

Speed  
(Original) itemspeed: Return a constant speed of 1 .

Side Offset 0,000

**Maximum Content** - This field specifies the maximum number of flowitems allowed in the FlowNode at one time. This includes all flowitems that are traveling to other objects from this FlowNode.

**Speed** - This pick list specifies the speed for each flow item that enters the FlowNode. Access variables and pick options for this pick list are exactly the same as the Cycle Time pick list, only the return value is a speed instead of a time value.

**Side Offset** - This value specifies an offset distance for a traveling flowitem. It is for visualization purposes only and does not change the operation of the FlowNode.

## This Page is Used By

FlowNode

## MergeSort Tab Page

MergeSort

Send Requirement  
 (Original) sendrequirement: Always send. [v] [icon] [AI]

Pull Requirement  
 (Original) pullrequirement: No specific requirement. (Always returns true.) [v] [icon] [AI]

	Entry Point		Exit Point	Blocking
Input Port 1	1.000	Output Port 1	4.000	0.000
Input Port 2	2.000	Output Port 2	5.000	0.000
Input Port 3	3.000	Output Port 3	6.000	1.000

**Send Requirement** - This pick list replaces the Send To Port pick list in other FixedResources' Flow page. It is fired when a flowitem passes an output position on the conveyor. It should a true or false value (1 or 0) of whether to send the flowitem out that output port.

**Pull Requirement** - This pick list is the same as the Pull Requirement on other FixedResources' Flow page. See Pull Requirement pick list.

**Output Port / Input Port tables** - These tables allow you to define the position and blocking logic for transfer points on the conveyor. Every input port of the MergeSort has an associated entry location along the length of the conveyor. Every output port has an associated exit location and a blocking parameter. If you connect the input/output ports of the conveyor while this parameters window is open, you will need to reopen the parameters window in order for your changes to be registered. In your Ortho/Perspective view, the positions of the entry/exit points will be drawn as red or green arrows. Input locations have arrows drawn pointing into the conveyor. Exit locations have arrows pointing out of the conveyor. To hide these arrows, set the object to not show ports, either from its Properties window, or from the Edit Selected Objects menu. If your entry/exit points don't appear to be at the right locations, reset the model and they should be re-configured to the correct position.

**Note on entry/exit positions:** if you've changed the entry/exit positions of the mergesort, you will need to reset the model in order for those positions to be placed correctly.

**Entry Point** - This is the position of the entry/exit point, measured from the start of the conveyor.

**Blocking** - This is found only in the Exit locations table. If this value is a 1, then whenever a flowitem passes that exit point and its Send Requirement returns 1, the conveyor will stop until that flowitem has exited. If this value is 0, the conveyor will "attempt" to send the flowitem out that exit point, but if the downstream object isn't ready to receive it, the flowitem will simply continue to the next exit point. We advise that the last exit point on the conveyor be blocking, or else flowitems that reach the end of the conveyor without exiting will simply start at the beginning of the conveyor again.

### This Page is Used By

MergeSort

---

## MultiProcessor Tab Page

**Number of Processes** - This number defines the number of processes at this MultiProcessor. A process is one step in a sequence of steps. Enter the number of processes you want, then hit the Apply Processes button, and the appropriate number of processes will be created.

### Process Tabs

For each process there will be a process tab or page to define the process. Process tabs will have the following information to define the process:

**Process Name** - The process name field allows the modeler to define the name of each process. This name will be used in the state reporting of the MultiProcessor.

**Process Time** - This pick list determines how long a processor spends processing a single flowitem. See also Cycle Time pick list.

**Number of Operators** - This number determines how many operators the object will use during its process time.

**Priority** - This value sets the priority of the task sequence that will be sent to the operator. Operators generally sort task sequences so that sequences with higher priorities will be performed first. Task sequences with the same priority will be performed in the order that they were received.

**Preempt** - If this box is checked, the task sequences sent to the operator will automatically preempt whatever the operator is doing at the time.

**Pick operator** - This pick list returns a reference to the operator or dispatcher that the object is using during the given process. See also Pick Operator pick list.

### **This Page is Used By**

MultiProcessor

---

## NetworkNode Connection Tab Page

For more detailed information on NetworkNode connections, see NetworkNode documentation.

**Note on NetworkNode connections:** Each path between two NetworkNodes contains two one-way connections. This page defines behavior only for the connections extending **from** this NetworkNode **to** other NetworkNodes. If you would like to edit behavior for connections extending from other NetworkNodes to this NetworkNode, then open the Parameters window of those other nodes.

**Name** - This field allows the user to name each connection in the network. These names should be descriptive of any special purpose this connection has in the model.

**Connection Type** - This drop-down list allows the user to define how this connection behaves. There are three options.

**No Connection** - Transporters cannot travel on this connection. The connection is drawn in red in the view window.

**Passing** - Transporters are allowed to pass each other on this connection. The connection is drawn in green in the view window.

**No Passing** - Transporters will not pass each other on this connection. The minimum distance between transporters on the path can be set by the user in the Spacing field of the dialog. These connections are drawn in yellow in the view window.

**Spacing** - This number determines the minimum distance allowed between two transporters on a connection that is designated as no passing. This is the distance from the back of one traveler to the front of the traveler behind it.



**Speed Limit** - This number determines the maximum speed that a traveler can travel along this connection.

**Current Distance** - This number shows you the current distance that is being simulated for that connection. If the virtual distance is specified as 0, then it will be the actual distance of the spline path. Otherwise it will be the distance that is specified in the virtual distance field.

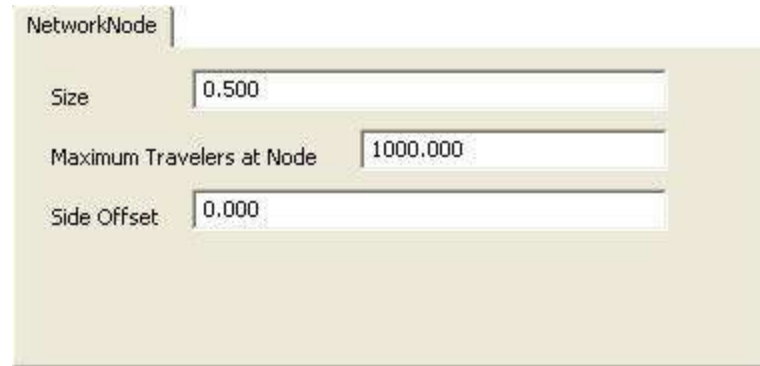
**Virtual Distance** - This number let's you specify an exact distance for the connection.

### **This Page is Used By**

NetworkNode

---

## NetworkNode Tab Page



The screenshot shows a configuration window titled "NetworkNode". It contains three input fields with labels to their left: "Size" with a value of "0.500", "Maximum Travelers at Node" with a value of "1000.000", and "Side Offset" with a value of "0.000".

For more detailed information on these fields, see [NetworkNode](#) documentation.

**Size** - This defines the size of the network node, as well as the width of paths that the network node draws.

**Maximum Travelers at Node** - This number defines how many transporters that are not traveling on the network can be stationed at the node. This would represent the transporters that are not currently executing a travel task, but are doing other things while "stationed" at the node.

**Side Offset** - This number defines a distance to the right of outgoing paths that travelers will be offset. It does not affect the distance that the traveler travels, but is purely for visual purposes, so travelers going in different directions along the same path don't run over each other.

## This Page is Used By

NetworkNode

## Operators Tab Page

### Job

These parameters define how the object uses operators during setup and processing time.

**Use operator(s) for setup** - If this box is checked the object will call for one or more operators during its setup time. The operator(s) will be released after the setup time has expired.

**Use operator(s) for process** - If this box is checked the object will call for one or more operators during its processing time. The operator(s) will be released after the process time has expired.

**Number of Setup Operators** - This parameter is only visible when the "use operator(s) for setup" box is checked. This number determines how many operators the object will use during its setup time.

**Number of Process Operators** - This parameter is only visible when the "use operator(s) for process" box is checked, and the "use setup operators for both setup and process" box is not checked. This number determines how many operators the object will use during its process time.

**Priority** - This parameter is only visible when either of the "use operator(s)" boxes is checked. This value sets the priority of the task sequence that will be sent to the operator. Operators generally sort task sequences so that sequences with higher priorities will be performed first. Task sequences with the same priority will be performed in the order that they were received.

**Preempt** - This parameter is only visible when either of the "use operator(s)" boxes is checked. If this box is checked, the task sequences sent to the operator will automatically preempt whatever the operator is doing at the time. This may cause the operator to perform tasks that would normally not be allowed.

**Use the Setup Operator(s) for Process Time** - This parameter is only visible if both of the "use operator(s)" boxes are checked. If this box is checked, the operators that were called for setup time will be utilized during process time. If this box is not checked, the operators used for the setup time will be released and new operators will be called for the process time. Different operators can be called using a special pick option in the "pick operator" parameter.

**Pick operator** - This field is only visible when either of the "use operator(s)" boxes is checked. This pick list returns a reference to the operator or dispatcher that the object is using during setup or process time. See Process Dispatcher picklist.

## Repair

These parameters define how the object uses operator during repair time after MTBF.

**Use operator(s)** - If this box is checked, the object will call for one or more operator during repair time. The operator(s) will be released after MTTR has expired.

**Number of operator(s)** - This parameter is only visible when the "use operator(s)" box is checked. This number determines how many operators the object will require during its repair time.

**Priority** - This parameter is only visible when the "use operator(s)" box is checked. This value sets the priority of the task sequence that will be sent to the operator. Operators generally sort task sequences so that sequences with higher priorities will be performed first. Task sequences with the same priority will be performed in the order that they were received.

**Preempt** - This parameter is only visible when the "use operator(s)" box is checked. If this box is checked, the task sequences sent to the operator will automatically preempt whatever the operator is doing at the time. This may cause the operator to perform tasks that would normally not be allowed.

**Pick operator** - This parameter is only visible when the "use operator(s)" box is checked. This pick list returns a reference to the operator or dispatcher that the object is using during repair time. See Down Dispatcher pick list.

## This Page is Used By

Processor  
Combiner  
Separator

## Photo Eyes Tab Page

Photo Eyes

OnCover: (Original) photoeyecovertrigger: Do Nothing.

OnUncover: (Original) photoeyeuncovertrigger: Do Nothing.

Number of Photo Eyes: 0 Refresh ☐ Show PhotoEyes


For more detailed information, refer to the conveyor photo eye logic section.

**OnCover** - This pick list is fired in when a flowitem covers the photo eye and when the debounce time has expired without the photo eye being uncovered. See OnCover OnUncover Trigger.

**OnUncover** - This pick list is fired when a flowitem passes the photo eye with no flowitem behind it, uncovering the photo eye. See OnCover OnUncover Trigger.

**Number of Photo Eyes** - This field specifies how many photo eyes to place along the length of the conveyor. Enter a number and press the Refresh button. The specified number of rows will appear in the Photo Eye table.

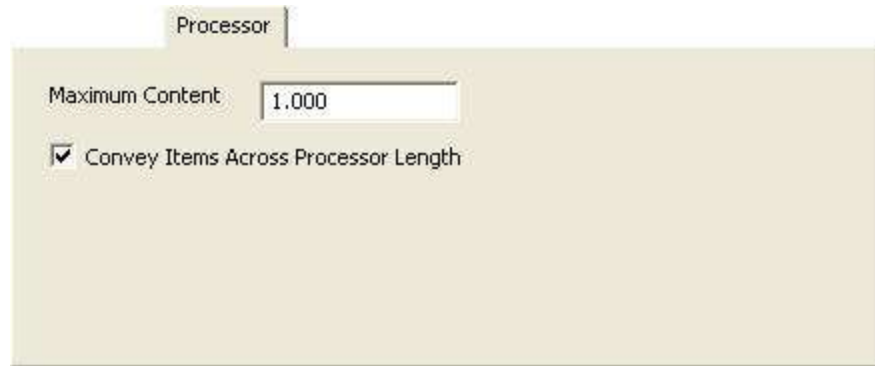
**Photo Eye Table** - This table specifies the location and debounce time of each photo eye. The location is measured from the start of the conveyor.

**Show Photo Eyes** - Check this box if you want the conveyor's photo eyes to be shown.

## This Page is Used By

Conveyor

## Processor Tab Page



The screenshot shows the 'Processor' tab in a software interface. It contains two main settings: a text input field for 'Maximum Content' which is set to '1,000', and a checkbox labeled 'Convey Items Across Processor Length' which is currently checked.

**Maximum Content** - This number defines the number of flowitems that the processor can hold at one time.

**Convey items across processor length** - If this box is checked, flowitems will be seen translating from one side of the processor to the other as their process time elapses. It is for visualization purposes only. If unchecked, entering flowitems will be placed in the middle of the processor and remain until exiting.



## This Page is Used By

Processor

## ProcessTimes Tab Page

**Setup time** - This pick list defines the amount of time that the processor waits after receiving a flowitem to begin processing that flowitem. The model must be compiled after changing this parameter. See Setup Time pick list.

**Process time** - This pick list determines how long a processor spends processing a single flowitem. The model must be compiled after changing this parameter.

**MTBF MTTR and Time Tables** - This tab page lets you add this object as a member of global MTBR MTTR and Time Tables. You can also create, delete, or edit MTBF MTTR or Time Table objects using the buttons at the bottom of the panel. Click on the  button to add the Processor as a member. The object will be added to the list on the right. Click on the  button to remove the object from the list on the right.

**Processor MTBF MTTR** - This opens a window showing deprecated MTBF MTTR functionality. This functionality will work but is obsolete and we advise using the global MTBF MTTR functionality instead.

### This Page is Used By

Processor  
Combiner  
Separator

## Queue Tab Page

Queue

Maximum Content: 10.000

Batching

☐ Perform Batching

Target Batch Size: 2.000

Max Wait Time: 0.000

☐ Flush contents between batches

Visual

Item Placement: Stack Vertically

Stack Base Z: 0.000

**Maximum Content** - This is the maximum number of flowitems the queue can hold at once.

## Batching

These fields define the queue's batching abilities.

**Perform batching** - If this box is checked, the queue will accumulate flowitems into a batch before releasing them downstream. Accumulation continues until either the target batch size is met or the max wait time expires. If this box is not checked, no batching will occur, and flowitems may leave as soon as downstream objects are available.

**Target Batch Size** - This number defines the size of the batches that the queue will gather before sending the flowitems downstream. Flowitems are sent downstream individually.

**Max Wait Time** - This number is the maximum length of time that the queue will wait before sending the flowitems downstream. If this time expires and the batch size has not been met, the currently collected batch will be released anyway. If 0 is specified in this field, then there is no maximum wait time, or in other words the queue will wait indefinitely.

**Flush contents between batches** - If this box is checked the queue will not allow new flowitems to enter until the entire current batch has left.



## Visual

These parameters define how the queue locates the flowitems within itself when they enter.

**Item Placement** - This defines how the flowitems are placed in the queue visually.

- **Stack Vertically** - The flowitems are stacked on top of each other. The flowitem at the bottom of the pile is the one that has been in the queue the longest.
- **Horizontal Line** - The flowitems are lined up horizontally. The one closest to the output ports of the queue is the one that has been in the queue the longest.
- **Stack inside Queue** - The flowitems are stacked in rows inside the queue. The flowitems' positions will move if a product ranked before them is taken out of the queue. If you would like the product positions to stay the same once they are in the queue, then have the queue be LIFO by having downstream objects pull only the last product in the queue.
- **Do Nothing** - Flowitems are all placed at the same point in the queue. This may make it appear as if the queue is only holding one flowitem.

**Stack Base Z** - This number defines the height where the queue begins placing flowitems that are being stacked vertically or inside the queue.



## Rack Tab Page

Rack

☐ Floor Storage ☐ Mark shelves that have called a transporter

Shelf tilt amount: 0.000 Picking/Placing Y Offset: 0.000

Maximum Content: 1000000000.000 Opacity: 1.000

Place in Bay  
 (Original) placeinbay: Place it in a random bay (cells are assumed to have unlimited capacity) [List Icon] [AI]

Place in Level  
 (Original) placeinlevel: Place it in a random level (cells are assumed to have unlimited capacity) [List Icon] [AI]

Minimum Dwell Time  
 (Original) minimumstaytime: Return constant time of 0 [List Icon] [AI]

**Floor Storage** - If this box is checked, then instead of having a vertical storage rack, the rack will simulate storage space on the floor. Looking down from above the rack, bays are vertical columns, and levels are horizontal rows on the rack.

**Shelf Tilt Amount** - This number defines the amount of tilt of items placed in a given cell of the rack, as some racks have products slide down from the back of the rack to the front.

**Maximum Content** - This number defines how many flowitems the rack will be allowed to hold at a given time.

**Mark Shelves that have called a transporter** - This check will highlight the shelf in a red color when it has called the transporter for pickup.

**Picking/Placing Y Offset** - This value is used to configure how close transport objects come to the rack when they drop off or pick up flowitems from the rack. This is especially useful if operators are used to drop off and pick up from the rack, because often they will walk into the middle of the rack to get a flowitem. Specify a value of 1, for example, and the operators will keep better distance from the rack when dropping off and picking up flowitems.

**Opacity** - This value allows the drawing of the rack to be translucent, so that if there are several racks in the same area, many of them can be seen because of the translucency of the racks in front. A value of 0 means totally transparent, and a value of 1 means totally opaque.

**Place in bay** - This pick list is called when a flowitem is entering the rack. It returns which bay the flowitem will be placed in. See Place in Bay pick list.

**Place in level** - This pick list is called when a flowitem is entering the rack. It returns which level the flowitem will be placed in. See Place in Level pick list.

**Minimum Dwell Time** - This pick list returns a value of how long a flowitem must stay in the rack before it is released to continue downstream. You can also return a value of -1 from this function so the Rack will not release the item at all, and then implement your own releasing strategy using the `releaseitem()` command. See Minimum Staytime pick list.

### This Page is Used By

Rack

---

## Rack Size Table Tab Page

This page allows you to configure the layout of bays and levels on the rack. If you want a simple grid of bays and levels, then you can specify these settings from the Basic panel on the left. If you want to specify different parameters for different bays and levels of the rack, then you can use the Advanced panel on the right to configure each bay individually.

The screenshot shows the 'SizeTable' tab with two main panels: 'Basic' and 'Advanced'.

**Basic Panel:**

- Number of Bays: 10
- Width of Bays: 2.00
- Number of Levels: 10
- Height of Levels: 1.00
- Apply Basic Settings button

**Advanced Panel:**

- A list of bays from Bay 1 to Bay 10. Bay 1 is selected.
- Buttons: Duplicate Bay, Delete Bay, Add Level, Delete Level.
- Bay 1 configuration:
  - Bay Width: 2.00
  - Level Location: 0.00
  - Level Heights table:

Level Heights	
Level1	1.00
Level2	1.00
Level3	1.00
Level4	1.00
Level5	1.00
Level6	1.00
Level7	1.00
Level8	1.00
Level9	1.00
Level10	1.00

### Basic Panel

Use the basic panel if your rack is just a simple grid of uniform cells. You can also use it to set basic settings for the rack before going in and editing individual bays in the advanced panel. Once you have specified the basic dimensions for the rack, click the Apply Basic Settings button, and these settings will be applied to the rack.

**Number of Bays** - This value is for the number of bays (columns) for the rack.

**Width of Bays** - This value defines the default width of each bay.

**Number of Levels** - This value is for the number of levels in the rack.

**Height of Levels** - This value is for the default height of each level. This value can be edited for each level and bay using the advanced edit.

### Advanced Panel

Use this panel to configure individual bays and levels on the Rack. On the left side of this panel is a list of all the bays in the rack. Select a bay and configure it by using the options and buttons on the right side of the panel. Changes should immediately be shown on the Rack in the orthographic/perspective view. If they do not show up immediately, hit the Apply button, and they should appear.

**Duplicate Bay** - This button duplicates the currently selected bay and adds it to the end of the rack.

**Delete bay** - This button deletes the currently selected bay in the rack.

**Add Level** - This button adds a level to the end of the currently selected bay.

**Delete Level** - This button deletes a level from the end of the currently selected bay.

**Bay Width** - This field specifies the width of the currently selected bay.

**Level Location** - This field specifies the initial z location of the first level of the selected bay.

**Level Heights Table** - Here you can specify the height of each level in the currently selected bay.

### **This Page is Used By**

Rack



## Recorder Parameters

The Recorder does not use the standard tab-page dialog box. It uses its own custom dialog. When a Recorder is placed in your model you can double click on the object to open this window.

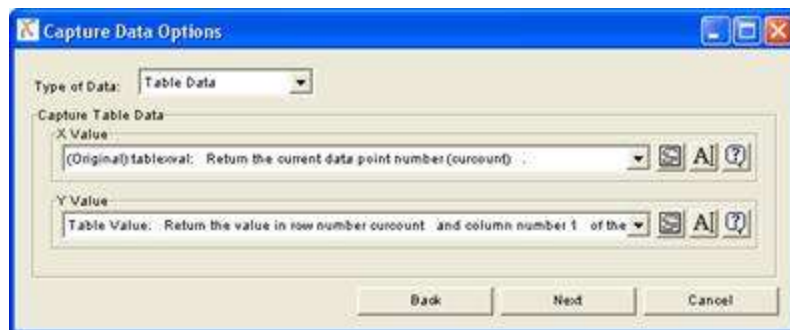


### Data Capture Settings

The data capture settings are used to define the type of data that you want to display or capture. Click on this button before clicking on the Display Options button. The 3 types are Table Data, Standard Data, and User-defined Data.

### Table Data

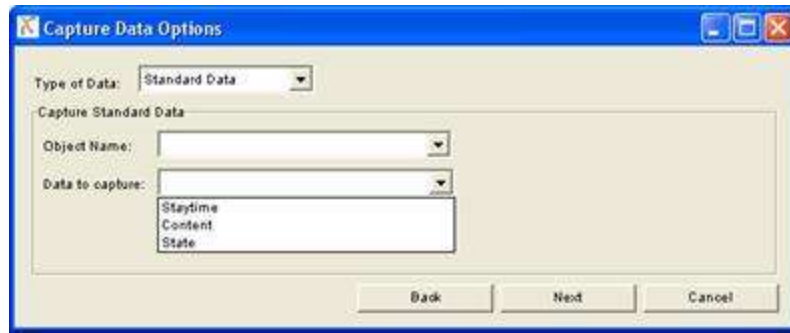
Table data refers to information that is currently contained in a Flexsim table.



By selecting a pick option and then selecting the code template button you can define the table and data you want to display. The data in the table must have been previously written to the table by using the trigger options on an object or some other means. When this option is selected, the Recorder generates a set of x/y points, depending on the input to the pick lists. It then draws a graph of those points, according to the graph type specified in the Display Options window. For more information on the pick list, refer to the Table X Val pick list and the Table Y Val pick list.

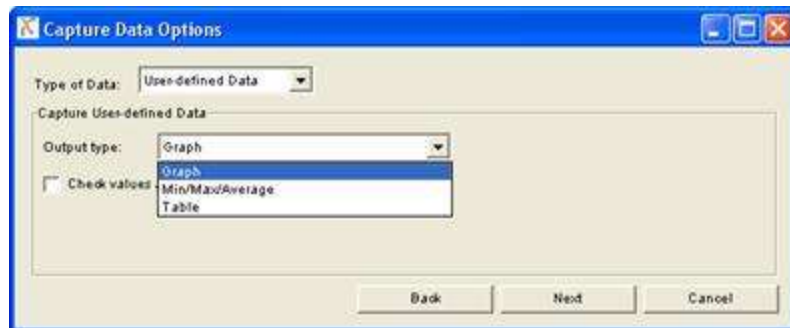
### Standard Data

Standard data refers to data that is automatically collected on every object. Standard data includes Staytime, Content, and State information. This is the same information that is shown in the Properties Dialog under the statistics tab. Use this option if you would like to display a 3D content graph, 3D staytime histogram, or 3D state pie graph in your model. Once you have selected standard data you must then select the object that you would like to display information on, then select the type of standard data to display.



## User-defined Data

User-defined data refers to specific data that the modeler might want to show in a graph or record to a table. Here the Recorder "watches" the value of some node in your model, like the state of an object, or the content of the object, etc. When you select user-defined data you are presented three options. You can choose Graph, Min/Max/Average, or Table



**Check values at set intervals** - If this box is checked, then data points will be recorded at set time intervals. If the box is not checked, then data points will be recorded whenever the value of the node changes.

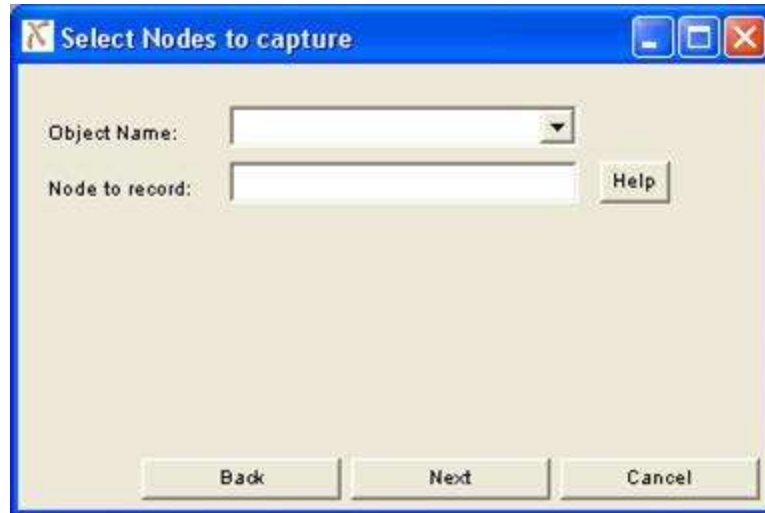
**Length of intervals** - This field only applies if the "Check values at set intervals" box is checked. It is the span of time between each recorded data point.

**Weight averages by time/occurrences** - These radio buttons only apply if you've chosen the Min/Max/Average option for Output Type. Statistically there are two ways to calculate a running average for a variable: 1) by time and 2) by occurrences. If time is chosen, then the time that the watched node had a certain value is weighted into the average. If occurrences is chosen, then each occurrence of a value will have

the same weight as any other occurrence, and the time that the node stayed at a given value does not matter.

## Graph

The graph option is used to graph the changing values of a node on an object in your model. Any number data node of any object in your model may be used. Click the Help button next to the "Node to record" field to see a list of possible node names that you may wish to use. Choose the object in the model from the drop down pick list, and type in the name of the node you want to graph.

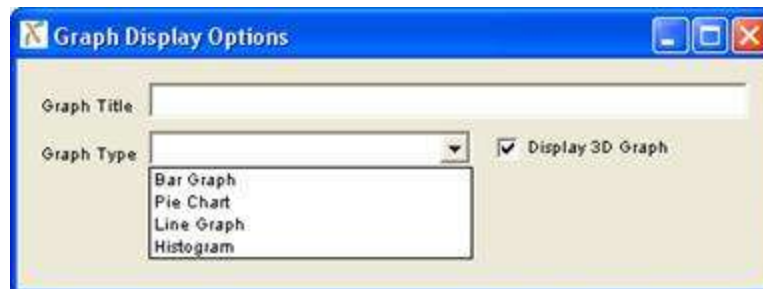


The object will be selected from the pick list of the model objects. The node to record will be the name of a node on the object. By pressing the Help button a list of the possible nodes will be displayed.





You must type the name of the node into the "Node to record" field. When you press the next button you then must define the title of the chart and select the chart type from the pick list.

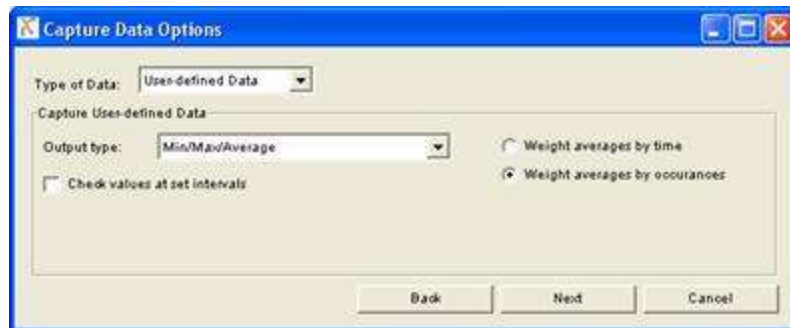


**Note:** Be aware that the type of data that is being requested may not fit any graph type. For example, the stats\_staytimemax is best shown in a line graph, not a pie chart. Once a graph type is selected you can set up the parameters of the graph for better viewing. To view the graph double click on the recorder and select "View Captured Data".

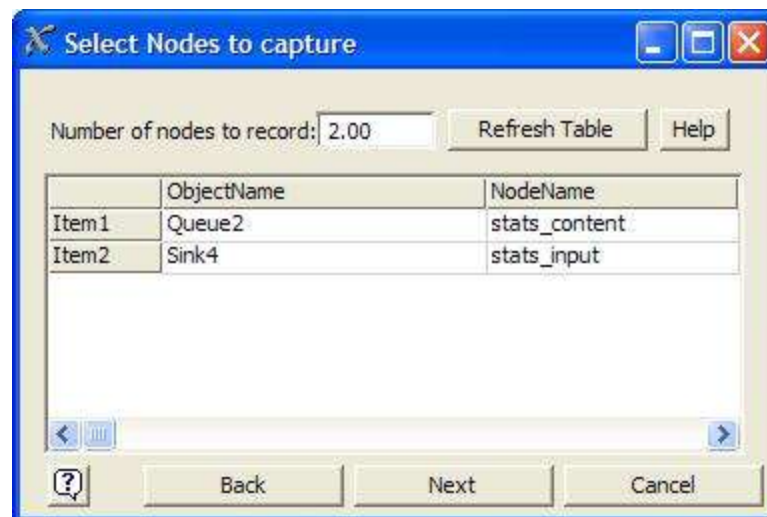
After choosing a graph type, additional edit fields specific to that type of graph will be visible. For a Bar Graph and Line Graph, you will be asked to specify the maximum number of data points you want the Recorder object to record and display in a graph. For instance, if you type in 100 for the number of data points to record for a Line Graph, you will only see the last 100 values recorded for the previously declared node. For a Pie Chart and Histogram, the Recorder object will automatically record the number of times the value of your node fell within one of the divisions or "buckets" you declare for the graph.

**Min/Max/Average**

The min/max/average option is used to view minimum, maximum, and average values of a specific node or list of nodes in table format.



Click Next to define the nodes you want to record the minimum, maximum and average values for. Specify the total number of nodes you want to record and then click the Refresh Table button. At this point you must specifically state both the name of the object and the name of the node on the object that you wish to record. Click the Help button for a list of possible nodes you may wish to record.



To view the recorded data in a table for the nodes you specified, click the "View Captured Data" button on the main parameters window for the Recorder.



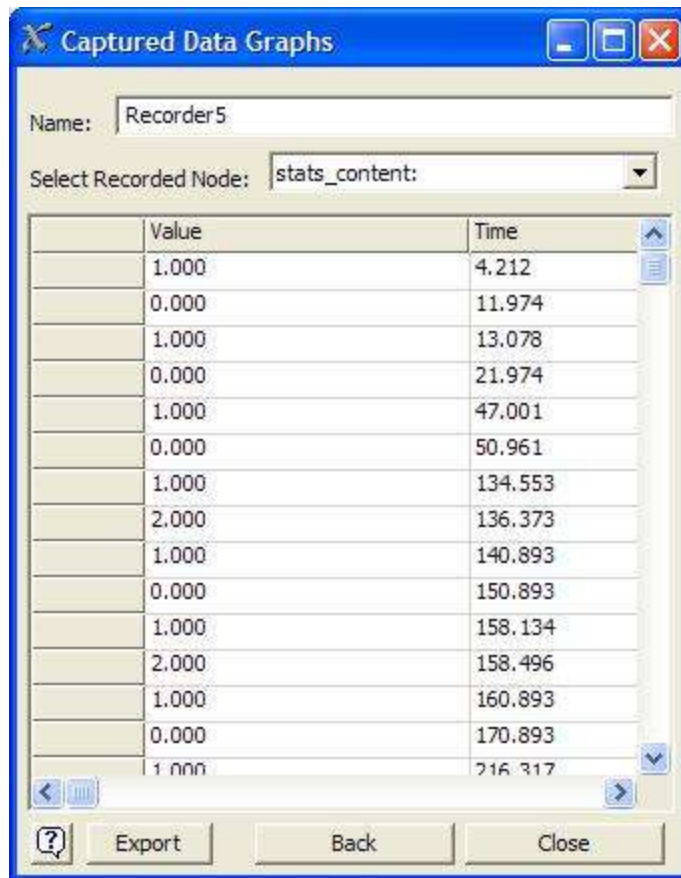
The data table will look something like this. You may have this window open while the model is running to see the values update dynamically.

 A screenshot of the "Captured Data Table" window. It has a title bar with a blue background and standard window controls. The main area is light beige. At the top, there's a "Name:" label followed by a text box containing "Recorder5". Below this is a table with 8 columns: "Object", "Node", "MinValue", "MaxValue", "AvgValue", "CurrentValue", and "ReadCount". The table has two rows of data. Below the table is a "Export" button with a help icon to its left. At the bottom right are "Back" and "Close" buttons.
 

	Object	Node	MinValue	MaxValue	AvgValue	CurrentValue	ReadCount
stats_content	Queue2	stats_content	0.000	10.000	4.000	4.000	1163.000
stats_input	Sink4	stats_input	0.000	223.000	223.000	223.000	1162.000

### Table

The "Table" option for output type will write the value of the user-defined nodes to a table every time they change or on a set time interval as determined by the user. Whereas the "Min/Max/Average" option only records point statistics for the nodes, this option records a continuous stream of data points for each of the nodes. You will define which nodes to capture exactly the same way as you do for the "Min/Max/Average" option. The following is an example of the data table you will see when you click on the "View Captured Data". Notice that the table is displayed for one node at a time. You select the other pre-defined nodes with the drop down pick list. Also note that you can export the data to a file with the Export button.



**Captured Data Graphs**

Name: Recorder5

Select Recorded Node: stats\_content:

	Value	Time
	1.000	4.212
	0.000	11.974
	1.000	13.078
	0.000	21.974
	1.000	47.001
	0.000	50.961
	1.000	134.553
	2.000	136.373
	1.000	140.893
	0.000	150.893
	1.000	158.134
	2.000	158.496
	1.000	160.893
	0.000	170.893
	1.000	216.317

? Export Back Close

The user-defined options for the Recorder object are advance features of the software. Your use of these options will increase as you become more familiar with the Flexsim software.

## Reservoir Tab Page

Reservoir

Maximum Content 10.000

High Mark 10.000

Middle Mark 5.000

Low Mark 0.000

Incoming Flow Rate  
(Original) inflowrate: Allow one flowitem in or out every 1 second(s).

Outgoing Flow Rate  
(Original) outflowrate: Allow one flowitem in or out every 1 second(s).

**Maximum Content** - This number is the maximum number of volume units that the reservoir can hold. By default, each flowitem represents one volume unit, but this can be changed using the Incoming Flow Rate options.

**High Mark** - When the content of the reservoir rises or falls to this amount, trigger functions are called on the reservoir.

**Middle Mark** - When the content of the reservoir rises or falls to this amount, trigger functions are called on the reservoir.

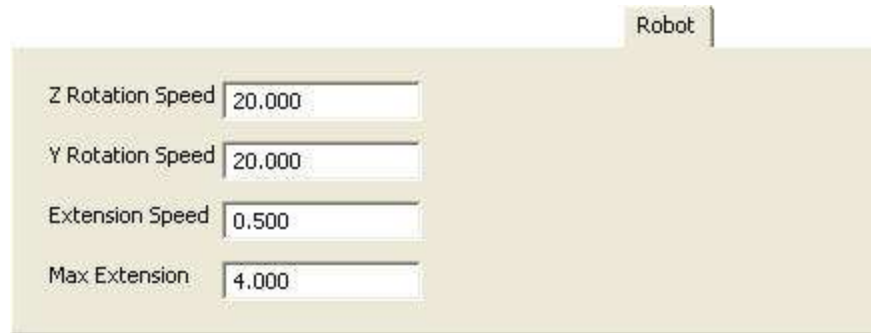
**Low Mark** - When the content of the reservoir rises or falls to this amount, trigger functions are called on the reservoir.

**Note on content values:** The content values do not necessarily represent the number of flowitems in the reservoir. In the incoming flow rate function, you can specify how much volume you want each flowitem to represent. By default, each flowitem represents 1 unit of volume, but you can have the pick list reference labels, itemtypes, etc, to specify each flowitem's volume representation. The maximum content and mark values are calculated based on the volume units, and not on the number of flowitems in the Reservoir.

**Incoming Flow Rate** - This pick list returns how many seconds the reservoir will wait after a flowitem enters before allowing the next flowitem to enter. Within this function you can also set a number that represents the "volume" that the entering flowitem takes up. By default this volume is 1, but alternate values can be set, and these alternate values will be used for the maximum content, high mark, middle mark, and low mark values. See Flow Rate pick list.

**Outgoing Flow Rate** - This pick list returns how many seconds the reservoir will wait after a flowitem leaves before allowing the next flowitem to leave. See Flow Rate pick list.

## Robot Tab Page



Robot	
Z Rotation Speed	20.000
Y Rotation Speed	20.000
Extension Speed	0.500
Max Extension	4.000

**Z Rotation Speed** - This value sets how fast the robot rotates around its z-axis whenever it needs to move. It is specified in degrees per time unit. For example, if the robot needs to make half a turn in one second the rotation speed should be set to 180.

**Y Rotation Speed** - This value sets how fast the robot rotates around its y-axis whenever it needs to move. It is specified in degrees per time unit. This is the rotation speed it uses to rotate to objects above and/or below it.

**Extension Speed** - This value sets how fast the robot's arm will extend to reach a flowitem or its destination. It is specified in distance units per time unit.

**Max Extension** - This value is the distance that the arm can reach when it is fully extended. Each arm section is always half of this length.

## This Page is Used By

Robot

## Separator Tab Page

The screenshot shows a tabbed interface with the 'Separator' tab selected. Inside the tab, there are three options: 'Unpack' (selected with a radio button), 'Split' (unselected with a radio button), and 'Convey Items Across Separator Length' (checked with a checkbox). Below these options is a section titled 'Split or Unpack Quantity' which contains a text box with the text '(Original) splitquantity: Unpack the full content of the flow item.' To the right of the text box are three icons: a dropdown arrow, a list icon, and a button labeled 'A'.

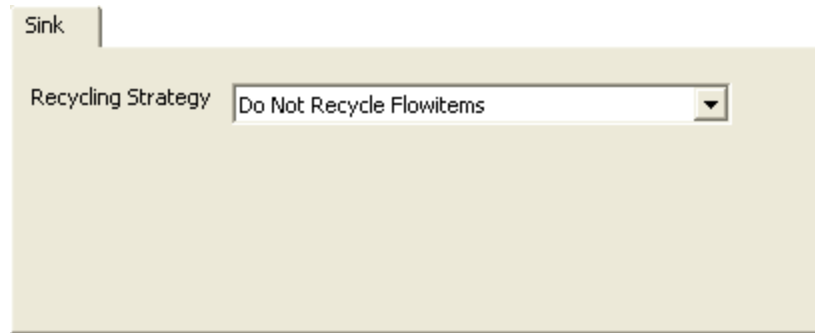
**Unpack** - If this box is checked, the separator will assume that the flowitem that entered contains other flowitems that need to be removed.

**Split** - If this box is checked, the separator will make duplicate copies of the entering flowitem.

**Convey Items Across Separator Length** - If checked, flowitems will travel across the Separator during their process time.

**Split/Unpack Quantity** - This pick list returns how many flowitems will be unpacked or duplicated by the separator. Refer to the split quantity pick list.

## Sink Page



**Recycling Strategy** - The recycling strategy drop-down list lets you specify how the Sink recycles flowitems. Recycling flowitems can significantly increase the speed of your model. By default, the Sink simply destroys flowitems that enter. To configure the sink to recycle flowitems, select from the drop-down list a flowitem in the flowitem bin that this sink's flowitems originate from.

**Note on recycling flowitems:** If you configure a Sink to recycle flowitems, those flowitems will be returned to the flowitem bin as-is. This means that if there were changes made to the flowitems during the model run, you will need to change them back to their original state from the entry trigger of the sink, so that when they are recycled, they will have the correct data, visuals, etc.

## This Page is Used By

Sink



## Source Tab Page

Source

Arrival Style: Inter-Arrival Time

FlowItem Class: Textured Colored Box

☐ Arrival at time 0

Item Type: 1.000

Inter-Arrivaltime

(Original) interarrivaltime: An Exponential distribution with location value of 0 and scale

### Arrival Style

This is used to specify the way that the source creates flowitems.

**Inter-Arrival time** - After a set period of time, the source creates one flowitem. This repeats until the model is stopped.

**Arrival Schedule** - The source follows a table that defines when flowitems are created, how many there are, and what itemtypes are assigned to them.

**Arrival Sequence** - The source follows a table that defines what order flowitems are created in. Flowitems are created as fast as the source can move them downstream.

### FlowItem Class

This is used to define what class of flowitem the source will create. To edit and view flowitem classes, use the FlowItems button in the toolbar.

## Inter-Arrivaltime Usage

These fields define how the source creates flowitems when inter-arrival time is selected as the arrival style.

**Arrival at time 0** - If this is checked, one flow item will be created at time 0. The next will be created at the end of the first inter-arrival time.

**Item Type** - Flow items that are created will be assigned the itemtype defined here.

**Inter-Arrival time** - A function that returns the amount of time the source should wait before creating the next flowitem. The model must be compiled after changing this parameter.

## Arrival Table Usage

These fields define how the source creates flowitems when arrival schedule or arrival sequence is selected as the arrival style.

**Number of Arrivals** - This specifies how many rows are in the arrival table.

**Refresh** - Pressing this button will update the on-screen table after changing the number of arrivals.

**Repeat Schedule/Sequence** - If this box is checked, the schedule or sequence will continually repeat itself until the model is stopped. In the case of a schedule, the arrival defined in row 1 will occur 0 seconds after the time defined for the arrival in the last row. This means that the arrival time for the very first row of the table is only used once for a given simulation. Note that this initial delay can be used as a “warm-up” period. If you would like to specify an interval time between the arrival time of the last row and the arrival time of the first for when repeated, then add another row to the end of the table and specify a quantity of 0.

**Refresh** - Each row in the table represents one arrival of flowitems at the source. The columns in the table define the details of each arrival (i.e time,name,type,quantity).

**ArrivalTime** - This is the time that the arrival will occur.

**ProductName** - All the flowitems that the source creates during this arrival will be given this name.

**ItemType** - All the flowitems that the source creates during this arrival will be given this itemtype.

**Quantity** - This number specifies how many flowitems will be created during this arrival.

## This Page is Used By

Source

## TaskExecuter Tab Page

Transporter72542 Parameters Window

Transporter72542

TaskExecuter | Collision | Dispatcher | TaskExecuterTriggers | Transporter

Capacity: 1.00 Acceleration: 1.00

Maximum Speed: 2.00 Deceleration: 1.00

☒ Rotate while travelling Travel offsets for load/unload tasks

Load Time  
Return constant time of 0

UnLoad Time  
Return constant time of 0

"Break to" Requirement  
Take only new tasksequences (no partially completed sequences)

MTBF MTTR | Time Tables

All MTBF's Member MTBF's

>> <<

Add Delete Edit Refresh

? Properties Apply OK Cancel

**Capacity** - This number is maximum number of flowitems that the Operator or Transport can carry at a given time.

**Maximum speed** - This is the fastest that the Operator or Transporter can travel.

**Acceleration** - This number is how fast the Operator or Transporter gains speed until it reaches its maximum speed or needs to slow down to reach its destination node.

**Deceleration** - This number is how fast the Operator or Transporter loses speed as it approaches its destination node.

**Travel offsets for load/unload tasks** – This box provides 3 options. If "Travel offsets for load/unload tasks" is selected, the transporter/operator will be move to the

exact point where the flowitem is being picked up or dropped off. If "Do not travel offsets for load/unload tasks", it will travel to the origin of the destination object and pick up or drop off the flowitem there. In the case where the transporter/operator is using networknodes to travel to the destination, it will travel to the networknode attached to the destination object and then stop there. The option "Do not travel offsets and block space on network" only applies when the object is connected to a network of nonpassing paths. If it is chosen, then the object will arrive at the node, finish its travel, and while it is doing the load/unload operation, it will continue to take up space on the network, and block other objects traveling on the path.

**Rotate while traveling** – If this box is checked, the transporter/operator will rotate as needed in order to orient itself in the direction of travel. If the box is not checked, it will always face the same direction. This option will not affect the statistics of the model if checked or unchecked. It is simply for visualization.

**Load time** - This pick list returns how long it takes this Operator or Transporter to load the flowitem.

**Unload time** - This pick list returns how long it takes this Operator or Transport to unload the flowitem.

**Break to Requirement:** This field is executed when the TaskExecuter comes to a break task or callsubtasks task. The return value is a reference to a task sequence. The logic within this field should search the TaskExecuter's task sequence queue, and find a task sequence that is appropriate to break to.

### This Page is Used By

TaskExecuter  
Transporter  
Operator  
Crane  
ASRSvehicle  
Robot  
Elevator

## Traffic Control Network Node Page

Member Network Nodes	Selected Node's Traffic Controls
1: NN2	1: TrafficControl75
2: NN1	2: TrafficControl74 -- This TrafficControl
3: NN3	3: TrafficControl76

Move Up  
Move Down  
Delete

The NetworkNode page of the TrafficControl object lets you edit a TrafficControl's connections to network nodes. The panel on the left shows all NetworkNodes that are connected to the TrafficControl. Select one of these nodes, and the panel on the right will refresh to a new list of traffic controls. The panel on the right shows all traffic controls that are connected to the NetworkNode you have selected on the left. The traffic control you are editing will have an extra " -- This TrafficControl" text, letting you know how this traffic control is situated among other traffic controls. The ranking of traffic controls in the right panel can have a significant effect on the behavior of the network. For more information on traffic control ranking, refer to the section on the TrafficControl. To move a traffic control up in its rank, press the Move Up button. To move a traffic control down rank, press the Move Down button. To delete a NetworkNode's connection to a TrafficControl, press the Delete button.

### This Page is Used By

TrafficControl

## Traffic Control Page

For more detailed information on the TrafficControl object, refer the TrafficControl object.

The screenshot shows the 'Traffic Control' dialog box with the 'Traffic Control Mode' set to 'Untimed Traffic Modes'. The 'Number of Modes' is 2 and the 'Number of Entries' is 3. The 'Search For Best Mode' checkbox is unchecked. Below these settings is a table with 8 columns: Max\_Nr, n\_a, From, To, From, To, From, To. The table contains two rows of data.

	Max_Nr	n_a	From	To	From	To	From	To
	1000.000	0.0000	NN1	NN2	NN2	NN1	NN3	NN1
	0.0000	0.0000	NN2	NN1	NN2	NN3	NN1	NN3

**Traffic Control Mode** - This defines the method by which the object controls traffic. There are two options: mutual exclusion or un-timed traffic modes.

### Mutual Exclusion

Mutual exclusion is used to only allow a given number of travelers into the Traffic Control's area, regardless of which paths they are on. Here you simply specify the maximum number value.

The screenshot shows the 'Traffic Control' dialog box with the 'Traffic Control Mode' set to 'Mutual Exclusion'. The 'Maximum Number in Area' is set to 1.0000.

**Maximum Number in Area** - This value defines the maximum number of travelers that are allowed to be in the traffic control's area.

### Un-timed Traffic Modes

Un-timed traffic modes are used if you want to control traffic based on each individual path in the object's traffic control area. A mode is defined by one row in the modes table. For each mode you can define a set of paths between nodes in the traffic control area. When the traffic control is in a given mode, travelers are only allowed into the traffic control area if the path they are entering on is one of the paths defined in the current mode. The traffic control will stay in the same mode until there are no travelers left in the traffic control area, after which it will take the first traveler

that arrives and find a mode that contains that traveler's requested entry path. This is why it is an un-timed mode. There is no limit on the amount of time that a traffic control may stay in the same mode.

	Max_Nr	n_a	From	To	From	To	From	To
	1000.000	0.0000	NN1	NN2	NN2	NN1	NN3	NN1
	0.0000	0.0000	NN2	NN1	NN2	NN3	NN1	NN3

**Number of Modes** - This is the number of modes, or rows in the table. Enter the number of modes you want, and click the Apply button, and the appropriate number of rows will be created.

**Number of Entries** - This is the maximum number of From/To entries (or columns) that you will need for your modes. If some modes don't need all of the columns, just leave them blank.

**Search for Best Mode** - If this box is checked, then whenever the traffic control gets an entry request for an entry path that is not in its current mode, it will search through its modes to see if there are any other modes that include all paths already entered as well as the new path. If so, it will change to that new mode and allow the traveler's entry. Note that this may slow down the model, since the traffic control must search the table every time an entry request is made.

### Mode Table Entries

**Max\_Nr** - This value specifies a maximum number of travelers allowed in when the traffic control is in that given mode. It is much like the maximum number value in mutual exclusion mode.

**n\_a** - This value is reserved for future development, when timed traffic modes are implemented.

**From/To Entries** - For each path of a mode, you specify the node from which the path extends, and the node to which the path extends. Enter the name of the nodes. Note that one entry describes only one direction of a node-to-node connection. Thus to specify both directions of a path, you will need to make two From/To entries, one in each direction.

**This Page is Used By**





## Transporter Tab Pages



The screenshot shows a software window titled "Transporter". Inside the window, there is a label "Lift Speed:" followed by a text input field containing the value "1.000". The window has a light beige background and a thin black border.

**Lift speed** - This number is how fast the lifts on the Transporter move up and down.

## This Page is Used By

Transporter

## Object Trigger Functions

**Note:** If changes are made to any trigger functions, the model must be compiled before running.

**OnArrival:** This function is called on a `NetworkNode` when a traveler arrives at the node. If this function returns a non-zero value, then the traveler's next path to go to will be changed to the path specified by the return value. This return value is the rank of the next path, as shown in the `NetworkNode`'s `Paths` tab page. See `OnArrival` trigger pick list.

**OnBreakDown:** This function is called on an object when its MTBF expires. See `Breakdown/Repair` Trigger pick list.

**OnContinue:** This function is called on a `NetworkNode` when a traveler continues on to the next path leading out of the node. See `OnContinue` trigger pick list.

**OnConveyEnd:** This function is called on a `Conveyor` when a flowitem reaches its end. See `Process Finish` Trigger pick list.

**OnCreation:** This function is called on a `Source` when it creates a new flowitem. See `Creation` Trigger pick list.

**OnEndCollecting:** This function is called on a `Queue` when it has reached its batch limit. See `Process Finish` Trigger pick list.

**OnEndDwellTime:** This function is called on a `Rack` when a flowitem's dwelltime has expired and it is ready to leave. See `Process Finish` Trigger pick list.

**OnEmpty:** This function is called on a fluid object when all of the material that it was holding left and its content became 0. See `Empty/Full` Trigger pick list.

**OnEntry:** This function is called on an object when a flowitem is moved into it. See `Entry/Exit` Trigger pick list.

**OnExit:** This function is called on an object when a flowitem is moved out of it. See `Entry/Exit` Trigger pick list.

**OnFallThroughHighMark:** This function is called on a reservoir when the content falls below the designated high mark. See `Rise/Fall Through Mark Triggers` pick list.

**OnFallThroughLowMark:** This function is called on a reservoir when the content falls below the designated low mark. See `Rise/Fall Through Mark Triggers` pick list.

**OnFallThroughMiddleMark:** This function is called on a reservoir when the content falls below the designated middle mark. See `Rise/Fall Through Mark Triggers` pick list.

**OnFull:** This function is called on a fluid object when its content reaches its maximum content. See `Empty/Full` Trigger pick list.

**OnLoad:** This function is called on an Operator or Transport when it completes loading a flowitem. See Load/Unload Trigger pick list.

**OnMessage:** This function is called on an object when another object sends a message to it using the `sendmessage()` or `senddelayedmessage()` commands. See Message Trigger pick list.

**OnProcessFinish:** This function is called on an object when its process time has expired. See Process Finish Trigger pick list.

**OnRepair:** This function is called on an object when its MTTR expires. See Breakdown/Repair Trigger pick list.

**OnResourceAvailable:** This function is called when a downstream resource of a Dispatcher becomes available.

**OnRiseThroughHighMark:** This function is called on a reservoir when the content rises above the designated high mark. See Rise/Fall Through Mark Triggers pick list.

**OnRiseThroughLowMark:** This function is called on a reservoir when the content rises above the designated high low. See Rise/Fall Through Mark Triggers pick list.

**OnRiseThroughMiddleMark:** This function is called on a reservoir when the content rises above the designated middle mark. See Rise/Fall Through Mark Triggers pick list.

**OnSetupFinish:** This function is called on an object when its setup time has expired. See Process Finish Trigger pick list.

**OnUnload:** This function is called on an Operator or Transport when it completes unloading a flowitem. See Load/Unload Trigger pick list.

## FluidBlender Tab Page

The screenshot shows the FluidBlender configuration window. It has a title bar 'FluidBlender'. Inside, there are two input fields at the top: 'Maximum Content' with the value '100.00' and 'Target Product ID' with the value '1.00'. Below these are two main sections: 'Input Ports' and 'Output Ports'. The 'Input Ports' section contains a single input field 'Maximum Input Rate' with the value '1.00'. The 'Output Ports' section contains two input fields: 'Maximum Object Rate' with '1.00' and 'Maximum Port Rate' with '1.00'. Below these is a label 'Output port scale factor (0-1)' followed by a table with two columns and two rows. At the bottom, there is a section 'Adjust Output Rates' with a dropdown menu showing 'Do nothing', a help button (question mark icon), and an apply button (A icon).

**Maximum Content** - The maximum amount of fluid material that this object can hold at any time.

**Target Product ID** - The ProductID that will be assigned to the material that leaves this object.

**Adjust Output Rates** - This function is called every tick and allows the user to change the output rates and scale factors during a model run. There are access variables that are references to the nodes that hold the rates and scale factors. To read the current rate (object or port) the `getnenum()` command should be used. To change the rate the `setnenum()` should be used. To read or change the scale factors `getnenum()` and `setnenum()` should be used in conjunction with the `rank()` command.

### Input Ports

These parameters affect how the object receives material from upstream objects.

**Maximum Input Rate** - The maximum rate that material will be allowed into this object through all input ports combined. The actual input rate is based on the amount of material available upstream and the space available in this object.

## Output Ports

These parameters affect how the object sends material to downstream objects.

**Maximum Object Rate** - The maximum rate that material will leave this object through all output ports combined.

**Maximum Port Rate** - The maximum rate that material will leave this object through any one port.

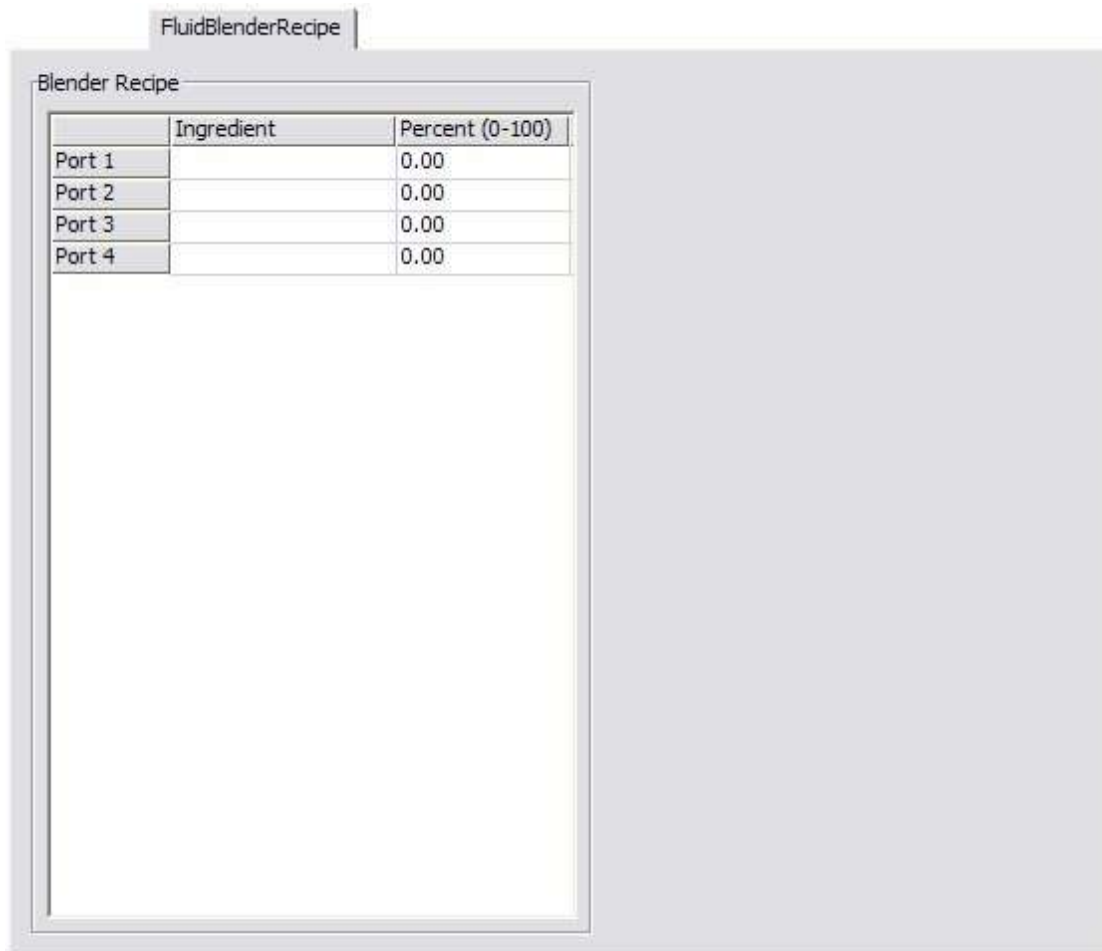
**Output port scale factor** - Each row of the table is the scale factor for a single output port. The value is multiplied by the maximum port rate to determine the actual maximum rate for that specific port. This allows the modeller to change the rate of individual output ports during a model run.

## This Page is Used By

FluidBlender



## FluidBlenderRecipe Tab Page



### Blender Recipe

Each row of this table represents a single input port. The rows do not appear in the table unless the object is already connected to upstream objects when the Parameters GUI is opened. There are two columns that the modeller can change in the table:

**Ingredient** - This is a text description of the ingredient coming from the port the row represents. This is for the modeller's use only, the Blender will ignore this value.

**Percent** - This is a number between 0 and 100 that is the percentage of the total incoming material that should come from the port represented by the row. The Blender will adjust the actual amount of material pulled from each port to make sure these percentages are correct, even when there is not enough material or space available to pull at the maximum rate.

### This Page is Used By

FluidBlender

## FluidGenerator Tab Page

The screenshot shows the FluidGenerator GUI with the following components:

- Maximum Content:** A text box containing "100.00".
- Initial Content:** A text box containing "100.00".
- Initial Product:** A button.
- Generator Refill:** A section containing:
  - Refill Mode:** A dropdown menu set to "Continuous Refill".
  - Refill Rate:** A text box containing "1.00".
  - Delay Time:** A text box containing "0.00".
- Output Ports:** A section containing:
  - Maximum Object Rate:** A text box containing "1.00".
  - Maximum Port Rate:** A text box containing "1.00".
  - Output port scale factor (0-1):** A label above a large empty rectangular area.
- Adjust Output Rates:** A section at the bottom containing a dropdown menu set to "Do nothing", a list icon, and a button labeled "A".

**Maximum Content** - The maximum amount of fluid material that this object can hold at any time.

**Initial Content** - The amount of material that is in the object when the model is reset.

**Initial Product** - This opens the Initial Product GUI which allows the modeller to define the Product ID and sub-component mix of the material that is in this object.

**Adjust Output Rates** - This function is called every tick and allows the user to change the output rates and scale factors during a model run. There are access variables that are references to the nodes that hold the rates and scale factors. To read the current rate (object or port) the `getnodenum()` command should be used. To change the rate the `setnodenum()` should be used. To read or change the scale factors `getnodenum()` and `setnodenum()` should be used in conjunction with the `rank()` command.

### Generator Refill

These parameters affect how the object refills itself as the model runs.

**Refill Mode** - This selects the type of refill the Generator performs. It can refill itself continuously (at a specified rate) or it can refill itself completely after it becomes empty.

**Refill Rate** - The rate at which the Generator refills itself. This is available if Continuous Refill is selected in the Refill Mode drop-down list.

**Delay Time** - The time that the Generator waits after becoming empty before it completely refills itself. This is available if Refill When Empty is selected in the Refill Mode drop-down list.

## Output Ports

These parameters affect how the object sends material to downstream objects.

**Maximum Object Rate** - The maximum rate that material will leave this object through all output ports combined.

**Maximum Port Rate** - The maximum rate that material will leave this object through any one port.

**Output port scale factor** - Each row of the table is the scale factor for a single output port. The value is multiplied by the maximum port rate to determine the actual maximum rate for that specific port. This allows the modeller to change the rate of individual output ports during a model run.

## This Page is Used By

FluidGenerator



## FluidLevelDisplay Tab Page

FluidLevelDisplay

☒ Draw Level Indicator

☒ Rectangular ☐ Cylindrical

X	0.00	RX	0.00	SX	0.10
Y	0.00	RY	0.00	SY	0.10
Z	0.00	RZ	0.00	SZ	1.00

**Draw Level Indicator** - If this is checked, the level indicator bar will be drawn on the object.

**Rectangular** - If this is selected, the level indicator bar will be drawn as a colored box.

**Cylindrical** - If this is selected, the level indicator bar will be drawn as a colored cylinder.

**X** - The X location of the bar.

**Y** - The Y location of the bar.

**Z** - The Z location of the bar.

**RX** - The rotation of the bar around the X axis.

**RY** - The rotation of the bar around the Y axis.

**RZ** - The rotation of the bar around the Z axis.

**SX** - The size of the bar in the X direction.

**SY** - The size of the bar in the Y direction.

**SZ** - The size of the bar in the Z direction.

**Note:** The location and size values are expressed as a percentage (0-1) of the size of the object.

**This Page is Used By**

## Flexsim User Guide

FluidBlender  
FluidGenerator  
FluidMixer  
FluidProcessor  
FluidSplitter  
FluidTank  
FluidToItem  
ItemToFluid

---

## FluidMixer Tab Page

FluidMixer

Target Product ID

**Input Ports**

Maximum Object Rate

Maximum Port Rate




**Output Ports**

Maximum Object Rate

Maximum Port Rate

Output port scale factor (0-1)


Adjust Output Rates

   A]

**Target Product ID** - The ProductID that will be assigned to the material that leaves this object.

**Adjust Output Rates** - This function is called every tick and allows the user to change the output rates and scale factors during a model run. There are access variables that are references to the nodes that hold the rates and scale factors. To read the current rate (object or port) the `getnodenum()` command should be used. To change the rate the `setnodenum()` should be used. To read or change the scale factors `getnodenum()` and `setnodenum()` should be used in conjunction with the `rank()` command.

### Input Ports

These parameters affect how the object receives material from upstream objects.

**Maximum Object Rate** - The maximum rate that material will be allowed into this object through all input ports combined. The actual input rate is based on the amount of material available upstream and the space available in this object.

**Maximum Port Rate** - The maximum rate that material will be allowed into this object through any single input port.

## Output Ports

These parameters affect how the object sends material to downstream objects.

**Maximum Object Rate** - The maximum rate that material will leave this object through all output ports combined.

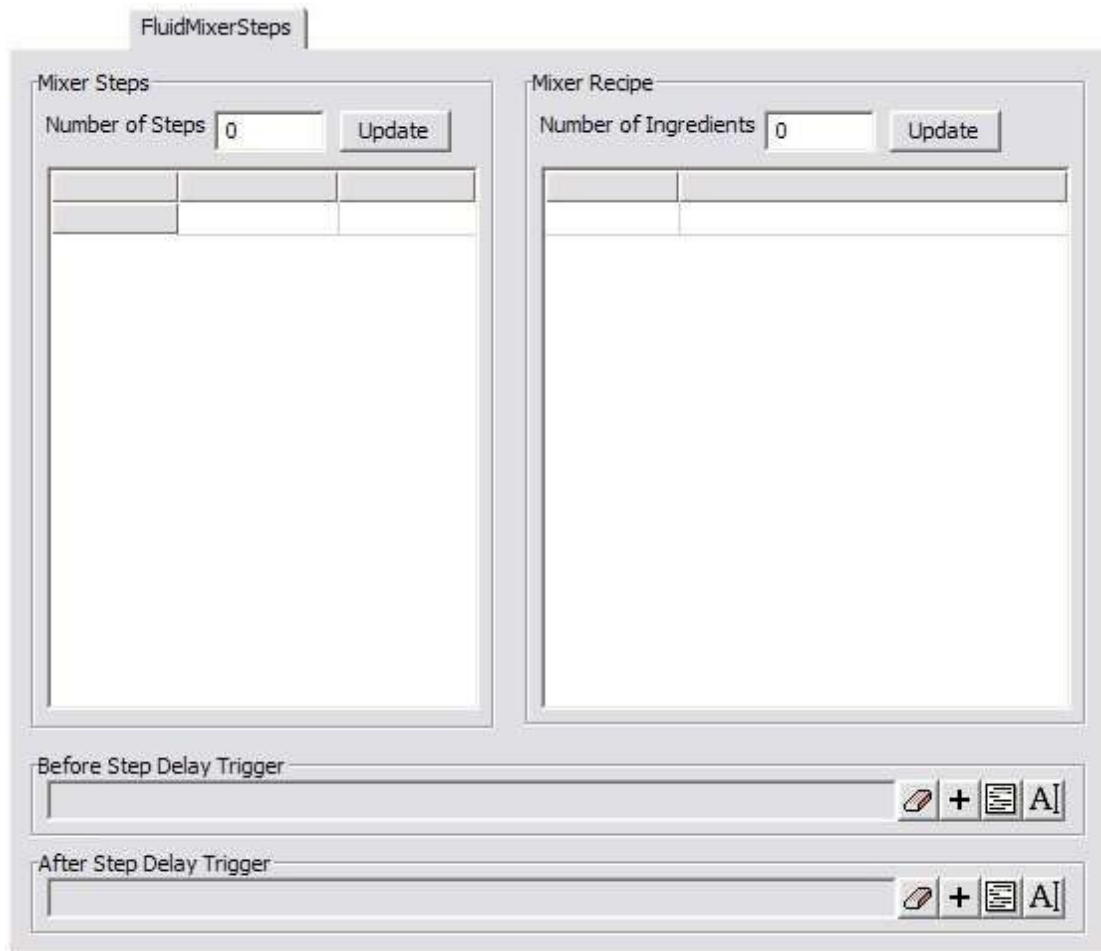
**Maximum Port Rate** - The maximum rate that material will leave this object through any one port.

**Output port scale factor** - Each row of the table is the scale factor for a single output port. The value is multiplied by the maximum port rate to determine the actual maximum rate for that specific port. This allows the modeller to change the rate of individual output ports during a model run.

## This Page is Used By

FluidMixer

## FluidMixerSteps Tab Page



The screenshot shows a software window titled "FluidMixerSteps". It is divided into two main panels. The left panel, titled "Mixer Steps", contains a "Number of Steps" input field with the value "0" and an "Update" button. Below this is a large, empty table with a header row and several data rows. The right panel, titled "Mixer Recipe", contains a "Number of Ingredients" input field with the value "0" and an "Update" button. Below this is another large, empty table with a header row and several data rows. At the bottom of the window, there are two text input fields. The first is labeled "Before Step Delay Trigger" and the second is labeled "After Step Delay Trigger". To the right of each text field is a set of four icons: a pencil, a plus sign, a document icon, and a keyboard icon.

**Before Step Delay Trigger** - This trigger fires after all of the material for a step has been collected, but before the step's delay time begins. This gives the modeller a chance to do things like call an operator for the delay.

**After Step Delay Trigger** - This trigger fires after the delay for a step is complete. It gives the modeller a chance release an operator or send messages to other objects.

### Mixer Steps

**Number of Steps** - This is the number of steps that the Mixer will go through for every batch of material that it makes.

**Update** - Pressing this button updates the Step Table so that it has the number of rows specified by the modeller.

The Step Table shows all of the steps that the Mixer must go through for each batch. Each step has two columns that the modeller must fill out:

**Description** - This is a text description of the step. It is displayed by the Mixer's name in the model view window when the Mixer is on the step.

**Delay** - This is the amount of time that the Mixer must wait after collected all of the ingredients for the step before it can go on to the next step.

## Mixer Recipe

**Number of Ingredients** - This is the number of ingredients that the Mixer will pull as it goes through its Step Table.

**Update** - Pressing this button updates the Ingredients Table so that it has the number of rows specified by the modeller.

The Ingredients Table show all of the ingredients that the Mixer pulls as it goes through its Step Table. If a single ingredient needs to be pulled in more than one step, it should appear in more than one row in the table. The table has four columns that the modeller must fill out:

**Ingredient** - This is a text description of the ingredient that the row represents. It is only to help the modeller document their model. It does not affect the Mixer's behavior.

**Port** - This is the input port that the ingredient will be pulled from.

**Amount** - This is the amount of the ingredient that will be pulled.

**Step** - This is the step number that the Mixer must be in for this ingredient to be pulled.

## This Page is Used By

FluidMixer

## FluidPipe Tab Page

FluidPipe

Maximum Content: 100.00      Flow Mode: User-Defined

**Input Ports**

Maximum Flow Rate: 1.00

Maximum Port Rate: 1.00

Input port scale factor (0-1)



**Output Ports**

Maximum Port Rate: 1.00

Output port scale factor (0-1)



Adjust Input Rates: Do nothing

Adjust Output Rates: Do nothing

**Maximum Content** - The maximum amount of material that this object can hold.

**Flow Mode** - The Pipe has three different modes that can be used to define how fluid is sent downstream:

**Flow Evenly** - The output ports are configured to have a maximum flow rate equal to the incoming flow rate divided by the number of output ports. The output ports may not send the same amount, depending on the content of the downstream objects

**First Available** - The output ports are configured to have a maximum flow rate equal to the incoming flow rate. Material will be sent to downstream objects in a first-come-first-served manner.

**User Defined** - The modeller has control over the input rate (both for the object and the individual ports) and the output rate for individual ports.

**Adjust Input Rates** - This function is called every tick and allows the user to change the input rates and scale factors during a model run. There are access variables that

are references to the nodes that hold the rates and scale factors. To read the current rate (object or port) the `getnodenum()` command should be used. To change the rate the `setnodenum()` should be used. To read or change the scale factors, `getnodenum()` and `setnodenum()` should be used in conjunction with the `rank()` command.

**Adjust Output Rates** - This function is called every tick and allows the user to change the output rates and scale factors during a model run. There are access variables that are references to the nodes that hold the rates and scale factors. To read the current rate (object or port) the `getnodenum()` command should be used. To change the rate the `setnodenum()` should be used. To read or change the scale factors `getnodenum()` and `setnodenum()` should be used in conjunction with the `rank()` command.

## Input Ports

These parameters affect how the object receives material from upstream objects.

**Maximum Flow Rate** - The maximum rate that material will be allowed into this object through all input ports combined. This value serves as both the maximum input and maximum output rates. The actual rate is based on the amount of material available upstream and the space available in this object. Material will attempt to leave the Pipe at the same rate that it came in. If there is not enough room downstream, the material will "back up" and more (up to the maximum rate) will be available to send in the next tick.

**Maximum Port Rate** - The maximum rate that material will be allowed into this object through any single input port.

**Input port scale factor** - Each row of the table is the scale factor for a single input port. The value is multiplied by the maximum port rate to determine the actual maximum rate for that specific port. This allows the modeller to change the rate of individual output ports during a model run.

## Output Ports

These parameters affect how the object sends material to downstream objects.

**Maximum Port Rate** - The maximum rate that material will leave this object through any one port.

**Output port scale factor** - Each row of the table is the scale factor for a single output port. The value is multiplied by the maximum port rate to determine the actual maximum rate for that specific port. This allows the modeller to change the rate of individual output ports during a model run.

## This Page is Used By

FluidPipe



## FluidPipeLayout Tab Page

FluidPipeLayout

Pipe Section Edit Table

Initial Z Rotation  Number of Sections   ☐ Conveyor View

	length	diameter	zrotation	yrotation	showjoint	
section1	2.00	0.20	30.00	0.00	1.00	
section2	2.00	0.20	-30.00	45.00	1.00	
section3	2.00	0.20	0.00	0.00	1.00	

### Pipe Section Edit Table

**Initial Z Rotation** - This is the rotation around the Z axis that is applied to the starting point of the Pipe. This is used to orient the Pipe in a particular direction before sections are drawn.

**Number of Sections** - The number of individual sections in Pipe's display.

**Refresh** - Pressing this button updates the table so that it has the number of rows the modeller specified.

**Conveyor View** - If this box is checked, the Pipe will be drawn as a simple conveyor.

**Section Table** - Each row in the table represents a single section of the Pipe. There are five columns the modeller should fill out:

**Length** - The length of the section

**Diameter** - The diameter of this section of the Pipe. If the Pipe is being shown as a conveyor, this is the width of the end of the section. The actual section will become wider or more narrow depending on the diameter value of the previous section.

**Z Rotation** - The rotation around the Z Axis that is applied at the end of the section.

**Y Rotation** - The rotation around the Y Axis that is applied at the end of the section.

**Show Joint** - If this is 1, a pipe joint will be drawn between the end of this section and the start of the next one. The size of the joint is based on the diameter of this section. This value is ignored if the Pipe is being drawn as a Conveyor.

### This Page is Used By

FluidPipe

## FluidProcessor Tab Page

FluidProcessor

Maximum Content  Loss Value

Input Ports  
No input information required

Output Ports  
Maximum Output Rate

Receive Port Number  
By Expression Input Port: 1 Note: The expression may be a constant value or the result of an expression

Destination Port Number  
By Expression Output Port: 1 Note: The expression may be a constant value or the result of an expression

**Maximum Content** - The maximum amount of material that this object can hold.

**Loss Amount** - A value between 0 and 1 that represents the percentage of material that is lost going through the Processor. This loss could be due to evaporation, inefficiency or many other factors. A value of 0 means that there is no material lost, a value of 1 means that all material is lost. This loss is applied as soon as material is pulled into the Processor.

**Receive Port Number** - If this field returns a 0, the Processor will receive material from all input ports. If it returns a number greater than zero, the Processor will only receive material from that input port.

**Destination Port Number** - If this field returns a 0, the Processor will allow material to leave from all of its output ports. If it returns a number greater than zero, the Processor will only allow material out that output port.

### Input Ports

There is no input information the modeller has to define.

## Output Ports

These parameters affect how the object sends material to downstream objects.

**Maximum Output Rate** - The maximum rate that material will leave this object through all of the output ports combined. The actual rate will be determined by the rate of material coming into the Processor.

## This Page is Used By

FluidProcessor

---



maximum rate for that specific port. This allows the modeller to change the rate of individual input ports during a model run.

## Output Ports

These parameters affect how the object sends material to downstream objects.

**Maximum Output Rate** - The maximum rate that material will leave this object through all output ports combined.

## This Page is Used By

FluidSplitter

---

## FluidSplitterPercents Tab Page

The screenshot shows a window titled 'FluidSplitterPercents' with a tab labeled 'Splitter Percents'. Inside the window is a table with the following data:

	Ingredient	Percent (0-100)
Port 1		0.00
Port 2		0.00
Port 3		0.00
Port 4		0.00

### Splitter Percents

Each row of this table represents a single output port. The rows do not appear in the table unless the object is already connected to downstream objects when the Parameters GUI is opened. There are two columns that the modeller can change in the table:

**Ingredient** - This is a text description of the material going to the port the row represents. This is for the modeller's use only, the Splitter will ignore this value.

**Percent** - This is a number between 0 and 100 that is the percentage of the total outgoing material that should go to the port represented by the row. The Splitter will adjust the actual amount of material sent to each port to make sure these percentages are correct, even when there is not enough material or space available to send at the maximum rate.

### This Page is Used By

FluidSplitter

## FluidTank Tab Page

The screenshot shows the 'FluidTank' configuration window. At the top, there are three fields: 'Maximum Content' (100.00), 'Initial Content' (0.00), and a button labeled 'Initial Product'. Below these are two main sections: 'Input Ports' and 'Output Ports'. Each section contains 'Maximum Object Rate' (1.00) and 'Maximum Port Rate' (1.00) fields, followed by a table for 'Input port scale factor (0-1)' and 'Output port scale factor (0-1)' respectively. At the bottom, there are two dropdown menus for 'Adjust Input Rates' and 'Adjust Output Rates', both currently set to 'Do nothing', with buttons for help and OK.

**Maximum Content** - The maximum amount of fluid material that this object can hold at any time.

**Initial Content** - The amount of material that is in the object when the model is reset.

**Initial Product** - This opens the Initial Product GUI which that allows the modeller to define the Product ID and sub-component mix of the material that is in this object.

**Adjust Input Rates** - This function is called every tick and allows the user to change the input rates and scale factors during a model run. There are access variables that are references to the nodes that hold the rates and scale factors. To read the current rate (object or port) the `getnodenum()` command should be used. To change the rate the `setnodenum()` should be used. To read or change the scale factors `getnodenum()` and `setnodenum()` should be used in conjunction with the `rank()` command.

**Adjust Output Rates** - This function is called every tick and allows the user to change the output rates and scale factors during a model run. There are access variables that are references to the nodes that hold the rates and scale factors. To read the current rate (object or port) the `getnodenum()` command should be used. To change



the rate the `setnodenum()` should be used. To read or change the scale factors `getnodenum()` and `setnodenum()` should be used in conjunction with the `rank()` command.

## Input Ports

These parameters affect how the object receives material from upstream objects.

**Maximum Object Rate** - The maximum rate that material will enter this object through all output ports combined.

**Maximum Port Rate** - The maximum rate that material will enter this object through any one port.

**Output port scale factor** - Each row of the table is the scale factor for a single input port. The value is multiplied by the maximum port rate to determine the actual maximum rate for that specific port. This allows the modeller to change the rate of individual input ports during a model run.

## Output Ports

These parameters affect how the object sends material to downstream objects.

**Maximum Object Rate** - The maximum rate that material will leave this object through all output ports combined.

**Maximum Port Rate** - The maximum rate that material will leave this object through any one port.

**Output port scale factor** - Each row of the table is the scale factor for a single output port. The value is multiplied by the maximum port rate to determine the actual maximum rate for that specific port. This allows the modeller to change the rate of individual output ports during a model run.

## This Page is Used By

FluidTank

## FluidTankMarks Tab Page

**Low Mark** - If the content passes this value (while rising or falling), the PassingLowMark Trigger will fire.

**Mid Mark** - If the content passes this value (while rising or falling), the PassingMidMark Trigger will fire.

**High Mark** - If the content passes this value (while rising or falling), the PassingHighMark Trigger will fire.

**PassingLowMark** - If the content passes the Low Mark, this trigger fires. Its common uses include opening and closing ports or sending messages. There is an access variable that informs the modeller if the fluid level is rising through the mark or falling.

**PassingMidMark** - If the content passes the Mid Mark, this trigger fires. Its common uses include opening and closing ports or sending messages. There is an access variable that informs the modeller if the fluid level is rising through the mark or falling.

**PassingHighMark** - If the content passes the High Mark, this trigger fires. Its common uses include opening and closing ports or sending messages. There is an access variable that informs the modeller if the fluid level is rising through the mark or falling.

### This Page is Used By

FluidTank

## FluidTerminator Tab Page

FluidTerminator

Input Ports

Maximum Object Rate 1.00

Maximum Port Rate 1.00

Input port scale factor (0-1)


Adjust Input Rates

Do nothing

**Adjust Input Rates** - This function is called every tick and allows the user to change the input rates and scale factors during a model run. There are access variables that are references to the nodes that hold the rates and scale factors. To read the current rate (object or port) the `getnodenum()` command should be used. To change the rate the `setnodenum()` should be used. To read or change the scale factors `getnodenum()` and `setnodenum()` should be used in conjunction with the `rank()` command.

### Input Ports

These parameters affect how the object receives material from upstream objects.

**Maximum Object Rate** - The maximum rate that material will enter this object through all output ports combined.

**Maximum Port Rate** - The maximum rate that material will enter this object through any one port.

**Output port scale factor** - Each row of the table is the scale factor for a single input port. The value is multiplied by the maximum port rate to determine the actual maximum rate for that specific port. This allows the modeller to change the rate of individual input ports during a model run.

### This Page is Used By

FluidTerminator

## FluidTicker Tab Page

The screenshot shows the FluidTicker configuration window. It has a title bar 'FluidTicker'. Inside, there are four main sections: 'Tick Time' with a text box containing '1.00'; 'Product Components' with a text box containing '0' and an 'Update' button; 'Optimize object list resorting' with a checked checkbox; and 'Component Names' with a large, empty rectangular table area.

**Tick Time** - This is length of time in each tick. At the end of a tick, the Ticker calculates how much fluid moved between the fluid objects in the model.

**Optimize object list resorting** - The Ticker keeps an internal list of the order that the Fluid Objects should be evaluated. If this box is not checked, the order in which certain objects are evaluated may be different in different runs of the model. This can cause a model to give different results, even if nothing in the model has actually changed. Typically, this box should be checked.

**Product Components** - This is the number of sub-components available to all of the fluid objects in the model. All of the objects use the same list of sub-components, although they do not have to specify a value greater than 0 for all of the components.

**Update** - Pressing this button updates the list of component names so that there are the number that the modeller specified.

**Component Names** - This table lists the names of the sub-components that are available to all of the fluid objects in the model.

## This Page is Used By

Ticker

## FluidToltem Tab Page

FluidToItem

Maximum Content

100.00

Flowitem

Textured Colored Box

Input Ports

Maximum Object Rate

1.00

Maximum Port Rate

1.00

Input port scale factor (0-1)

Flowitem Output

Fluid per Discrete Unit

1.00

Discrete Units per Flowitem

1.00

Flowitem ItemType

0.00

Flowitem Name

Product

Adjust Input Rates

Do nothing

**Maximum Content** - The maximum amount of fluid material that this object can hold at any time.

**Flowitem** - This is the class of flowitem that the FluidToItem will create.

**Adjust Input Rates** - This function is called every tick and allows the user to change the input rates and scale factors during a model run. There are access variables that are references to the nodes that hold the rates and scale factors. To read the current rate (object or port) the `getnodenum()` command should be used. To change the rate the `setnodenum()` should be used. To read or change the scale factors `getnodenum()` and `setnodenum()` should be used in conjunction with the `rank()` command.

## Input Ports

These parameters affect how the object receives material from upstream objects.

**Maximum Object Rate** - The maximum rate that material will enter this object through all output ports combined.

**Maximum Port Rate** - The maximum rate that material will enter this object through any one port.

**Output port scale factor** - Each row of the table is the scale factor for a single input port. The value is multiplied by the maximum port rate to determine the actual maximum rate for that specific port. This allows the modeller to change the rate of individual input ports during a model run.

## Flowitem Output

These parameters define when the FluidToItem creates a flowitem and some information that will be defined on the flowitem when it is created.

**Fluid per Discrete Unit** - This is the number of units of fluid material that are in a single discrete unit in the flowitem. For example: 5 gallons per can.

**Discrete Units per Flowitem** - This is the number of discrete units of material that are in each flowitem. For example: 10 cans per case, where a flowitem is a single case.

**Flowitem Itemtype** - This value will be assigned to the itemtype of the flowitems as they are created. It can be changed using the OnCreation or OnExit triggers.

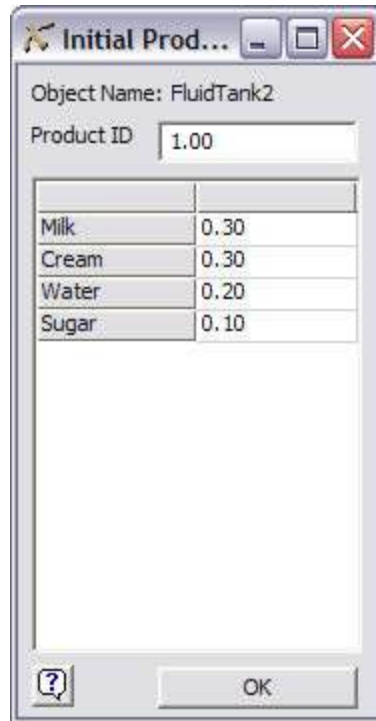
**Flowitem Name** - This name will be assigned to the flowitems that are created. It can be changed later using triggers.

## This Page is Used By

FluidToItem

---

## Initial Product GUI



This GUI is used to define the initial Product ID and sub-component list for fluid material that is created by the object.

**Object Name** - The object that is creating this material.

**Product ID** - The Product ID that will be assigned to the material that is created.

**Sub-Component List** - Each row in the list is a different sub-component that is available for this material. The list is defined on the FluidTicker GUI. The values are percentages from 0 to 1. They should add up to 1. All of the available sub-components are listed here, but the material does not have to use them all. If there are any that it does not use, the values in those rows should be set to 0.

### This Page is Used By

FluidGenerator  
FluidTank  
ItemToFluid

## ItemToFluid Tab Page

**Maximum Content** - The maximum amount of fluid material that this object can hold at any time.

**Initial Product** - This opens the Initial Product GUI which that allows the modeller to define the ProductID and sub-component mix of the material that is created by this object.

**Adjust Output Rates** - This function is called every tick and allows the user to change the output rates and scale factors during a model run. There are access variables that are references to the nodes that hold the rates and scale factors. To read the current rate (object or port) the `getnodenum()` command should be used. To change the rate the `setnodenum()` should be used. To read or change the scale factors `getnodenum()` and `setnodenum()` should be used in conjunction with the `rank()` command.

### Input Ports

**Fluid per Discrete Unit** - This is the number of units of fluid material that are in a single discrete unit in the flowitem. For example: 10 pounds per bag.

**Discrete Units per Flowitem** - This is the number of discrete units of material that are in each flowitem. For example: 5 bags per pallet, where a flowitem is a single pallet.



**Flowitem Recycling** - The modeller uses this drop-down list to decide where to store flowitems that need to be recycled. They should send flowitems back to the section of the flowitem bin that they originally came from.

## Output Ports

These parameters affect how the object sends material to downstream objects.

**Maximum Object Rate** - The maximum rate that material will leave this object through all output ports combined.

**Maximum Port Rate** - The maximum rate that material will leave this object through any one port.

**Output port scale factor** - Each row of the table is the scale factor for a single output port. The value is multiplied by the maximum port rate to determine the actual maximum rate for that specific port. This allows the modeller to change the rate of individual output ports during a model run.

## This Page is Used By

ItemToFluid

---

## Ticker



### Overview

The Ticker is responsible for breaking time into small, evenly spaced units called "ticks". The modeler can define the length of a tick. The Ticker is the object that controls all of the Fluid Objects in a model. For this reason, in any model that uses Fluid Objects, there should always be exactly one Ticker. The modeler also uses the Ticker to define the global list of sub-components that make up the fluid material in the model.

### Details

Any model that uses the Fluid Objects must have a Ticker. The Ticker must be named "TheTicker". The modeler can create a Ticker by dragging it from the library icon grid to the model view window. However, this is an easy step to forget, so when the modeler drags any Fluid Object into a model, a Ticker is created if there is not one already in the model. If the modeler tries to create another Ticker, a warning message is given saying that a Ticker already exists and that the new Ticker should be deleted.

The Ticker does not do much that the modeler can see, but it is very important behind-the-scenes. It is responsible for calculating how much material is transferred between Fluid Objects at the end of every tick. When the model is reset, the Ticker builds a list of all of the Fluid Objects in the model. These objects in this list are sorted based on how many fluid objects are upstream and downstream of them. At the end of each tick, the Ticker begins with the fluid object farthest downstream and calculates how much material it received during that tick. Then the Ticker calculates how much material moved into the object upstream of that starting object, and so on until it reaches the starting point of the fluid portion of the model.

The Ticker only has a few values that the modeler can change. The most important of these is the tick time. This is the length of time that the Ticker will wait between updates to the Fluid Objects. A very small tick time will result in a large number of events firing as the model runs (at least one per tick) and may slow down the model. It will also, in many cases, increase the accuracy of the Fluid Objects' behavior. A longer tick time will generally result in a faster running model, but at a cost in accuracy. It is up to the modeler to find the appropriate balance of speed and accuracy for their model.

The modeler can also use the Ticker to define a set of sub-components that will make up the fluid material in the model. Each of the sub-components should be given a name. These names are to help the modeler understand what is happening in the model, they have no effect on the objects' behavior or accuracy. All fluid material has the same list of sub-components. It is not required that any given fluid material use all the sub-components, however. The Fluid Objects keep track of the percentages of the sub-components that make up the material that they are currently

processing. If fluid material from two different sources is mixed together, the sub-components' percentages are adjusted accordingly. There is no limit to the number of sub-components that can be defined. Just remember that the list applies to all fluids.

### States

**Idle** - The Ticker is always in an idle state.

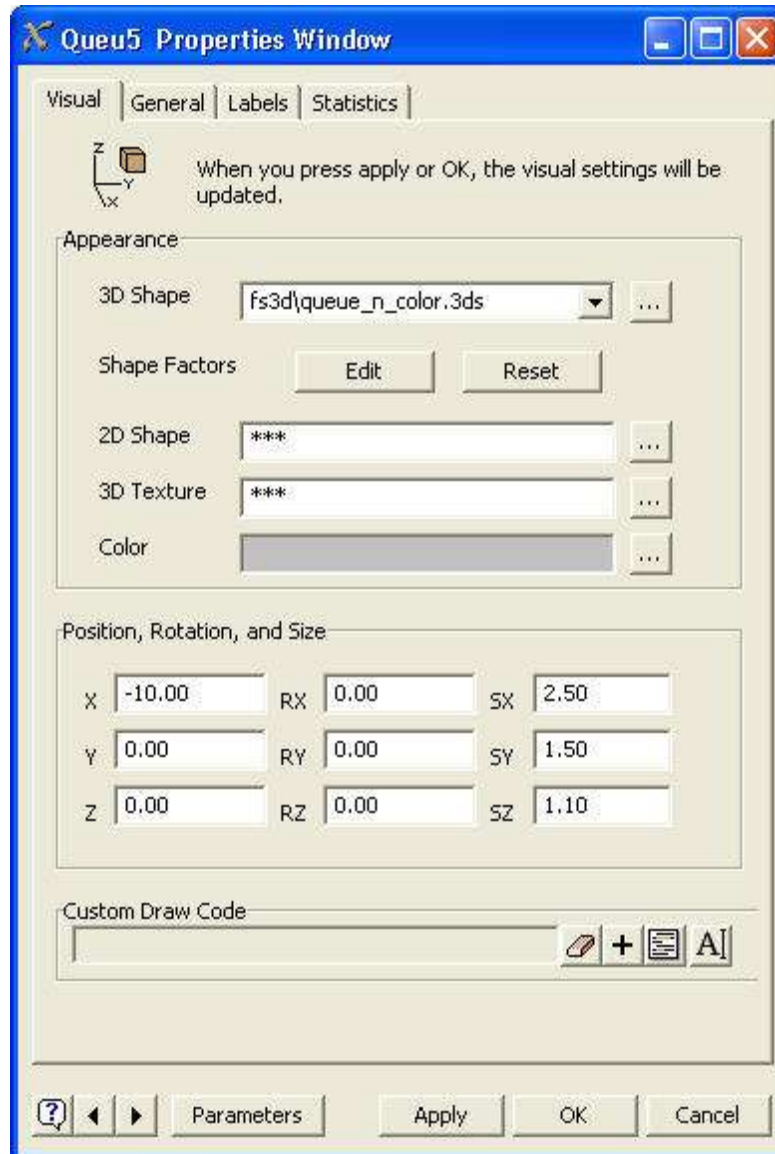
### Parameters **tab-pages**

FluidTicker

## Properties Pages

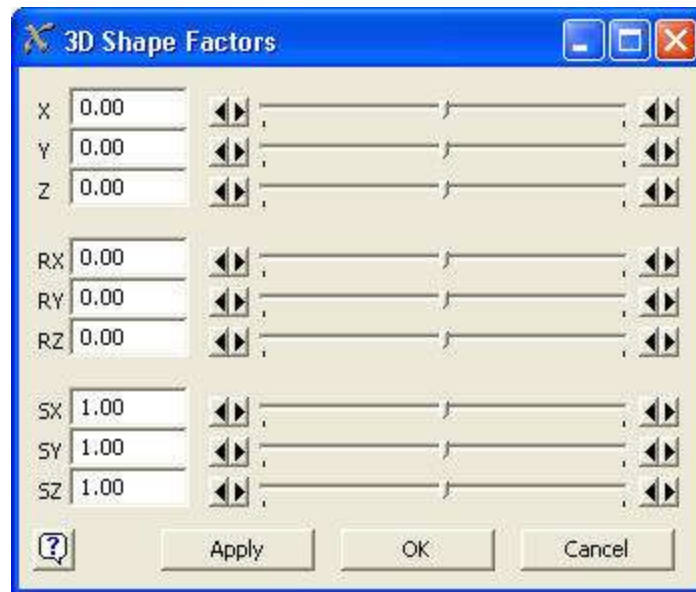
### Visual Properties Page

This page allows you to configure the appearance and orientation of the object.

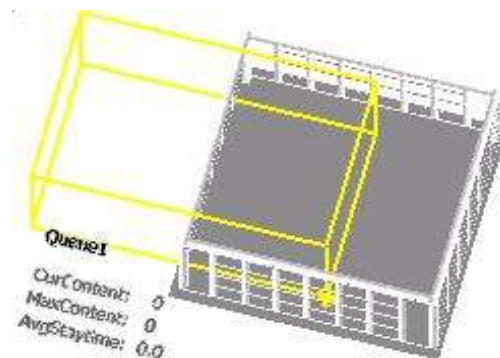


**3D Shape** - This option specifies the 3d shape for the object. Define a path to a .3ds, .wrl, .dxf, or .stl file. There is also a list of pre-selected options to choose from.

**Edit 3D Shape Factors** - Click on this button to bring up a window that edits the object's 3D shape factors. 3d shape factors specify how the object's 3d shape is oriented with respect to the object.



The window above shows the default shape factors of a Queue's 3d shape. Enter a value on the left, or use the slider to change the 3D shape factors. The buttons to the side of the slider let you change the minimum and maximum range of the tracker. The picture below shows a queue whose x shape factor has been changed from 0 to 0.5, y from 0 to -0.3, and whose sy shape factor has been changed from 1 to 1.5. Notice that the yellow bounding box still reflects the true position and size of the queue, but the 3d shape has been offset in the x direction and scaled in the y direction.



**2D Shape** - This option specifies the 2d shape for the object, or a texture to be drawn at its base. Usually the 2D texture is not shown in the orthographic view, but is shown in the planar view.

**3D Texture** - This field specifies the object's 3d texture. If the 3D shape does not already have a texture defined within its 3d file, then this texture will be drawn on the face of the 3d shape. Note that if the object's 3d shape already has a texture defined, then this texture will not be used.

**Color** - This field specifies the color of the object. Note that if the object already has materials defined in its 3d shape's file, then this color will not show. This color shows through only if no materials are defined in the 3d file.

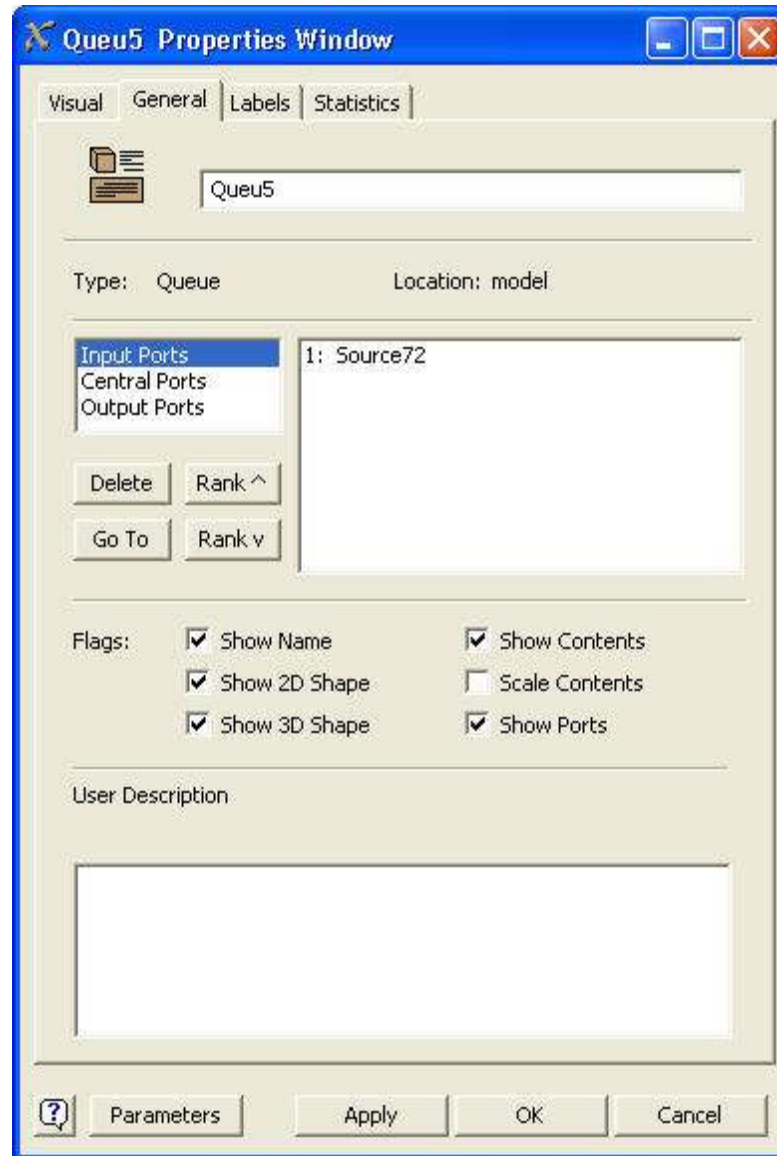
**Position, Rotation, and Size** - This option lets you define the position, rotation, and size of the object.

**Custom Draw Code** - This pick list allows you to define your own draw code for the object. If this field returns a 1, then the object's default draw code will not be drawn. Note that an object's draw code is different than its 3d shape being drawn. While most objects just show their 3d shape and don't have any draw code, some objects, like conveyors and racks, need more dynamic drawing capability, rather than a static 3d shape to draw. Returning 1 overrides this special drawing code, not the drawing of the object's 3d shape. To hide the 3d shape, un-check the show 3d shape box in the General Properties page.

---

## General Properties Page

This page mostly gives you information on the object, but you can also set the name of the object, as well as set flags for how the object is drawn.



**Name** - At the top of the page you can enter the name of the object. Be careful when having both Parameters and Properties windows open, because applying one could override the changes to another. Also, in specifying the name of the object, do not use any special characters like >, <, \*, -, (, ), etc. This will cause Flexsim to not compile correctly. Spaces and underscores should be the only non-alpha-numeric characters used. Also, do not begin the name with a number.

**Type** - This field shows what type of object it is.

**Location** - This field shows the name of the object containing this object.

**Input/Central/Output Ports** - This area lets you edit the object's connections. Select either Input Ports, Central Ports, or Output Ports from the combobox on the left. The list on the right shows the appropriate connections. Click on a connection and move it up or down in the list by using the Move Up and Move Down buttons. Delete the connection with the Delete button. Once you have finished editing an object's connections, you will need to reset the model before running it again.

**Flags** - Here you can check different boxes to show/hide different parts of the object, such as the contents of the object, the name, the ports, etc.

**User Description** - This field allows you to give a description of what the object does, including any special logic you have put on the object.

---



## Labels Tab Page

Labels are custom variables that you can specify on the object. For example, if you want to keep track of the number of flowitems of itemtype 3 that have entered an object, you can use a label to keep a record of this value. Use the commands `setlabelnum()`, `getlabelnum()`, and `label()` to interact with labels you have created. More information on these commands is found in the command summary.



The main panel shows a list of the labels on this object. You can add number and text labels by clicking on the Add Number/Text Label buttons at the bottom of the window. You can also edit labels by right clicking on a label. A pop-up menu will appear, giving you the options below. Make sure that you only right click on the label to get this pop-up menu. If you left click on a label, then right click, you will get a text editing menu instead.

You can also view the labels as a tree view by selecting Tree View in the drop-down box at the bottom.

**Note on deleting a label:** You can only delete a label by right-clicking on it and selecting the Delete Label option. You may need to click off of the table (in a blank white area of the panel), then right click on the label. Do not left click then right click on the label, or a different pop-up menu will appear.



**Add Number Label** - This option adds a number label to the object, the same as the Add Number Label button at the bottom.

**Add Text Label** - This option adds a text label to the object, the same as the Add Text Label button at the bottom.

**Delete Label** - This option deletes the selected label.

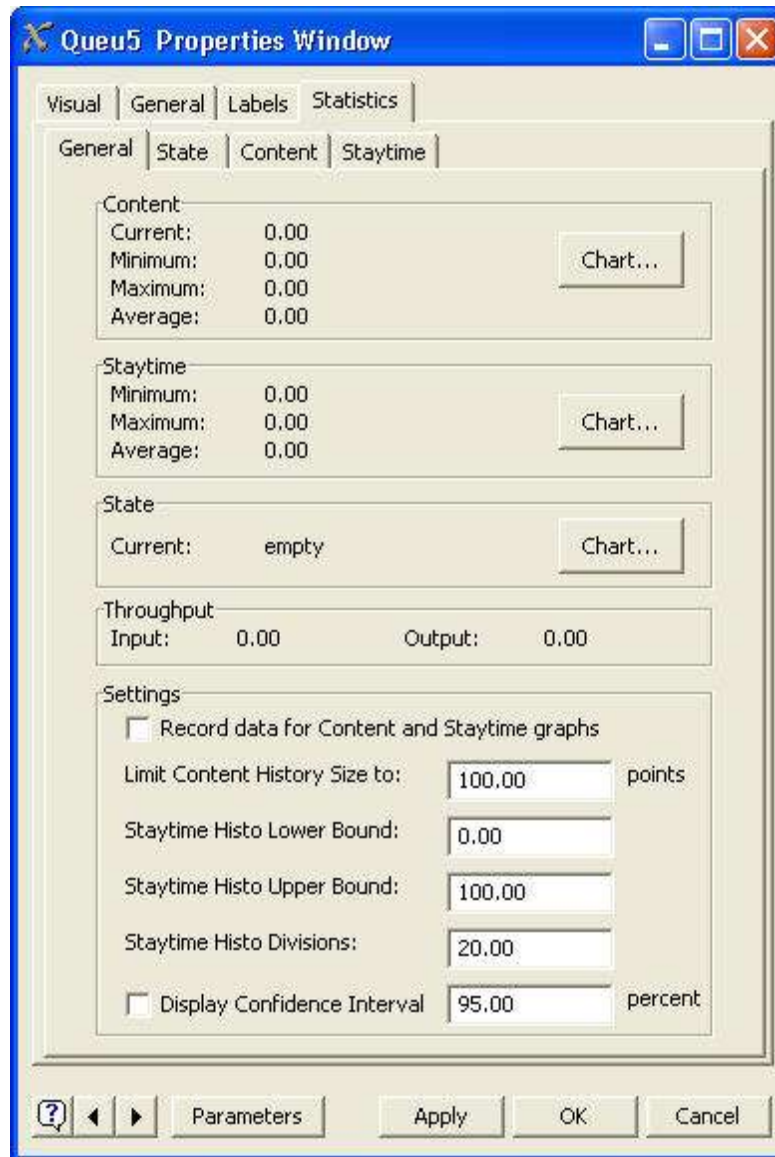
**Duplicate Label** - This option duplicates the selected label.

**Create/Edit Label Table** - This option lets you use a label as a two-dimensional table. It brings up a table edit window to edit the label as a table. To get and set values in this table during the model run, you can use the `gettablenum()` and `settablenum()` commands, passing a reference to the label as the first parameter using the `label()` command. For more information on these commands, refer to the command summary. Example: `gettablenum(label(current, "curitemtype"), 4, 5);`

**Explore as Tree:** - This option lets you explore the selected label in a tree view.

## Statistics Properties Page

The statistics page lets you see state, content, and staytime information on an object. This interface is separated into four sections. The first section shows general information on content, staytime, and state. The other three sections show graphs of the information.



**Content** - This box shows the object's current, minimum, maximum, and average content values. The average content is weighted by time and not by the number of instances of a given content. The Chart button on the right brings up a window that shows the object's content vs. time graph. The chart is the same as the one shown in the content tab page. The chart is shown below.

**Note on the Content Graph:** In order for an object's content graph to be recorded, the stats collecting of the object must be turned on for that object. For more information on stats collecting, refer to the Stats menu.

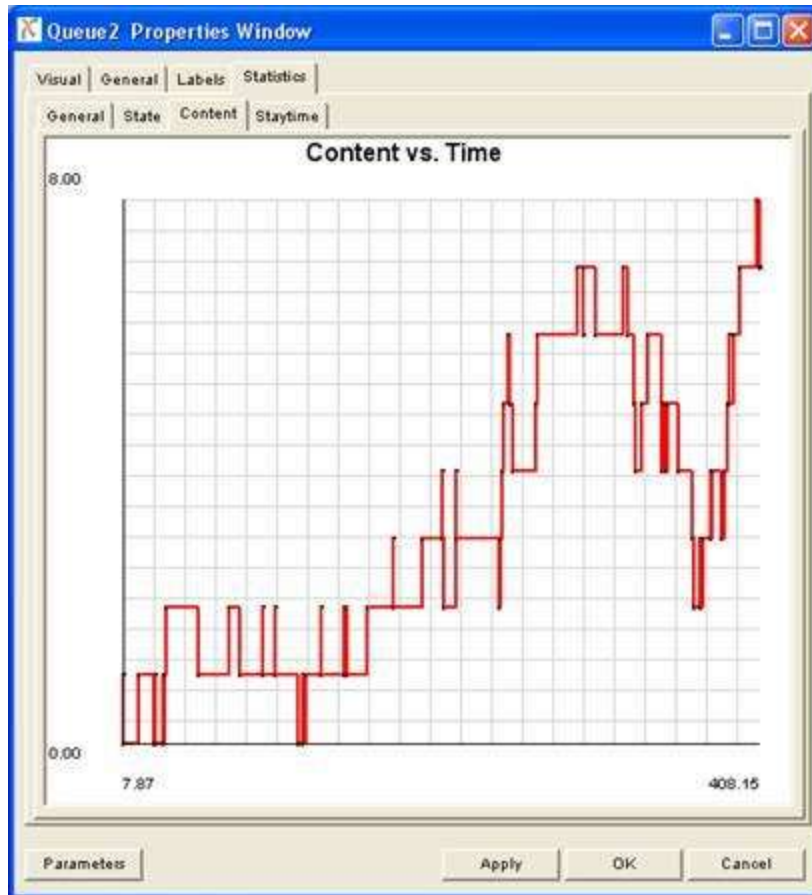
**Staytime** - This box shows the object's minimum, maximum, and average staytime values. The Chart button on the right brings up a window that shows the object's staytime histogram. The chart is the same as the one shown in the staytime tab page. This page is shown below.

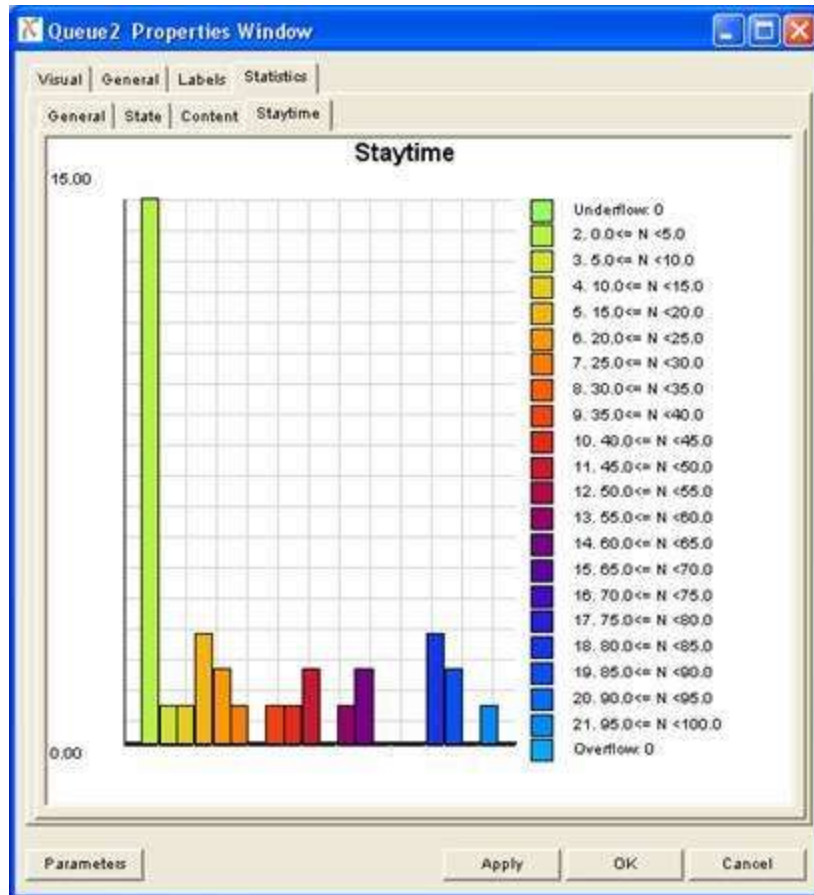
**Note on the Staytime Histogram:** In order for an object's staytime histogram to be recorded, the stats collecting of the object must be turned on for that object. For more information on stats collecting, refer to the Stats menu.

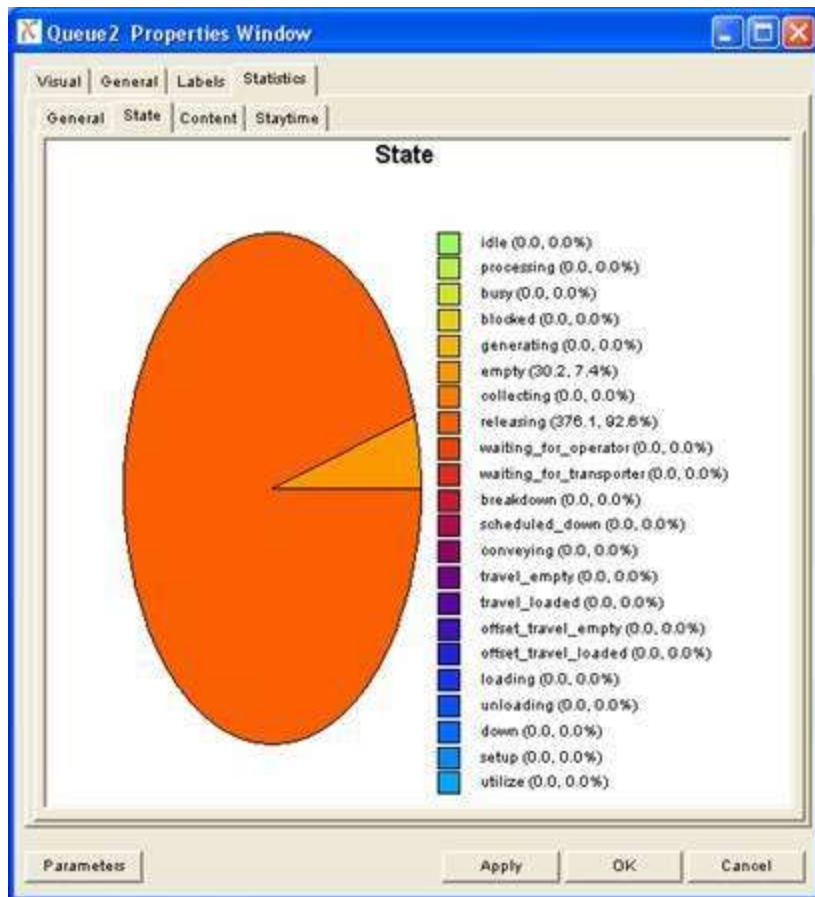
**State** - This box shows the object's current state value. Each state's meaning is dependent on the type of object involved. Refer to the library objects for more information about what each state means. Refer to the state list for a quick reference of each state's number and macro definition. The Chart button on the right brings up a window that shows the object's staytime histogram. The chart is the same as the one shown in the state tab page. This page is shown below.

**Throughput** - The throughput box shows the object's current input and output statistics.

**Settings** - Here you can define parameters for how statistics are recorded, such as the limit on the number of points stored in the content history, the lower and upper bounds on the staytime histogram, number of divisions or "buckets" in the staytime histogram, and whether to display the confidence interval for the mean staytime. Options for confidence percentage values are either 90, 95, or 99 percent confidence. Once you have changed these settings, you will need to reset the model before running again.











# Flexsim Object Library

## Flexsim Object Library

### Introduction

The Flexsim library is made up of objects that are designed to interact with each other in a way that is easy to use and understand. These objects have been implemented using an object-oriented methodology, which involves a super-class/subclass hierarchy. Subclass objects inherit attributes and default behavior from their super-classes while specializing that behavior to fit a specific situation. In Flexsim, most objects in the library have been created as one of two general object types, or super-classes. These two general types we refer to as *FixedResources* and *TaskExecuters*.

### FixedResources

FixedResources are stationary objects in your model that may represent steps in your process, such as processing stations or storage areas. Flowitems progress through the model by entering, being processed by, and then finishing at these steps in the model. When a flowitem is finished at one step, it is sent on to the next step, or FixedResource, in the model.

### TaskExecuters

TaskExecuters are used as shared, mobile resources in the model. They may be operators that are required in order for a given step to process a flowitem, or they may transport flowitems between steps. They can perform many other simulation functions as well.

As you become more experienced in using Flexsim, you will realize that the distinction between FixedResources and TaskExecuters can sometimes become very blurred. TaskExecuters have the capability of simulating FixedResource-like processing steps in a model, while FixedResources can also be configured to travel or operate as shared resources. The only difference is the perspective from which you approach the problem.

### Learning Suggestions

In getting to know the Flexsim object library, we suggest that you first read the help section for the FixedResource. Then read the help section for the TaskExecuter, as well as task sequence help. Once you are familiar with how these two general types of objects work, you can learn the specialized functionality for subclasses of these two general types. These subclasses are listed below.

### FixedResources

- Source
- Queue
- Processor
- Sink
- Conveyor
- Combiner
- Separator

FlowNode  
MergeSort  
MultiProcessor  
Rack  
Reservoir  
BasicFR

## **TaskExecuters**

Operator  
Transporter  
ASRSvehicle  
Crane  
Elevator  
Robot  
BasicTE

## **Other**

There are also some objects in the library that are neither TaskExecuter nor FixedResource. These object are listed below.

Dispatcher  
NetworkNode  
Recorder  
TrafficControl  
VisualTool

## ASRSvehicle



### Overview

The ASRSvehicle is a special type of transport specifically designed to work with racks. The ASRSvehicle will slide back and forth in an aisle between two racks picking up and dropping off flowitems. The reach, lift, and travel motions are fully animated by the ASRSvehicle. The lift and travel motions will occur simultaneously, but the reach will only occur after the vehicle has come to a complete stop.

### Details

The ASRSvehicle is a subclass of the TaskExecuter. It implements offset travel by only travelling along its own x-axis. It travels until it is perpendicular with the destination location, lifting its platform as well. If the offset travel is for a load or unload task, then once the offsetting is finished, it will use the user-specified load/unload time to convey the flowitem onto its platform, or off of its platform to the destination location.

The ASRSvehicle does not connect itself to a navigator by default. This means that travel tasks will not be performed. Instead, all traveling is done using offset travel.

**Note on conveying a flowitem onto an ASRSvehicle:** For a load task, the conveying of the item onto the platform may not work if the flowitem is in an object that continuously refreshes the location of the flowitem, like a conveyor for example. In this case, if you want the conveying of the item onto the platform to show up, then make sure that the ASRSvehicle is ranked after the object it is picking up from in the model tree (the ASRSvehicle must be lower down in the tree).

In addition to the standard TaskExecuter fields, the ASRSvehicle has a modeler-defined lift speed and initial lift position for the its platform. The platform will return to this position whenever the ASRSvehicle is idle or is not doing offset travel.

### Context

Since the main distinction of an ASRSvehicle is that it only moves along its x and z axes and doesn't rotate, this object can be used for any purpose in which you don't want the object to turn, but rather just go forward and backward and up and down. In some models it has been used as a simple transfer car, or as a transfer conveyor between two or more conveyors.

### States

This object follows TaskExecuter states.

### Parameter tab-pages

TaskExecuter  
Collision  
Dispatcher

TaskExecutorTriggers  
ASRSvehicle

## Related Topics

TaskExecutor  
Task Sequences

## BasicFR



### Overview

The BasicFR object is a FixedResource that is meant for developers to create user libraries with. It passes almost all inheritable FixedResource logic to pick-list functions, so that user library developers can specify virtually all functionality for the FixedResource.

### Details

The BasicFR is a sub-class of the FixedResource. It allows you to specify logic for its reset, entry, exit, and message triggers, as well as advanced functionality like stop object/resume object, pick/place offset, transport in notify/complete, transport out notify/complete, and other advanced functions.

In the entry, exit, reset, and message triggers of this object, you will need to implement logic that receives and releases flowitems using the `receiveitem()` and `releaseitem()` commands. There are also several more commands that you can use in processing items, such as `setstate()`, `senddelayedmessage()`, and all of the commands in the FixedResource category of the command list. This object is meant to be a bare-bones implementation of the FixedResource, where all logic is implemented by the modeler.

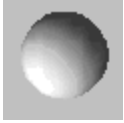
### Extra Parameters for the Entry/Exit Triggers

The BasicFR passes 2 extra parameters into its entry and exit triggers. As `parval(3)` it passes the current value of its `nrofrtransportsin` variable. As `parval(4)` it passes the current value of its `nrofrtransportsout` variable.

### Parameters tab-pages

- Flow
- FixedResourceTriggers
- Labels
- BasicFR Advanced

## BasicTE



### Overview

The BasicTE object is a TaskExecutor that is meant for developers to create user libraries with. It passes almost all inheritable TaskExecutor logic to pick list functions, so that user library developers can specify virtually all functionality for the TaskExecutor.

### Details

The BasicTE is a sub-class of the TaskExecutor. It allows you to specify logic for its offset travel functionality, as well as advanced functionality like stop object/resume object, pick/place offset, and other advanced functions. For more information, refer to the BasicTE Page.

### Parameters tab-pages

- BasicTE
- TaskExecutor
- Collision
- Dispatcher
- TaskExecutorTriggers

## Combiner



### Overview

The combiner is used to group multiple flowitems together as they travel through the model. It can either join the flowitems together permanently, or it can pack them so that they can be separated at a later point in time. The combiner will first accept a single flowitem through input port number 1 before it will accept the subsequent flowitems through the remaining input ports. The user specifies the quantity of subsequent flowitems to accept through input ports 2 and higher. Only after all subsequent flowitems required by the user have arrived will the setup/process time begin. The combiner can be set to require operators during its setup, processing and repair times.

### Details

The Combiner is a sub-class of the Processor, which is in turn a sub-class of the FixedResource. During operation, the Combiner first receives exactly one flowitem from its first input port. It will wait until a flowitem has arrived through input port 1 before it allows other flowitems in. Then it collects a batch of flowitems using its component list. The component list specifies the number of flowitems the Combiner should receive from each other input port for each batch.. Row 1 of the component list corresponds to the number of flow items that should be received from input port 2. Row 2 corresponds to input port 3, and so forth. The component list is updated automatically when you connect objects to its input ports. If you have the Combiner's parameters window open when you add an input port, you will need to close the window and double-click on the Combiner again to register the changes.

Once the Combiner has collected a batch, it goes through a setup and Process time, calling operators to the setup and process operations as defined by the Processor functionality.

The Combiner can operate in one of three modes: pack, join, or batch. In pack mode the Combiner moves all flowitems that were received through ports 2 and higher into the flowitem received through input port 1 and then releases this container flowitem. In join mode the Combiner destroys all flowitems except the one received through input port 1. In batch mode, the Combiner simply releases all the flow items once the batch is collected and the setup and process times have finished.

The Pull from Port field is not used for the Combiner. The Combiner handles this logic on its own.

If you are transporting flowitems into the Combiner, then while it is receiving the container flowitem, it will only allow one flowitem to be in transit to itself at a given time, namely the container flowitem. Once the container flowitem has arrived, the Combiner will allow all other flowitems for the components list to be in transit at the same time.

**Tips on receiving more than one flowitem through input port 1:** The Combiner is configured to always receive exactly one flowitem from input port 1. If you are in batch or join mode, you may want to receive more than one flowitem from the upstream object that is connected to input port 1. Here you can do one of two things. The simplest option is to connect the upstream object to both input ports 1 and 2 of the Combiner, then in the components list make the first row entry be one less than the number of flowitems you want to collect. The Combiner will receive one flowitem through input port 1 and then the remaining number you want from input port 2. If this doesn't work in your scenario, then the other option is to add a Source to your model, connect it to input port 1 of the Combiner, and give the Source a constant inter-arrival time of 0.

**Tip on receiving different types of flowitems from the same upstream object:** If you have a single upstream object that can hold many different types of flowitems, but you want to screen these different types separately in the Combiner's component list, you can do this by connecting several output ports of the upstream object to several input ports of the Combiner. For example, a Combiner receives item types 1 and 2 from a single upstream Processor. You want to collect 4 of item type 1, and 6 of item type 2, and pack them onto a pallet. To do this, first connect the pallet's source to input port 1 of the Combiner. Then connect the Processor's output port 1 to the Combiner's input port 2, and then connect the Processor's output port 2 to the Combiner's input port 3. Have the Processor's sendto strategy send by item type. Then in the Combiner's component list, enter a 4 in the row corresponding to input port 2, and a 6 corresponding to input port 3.

**Note on manually moving flowitems out of the combiner:** If you manually move the container flowitem out of the combiner, either using a task sequence or the moveobject command, make sure that you specify a non-zero port number for the item to exit through. When the combiner is packing, it moves packed flowitems out of itself into the container flowitem. This causes its exit trigger to fire, and the way that it distinguishes between a part moving into its container and the container exiting is by the port number of the exit trigger. If the port number is 0, it assumes it is a part being moved into the container, and does nothing. Thus, if you explicitly move the container flowitem out of the combiner, and the port number is 0, it will assume it is a packed flowitem, and will not receive the next container flowitem.

## States

**Idle** - The object has not received its first flowitem from input port 1.

**Collecting** - The object has received the first flowitem from input port 1, but is still collecting its remaining batch of flowitems.

**Setup** - The object is in its modeler-defined setup time.

**Processing** - The object is in its modeler-defined process time

**Blocked** - The object has released its flowitem(s), but downstream objects are not ready to receive them yet.

**Waiting for Operator** - The object is waiting for an operator to arrive, either to repair a breakdown, or to operate on a batch.

**Waiting for Transport** - The object has released a flow item, and a downstream object is ready to receive it, but a transport object has not picked it up yet.

**Down** - The object is broken down.



## Parameters tab-pages

- ProcessTimes
- Combiner
- Flow
- ProcessorTriggers
- Operators

## Conveyor



### Overview

The Conveyor is used to move flowitems through the model along a set path. The path is defined by creating different sections in the conveyor. Each section can be either straight or curved. Curves are defined by the angle that they turn and their radius. Straight sections are defined by their length. This allows the conveyor to have as many curves and bends as it requires. The conveyor can be either accumulating or non-accumulating.

### Details

The Conveyor is a sub-class of the FixedResource. It operates in one of two different modes, accumulating or non-accumulating. In accumulating mode, the conveyor acts like a roller conveyor. Flowitems can accumulate even when the end of the conveyor is blocked. In non-accumulating mode, the conveyor operates like a belt conveyor. If the conveyor is blocked then all flowitems on the conveyor will be stopped.

### Receive/Release Logic

When a flowitem arrives on the conveyor, it starts with its front edge at the starting edge of the conveyor. It begins to convey down the length of the conveyor. Once the flowitem's full length has conveyed over the start edge of the conveyor, the conveyor opens its inputs again to receive another product. When the flowitem's front edge hits the end of the conveyor, the conveyor releases the flowitem.

**Note on the first section being a curved section:** If the first section of the conveyor is a curved section, then entering flowitems will convey onto the conveyor as if the last section of the previous conveyor were also a curved section. Thus the flowitem will actually curve onto the conveyor. This may not be what you want. You may want the flowitem to flow straight onto the conveyor. To do this, instead of having the first section of the conveyor be a curved section, insert a straight section as the first section with length of 0. Then flowitems will convey straight onto the conveyor.

The Conveyor will only receive one flowitem at a time, and will only release one flowitem at a time. What this means is that, if flow items are being transported into or out of the conveyor using a TaskExecutor, only one flowitem can ever be in transit into it, and only one can wait for a transport to pick it up from the conveyor at a time. This is important to know if you want to have several operators pick up flowitems and transport them to the conveyor simultaneously. In order to do this, you would need to have a queue in front of the conveyor since a queue can receive multiple products in transit simultaneously.

### Speed Constraints

The Conveyor does not implement acceleration or deceleration of flowitems. Also, the speed value of the conveyor should not be changed dynamically during a simulation run. Instead, use the `changeconveyorspeed()` command.

### Aligning Conveyors

The Conveyor also has a simple ability to align subsequent conveyors. By holding down the 'X' key and clicking on a conveyor, the conveyor connected to its first output port will be re-aligned to be flush with the conveyor's end point.

### Conveyor X Size

You may notice that the actual size of the conveyor (the size of its yellow box when selected) is not the same as the conveyor's length. Changing the conveyor's y size will correctly change the conveyor width. Changing the conveyor's x size, on the other hand, will change the width of the conveyor's legs.

### Upstream Blocked Length Notification

On the parameters window of the conveyor, there is a checkbox to tell the conveyor to notify upstream of blocked content. This allows for proper accumulation on multiple conveyors in series. However, the inter-conveyor messaging involved in this operation is very processor intensive, and can slow your model down if used ubiquitously, so use this feature only where necessary. Also, photo eyes are not supported when this feature is used.

### Changing Speeds Across Conveyors

You may notice that if you have a model with a fast conveyor followed by a slow conveyor, products will often overlap in the transition. This is a visualization glitch and does not affect output of your model as long as the transfer is not a branch into several conveyors. Even if it is at a branch point, you can fix it by putting the item on a single slow and very short conveyor before the branch point.

### States

**Empty** - There are no flowitems on the conveyor.

**Conveying** - All flowitems are conveying down the conveyor.

**Blocked** - The front flowitem has reached the end of the conveyor and has been released, but has not been accepted by a downstream object yet. Note that this doesn't necessarily mean that all flowitems are stopped, as in the case of an accumulating conveyor.

**Waiting for Transport** - The front flowitem has reached the end of the conveyor and has been released and accepted by a downstream object, but the transport hasn't picked it up yet.

### Photo Eyes

The conveyor allows you to specify locations for photo eyes on the conveyor. Photo eyes are locations on the conveyor that, when blocked, fire OnCover and OnUncover triggers on the conveyor. They don't affect any other logic on the conveyor unless the cover/uncover triggers you specify explicitly change the conveyor behavior. Each photo eye has two user-defined fields: a location along the length of the conveyor, measured from the beginning of the conveyor, and a debounce time.

At any given time, each photo eye can be in one of three states, described as follows.

**Uncovered/Green** - This is when there is no flowitem covering the photo eye.

**Covered/Yellow** - This is when a flowitem is covering the photo eye, but the photo eye hasn't yet been covered for its full debounce time.

**Covered/Red** - This is when the flowitem is covering the photo eye, and the photo eye has been covered for at least its full debounce time.

The following state transfers may occur. A trigger is fired at each state transfer.

**Green to Yellow** - This state transfer happens when a photo eye is not covered, and a flowitem passes over and covers it. This fires the Conveyor's OnCover Trigger, passing a 1 into the covermode parameter of the trigger. The Conveyor also starts a time for the debounce time.

**Yellow to Red** - This state transfer happens when a photo eye is covered (in the yellow state), and its debounce timer expires. This fires the Conveyor's OnCover Trigger again, passing a 2 into the covermode parameter of the trigger. Note that in the trigger logic, you will likely need to distinguish between a green to yellow trigger and a yellow to red trigger. Note also that if you have given the photo eye a debounce time of 0, then the OnCover trigger will be fired twice at the same time: once when transferring from green to yellow, and again when transferring from yellow to red.

**Yellow to Green** - This state transfer happens when a photo eye is covered and in the yellow state, and a flow item with space behind it finishes passing over the photo eye, uncovering it. Here the OnUncover trigger is fired, and 1 is passed into the covermode parameter.

**Red to Green** - This state transfer happens when a photo eye is covered and in the red state, and a flow item with space behind it finishes passing over the photo eye, uncovering it. Here the OnUncover trigger is fired, and 2 is passed into the covermode parameter.

**Note on 0 debounce time:** If you have given the photo eye a debounce time of 0, then the OnCover trigger will be fired twice at the same time: once when transferring from green to yellow, and again when transferring from yellow to red.

**Note on uncovering a photo eye:** The uncover trigger is not necessarily fired every time a flowitem finishes passing over a photo eye. If there is another flowitem back-to-back with the flowitem, then the photo eye will remain covered when the first flowitem finishes passing over it. However, if you've specified a product spacing that is larger than the actual product length, then there will naturally be a gap between products, which will cause the uncover trigger to be fired each time a product finishes passing over the photoeye.

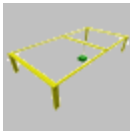
## Photo Eye Visualization

The conveyor's photo eyes are shown as lines across the conveyor. To hide a conveyor's photo eyes, you can hold the 'B' key down and click on the conveyor, or you can hide the photo eyes from the photo eyes page of the conveyor's parameters window.

## Parameters tab-pages

Conveyor  
Layout  
Photo Eyes  
Flow  
ConveyorTriggers

## Crane



### Overview

The crane has similar functionality to the transporter but with a modified graphic. The crane works in a fixed space following rectangular x,y,z movements. It is designed to simulate rail-guided cranes such as gantry, overhead, or jib cranes. By default, the crane picker rises to the height of the crane object after picking up or dropping off a flowitem before it will travel to the next location. To exercise more control over the movements of the picker from one pickup to the next, change the crane's travel sequence in its parameters window.

### Details

The Crane is a sub-class of the TaskExecuter. It implements offset travel according to a travel sequence that the user specifies. By default, this travel sequence is L>XY>D. The '>' character separates travel operations, L means lift the hoist, X means move the gantry, Y means move the trolley, and D means drop the hoist to the offset position. The default travel sequence tells the crane to first lift the hoist, then move the gantry and trolley simultaneously to the offset position, then drop the hoist. The crane travels so that its x/y center and z base arrive at the destination location. If there is an involved flowitem for the offset travel task, then the crane travels so that its x/y center and z base arrive at the top of the flow item, or in other words, it increases the arrival z location by the z size of the flowitem.

### States

This object follows TaskExecuter states.

### Parameters tab-pages

- Crane
- TaskExecuter
- Collision
- Dispatcher
- TaskExecuterTriggers

## Dispatcher



### Overview

The dispatcher is used to control a group of transporters or operators. Task sequences are sent to the dispatcher from an object and the dispatcher delegates them to the transports or operators that are connected to its output ports.. The task sequence will be performed by the mobile resource that finally receives the request.

### Details

The Dispatcher object performs queueing and routing logic for task sequences. Depending on the modeler's logic, task sequences can be queued up or dispatched immediately once they are given to a Dispatcher.

When a Dispatcher receives a task sequence, triggered by the `dispatchtasksequence()` command, it first calls its `Pass To` function. This function returns the port number to pass the task sequence on to. The Dispatcher will then immediately pass the task sequence on to the object connected to that port. If the function returns a 0 instead of a port number, then the task will be queued up in the Dispatcher's task sequence queue. This is done by calling the queue strategy function for the task sequence. This queue strategy returns a value associated with the task sequence, and represents a priority to sort the task sequence in the queue. Higher priority values go to the front of the queue, and lower values go the back. Usually you will simply return the priority value of the task sequence, but the queue strategy function allows you to dynamically change the priority of a task sequence if needed. When ordering the task sequence in the queue, the Dispatcher actually calls the queue strategy function several times, once for each task sequence in the queue, to get each priority value and compare it with the new task sequence's priority value. Once it has found the right location to put the task sequence, it ranks the new task sequence accordingly.

A Dispatcher is a super-class of all `TaskExecuters`, or in other words all `TaskExecuters` are also `Dispatchers`. This means that an `Operator` or `Transporter` can also act as a Dispatcher or team leader, giving task sequences to other members of its team, as well as executing task sequences itself.

### States

The Dispatcher doesn't implement any states.

### Parameters tab-pages

Dispatcher  
DispatcherTriggers

## Elevator



### Overview

The elevator is a special type of transport that moves flowitems up and down. It will automatically travel to the level where flowitems need to be picked up or dropped off. Flowitems are animated as they enter and exit the elevator. This gives a better feel for the load and unload time of the elevator.

### Details

The Crane is a sub-class of the TaskExecuter. It implements offset travel by only traveling the z portion of the offset location. If the offset travel is for a load or unload task, then once the offsetting is finished, it will use the user-specified load/unload time to convey the flowitem onto its platform, or off of its platform to the destination location. When conveying the item onto or off of the elevator, the flowitem moves directly along the elevator's x-axis.

**Note on conveying a flowitem onto an Elevator:** For a load task, the conveying of the item onto the platform may not work if the flowitem is in an object that continuously refreshes the location of the flowitem, like a conveyor for example. In this case, if you want the conveying of the item onto the platform to show up, then make sure that the Elevator is ranked after the object it is picking up from in the model tree (the Elevator must be lower down in the tree).

The Elevator does not connect itself to a navigator by default. This means that travel tasks will not be performed. Instead, all traveling is done using offset travel.

### Context

Since the main distinction of an Elevator is that it only moves along its z axis, this object can be used for any purpose in which you want the object to only travel along one axis.

### States

This object follows TaskExecuter states.

### Parameters tab-pages

- Elevator
- TaskExecuter
- Collision
- Dispatcher
- TaskExecuterTriggers



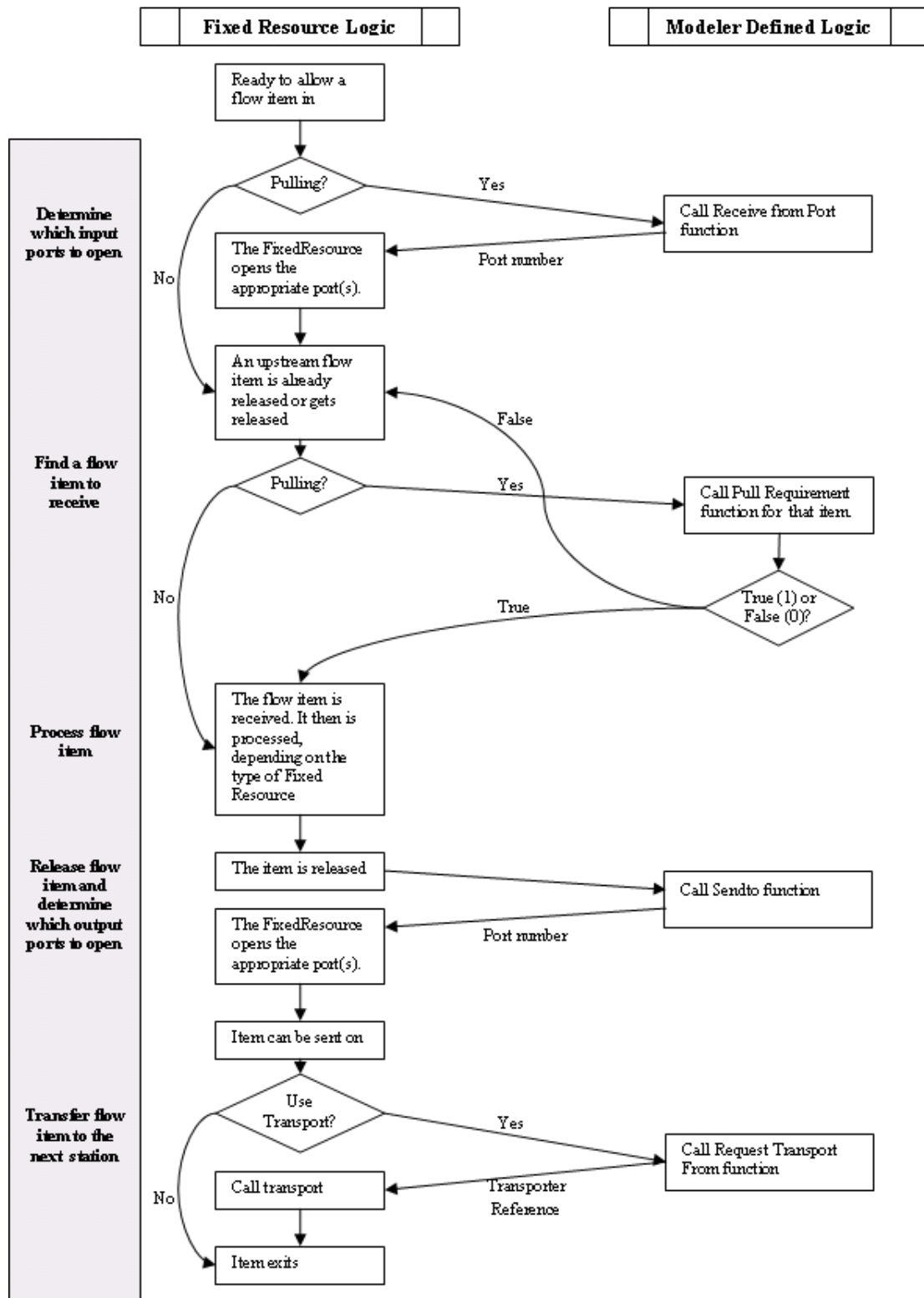
## FixedResource

### Overview

The FixedResource is a superclass of the Source, Queue, Processor, Sink, Combiner, Separator, Rack, FlowNode, MultiProcessor and Reservoir objects. It defines logic for pulling flowitems into the station, as well as sending the object on. You will not drag this object explicitly into your model, but will instead edit FixedResource logic using the Flow tab of sub-class objects.

### Details

The term FixedResource describes a class of objects that handles flowitems in a certain way. They "receive" flowitems through their input ports, do something to those flowitems, then "release" the flowitems to be sent on through their output ports. While different types of FixedResources receive and release flowitems at different times, the processes of receiving and releasing a flowitem are the same for all FixedResources. For example, the Queue can receive several flowitems at the same time. The Queue also releases each flowitem as soon as it enters the Queue. The Processor on the other hand receives exactly one flowitem, processes that flowitem, then releases it and waits until the flowitem has left before receiving the next flowitem. Although the Queue and Processor receive and release flowitems at different times, the processes of receiving and releasing the flowitem are the same for both. Each goes through a certain set of steps for each flowitem that it receives and releases. Some of these steps are automatically handled by the FixedResource, and some allow you as a modeler to define the way flowitems are received and released. All of these modeler-defined inputs can be edited in the "Flow" tab of an object's parameters window. The following diagram shows the steps that a FixedResource object goes through for each flowitem that it receives and subsequently releases. This diagram describes a FixedResource that does not have its "Continuously Evaluate Sendto" box checked.

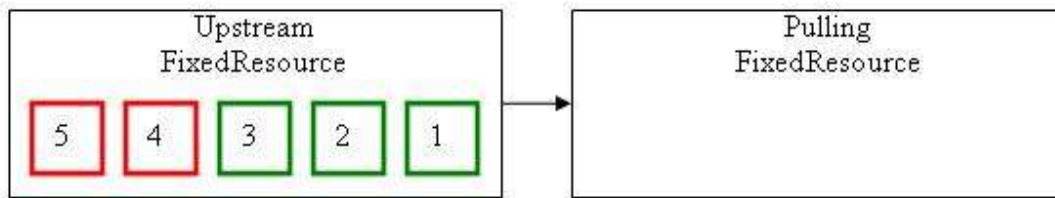


### 1. Open input ports and find a flowitem to receive

When the FixedResource becomes ready to receive a flowitem, it checks first to see if it is in Pull mode. If it is in Pull mode, then it calls the Receive from Port function. This

function returns a value of the input port number to open. If 0 is returned, then it will open all input ports. When an upstream item is released, it calls the Pull Requirement field for that item. This field should return a true (1) or false (0). If true, then it receives the flowitem. If false, then it tries calling the Pull Requirement function again for the next flowitem that has been released, or waits until another flowitem is released by an upstream FixedResource. It continues this loop until the Pull Requirement returns a yes (1) for a particular item, upon which it receives that item. If the object is not in Pull mode, however, then the FixedResource skips all of the pulling logic, and simply waits for the first flowitem that becomes available.

### Pull Logic Example



The above diagram shows two FixedResources. The Pulling FixedResource is set to pull from the Upstream Fixed Resource. The Upstream FixedResource has released 3 of its flowitems (green), while 2 flowitems are still being processed (red). When the Pulling FixedResource is ready to receive one of the Upstream FixedResource's flowitems, it calls its Pull Requirement function for each of the 3 released flowitems, until the Pull Requirement function returns a yes (1). As soon as a yes is returned, the Pulling FixedResource will receive that item and finish its pulling logic, until it is ready to receive the next flowitem. If all of the 3 calls to Pull Requirement return a no (0), then the Pulling FixedResource will wait. When flowitem 4 is released later in the simulation, the Pulling FixedResource will then call Pull Requirement for item 4, and if it returns yes, item 4 will be received. If it returns no, this process will repeat until a flowitem is found that meets the Pull Requirement.

**Note for a FixedResource waiting for an upstream flowitem to be released:** When an upstream flowitem is released, the Pulling FixedResource will only call the Pull Requirement on that specific flowitem. It will not call the Pull Requirement on previously released flowitems for which the Pull Requirement has already returned a no (0). If you would like to re-evaluate the Pull Requirement for all released flowitems, then check the "Continuously Evaluate Pull Requirement" checkbox, and the Pulling FixedResource will re-evaluate the pull requirement for all released flowitems each time a flowitem is released.

**Note on the sendto of upstream objects that an object is pulling from:** If an object is configured to pull from upstream objects, the sendto of those objects is nullified, meaning that you might as well not even specify a sendto for those objects. Just leave it as "First Available"

## 2. Process the flowitem

Once the flowitem has entered the FixedResource, the item is "processed" according to the type of FixedResource, and then released. For example, if it is a Processor object, then the flowitem will be processed for a certain amount of time. If it is a

Queue object, then the product is released immediately. If it is a Conveyor object, then the flowitem will convey down the length of the conveyor and be released when it hits the end.

### 3. Release the flowitem and determine which output ports to open

When the flowitem is released, the FixedResource calls the Sendto function. Like the Pull from Port function, this function returns a port number. The FixedResource then opens that port. If the port number is 0, then it will open all of its output ports. Once the ports are opened, the FixedResource waits until a downstream object becomes ready to receive the flowitem. If the FixedResource is configured to continuously evaluate its sendto, then each time a downstream FixedResource becomes available to receive a new flowitem, the upstream FixedResource will re-evaluate the sendto for that flowitem. It's important to note here that this is only executed when the downstream object **becomes** available. It does not continuously evaluate just because a downstream object is already available. If you want to manually force a re-evaluation at some other time than when a downstream object becomes available, then you can do so by calling the openoutput() command on the upstream object.

**Note on the returned sendto value:** If the returned port is greater than 0, then that port will be opened. If the returned port is 0, then all ports will be opened. If the returned port is -1, the flowitem will not be released, and should be released explicitly later on using the releaseitem() command, or should be moved out using the moveobject command. When it is released again, the sendtoport function will be called again. A -1 return value is more for advanced users.

### 4. Transfer the flowitem to the next station

Once the flowitem is released, and a downstream object is ready to receive it, if the "Use Transport" checkbox is not checked, then the item will be passed immediately in to the downstream object. If the "Use Transport" checkbox is checked, then the FixedResource will call the Request Transport from function. This function should return a reference to a TaskExecutor or Dispatcher. If a valid reference is returned, then a default task sequence will automatically be created, and a TaskExecutor will eventually pick the flowitem up and take it to its destination. You can also return a 0 value in the Request Transport From field. If a 0 value is returned, then the FixedResource will assume that a task sequence has been created explicitly by the user, and will not create the default task sequence himself. If you return a zero, then you will need to create the task sequence yourself. You can easily get started at doing this by selecting the "Create task sequence manually" pick option in the Request Transport Field picklist, then manually editing the code from there.

### Using a Transport

If an object is configured to use a transport to transport flowitems to downstream objects, then when the downstream object pulls the flowitem or becomes ready to receive the flowitem, instead of immediately moving the flowitem into the next station, the object instead creates a task sequence for a TaskExecutor to travel to the object, pick up the flowitem, travel to the downstream object, and drop it off there. This operation involves several important steps. First, when this happens, the object calls its Request Transport From function, and gets a reference of the object to give the task sequence to. Then the flowitem goes into a "Waiting For Transport" state. This means that the destination for that flowitem has been set in stone, and cannot be

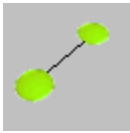
changed. Sendto and pull screening has already finished and decided on sending the flowitem out that port, and this decision will not change. Also, each FixedResource object keeps track of two numbers: the number of flowitems that are in transit to the object, and the number of flowitems that will be transported out of the object, but have not been picked up yet. These two variables are respectively called nrofransportsin and nrofransportsout. Once the object has called the Request Transport From field, it increments its own nrofransportsout variable, and notifies the downstream object, which subsequently increments its own nrofransportsin variable. The object then creates a task sequence of:

1. Travel to the upstream object: travel task.
2. Load the item: Frload task.
3. Break to other task sequences if appropriate: break task.
4. Travel to the downstream object: travel task.
5. Unload the item into the downstream object: Frunload task.

Note that the FixedResource uses frload/frunload tasks instead of regular load/unload tasks. These tasks are just like regular load and unload tasks except that right before the TaskExecutor moves the flowitem, it notifies the FixedResource of the operation being done, so that the FixedResource can appropriately decrement its nrofransportsin/nrofransportsout variable. In summary, the nrofransportsout of the upstream object and the nrofransportsin variable of the downstream object are both incremented at the same time that the Request Transport From function is called and the task sequence is created. The nrofransportsout variable is then decremented just before the TaskExecutor finishes the frload tasks and moves the flowitem out of the upstream object. The nrofransportsin variable is decremented just before the TaskExecutor finishes an frunload task and moves the flowitem into the downstream object.

The integrity of the nrofransportsin and nrofransportsout variables is essential to the correct operation of the objects involved because each object may screen further input/output based on these variables. For example, consider a queue with capacity of 10. If the queue's current content is 5 and it also has 5 products that have not arrived yet, but are in transit to the queue, then the queue must close its input ports because it may possibly become full, even though at this point it only has 5 products. An incorrect nrofransportsin variable could cause serious problems for the queue's content. What does this mean for you? Two things. First, if an object has chosen to transport a flowitem to a given downstream object, there is no turning back or redirecting the flowitem to a different destination, because this would mess up the proper decrementing of nrofransportsin/out variables. Secondly, be very aware of when you use frload/frunload versus regular load/unload, because this can affect the input/output functionality of the objects. Simply put, each execution of a Request Transport From function should eventually (if not immediately) result in the creation of exactly one frload task to load the item from the upstream object and exactly one frunload task to unload the item to the downstream object. In all other cases, regular load and unload tasks should be used.

## FlowNode



### Overview

The FlowNode is used to move flowitems from one location to another with time being consumed. The FlowNode is simply connected in the flow using the 'A' key to click-and-drag a connection. For example, if you want products to take time to move from a Queue to a Processor, place a FlowNode in between and connect up the output port from the Queue to the input port of the FlowNode. Then connect the output port of the FlowNode to the Processor.

### Details

The FlowNode is a sub-class of the FixedResource. It will continue receiving flowitems until its maximum content is met. As soon as a flowitem enters the FlowNode, the FlowNode executes the sendto for the flowitem. It then executes the speed field to find out the speed of the flowitem. Then it sets the flowitem's location to its own location, and starts moving the flowitem towards the downstream object that was returned by the sendto. Once the flowitem reaches the location of the destination object, the FlowNode releases it.

Note that the FlowNode executes the sendto field as soon as a flowitem arrives. This is different than the FixedResource's usual functionality of executing the sendto when the flowitem is released. This difference has some important implications. First of all, you should not return a 0 from the sendto field, since the FlowNode needs to decide immediately which downstream object to send to. If a 0 is returned from the sendto, then the FlowNode will always send to output port 1. Secondly, although you are free to do it, it wouldn't make much sense to evaluate the sendto continuously for a FlowNode. Since the flowitem must commit to one output port when it enters, evaluating sendto continuously would make it so the flowitem arrives at one destination object, but its sendto may change after it arrives, sending it out a different port than the one it traveled to.

### Context

FlowNodes allow you to simulate travel networks with flowitems. Using conveyors to simulate the travel networks can be done as well. Building with conveyors is done from the perspective of creating and connecting the edges of the travel network, while using FlowNodes is done from the perspective of creating and connecting the nodes of a travel network, and can in some instances make model building easier. However, unlike conveyors, FlowNodes do not provide accumulation of flowitems along the edges. They only allow one maximum content value to constrain traffic on the network. Hence, if you have a travel area where you would like more flexibility in defining traffic control, use conveyors instead of FlowNodes. You can also use NetworkNode travel networks, instead of FlowNodes, by using the TaskExecuter flowitem, checking the "Use Transport" box of an upstream FixedResource from which you want the flowitem to travel, and selecting the "Flow Item as TaskExecuter" option in the Request Transport From pick list.

**States**

The FlowNode does not implement any states. You can use its Content vs. Time graph to attain statistics on the FlowNode.

**Parameters tab-pages**

FlowNode

Flow

FlowNode Triggers

## MergeSort



### Overview

The MergeSort is a non-accumulating conveyor that allows you to have multiple input positions, as well as multiple output positions along the conveyor. Each input/output port of the conveyor has a user-defined input/output position.

### Details

The MergeSort is a sub-class of the Conveyor, which is subsequently a sub-class of the FixedResource. Every input port of the MergeSort has an associated entry location along the length of the conveyor. Every output port has an associated exit location and a blocking parameter.

### Receive/Release Logic

For each input point, the MergeSort will receive flowitems at that entry point as long as the entry point is clear of flowitems and there is enough room in front of the entry point to fit the entering flowitem. Entering flowitems are placed with their front edge against the entry point, and begin flowing down the length of the conveyor.

**Note on entry/exit positions:** if you've changed the entry/exit positions of the mergesort, you will need to reset the model in order for those positions to be placed correctly.

Each time that a flowitem reaches an exit point on the conveyor, the MergeSort calls its Send Requirement field for that port. If the send requirement returns true, then it will "attempt" to send the flowitem out that port by releasing it. If the downstream object is ready to receive the product, then the attempt will succeed and the flowitem will be sent out that output port. If the attempt fails, one of two things will happen. If the blocking parameter for that output port is 0, then the flowitem will be "un-released" and continue conveying down the length of the conveyor. If the blocking parameter is 1, then the whole conveyor will stop until the downstream object is ready to receive the product.

Flowitems that reach the end of the conveyor without exiting will loop back around to the beginning of the conveyor and convey down the length of the conveyor again. This is why it is advised that the last exit point of every MergeSort be blocking, because flow items should never be able to pass the last exit point without exiting, unless you want them to try again along the whole length of the conveyor. When a flowitem loops around to the beginning of the conveyor, the entry trigger is fired with an involved port of 0.

**Note on blocking:** The MergeSort is a NON-ACCUMULATING conveyor, which means that if one product on the conveyor is stopped, then all products on the conveyor are stopped. Products will not accumulate when blocked. Be aware of this



before you use a MergeSort in your model, because otherwise you may end up having to redo a lot of logic you put on your MergeSort using regular conveyors instead.

### FixedResource Logic Variation

The FixedResource implementation for the MergeSort has quite a few variations from the standard that are important to know. First of all, the MergeSort is always in pull mode. However, unlike other FixedResources in pull mode, it does check the sendto value of upstream flowitems to make sure it is alright to send the flowitems to the MergeSort. The other thing different from normal FixedResources is that the user doesn't have access to a Pull from Port field. The MergeSort handles this logic on its own, since each port's ability to receive flowitems is dependent on the position of the entry point as well as the location and size of other flowitems on the conveyor. Also, there is no Send to Port field that returns a port number. Again, the port the flowitem can exit through is determined by the position of the flowitem and the position of the exit point. Instead, the MergeSort provides the Send Requirement field, which is fired every time a flowitem passes an exit location. This field should return a true or false (1 or 0) of whether the flow is allowed to leave out that exit point.

**Note on using transports:** Be very careful about using TaskExecutors to transport flowitems into the conveyor. The ability of the MergeSort to receive flowitems through a given port is dependent on the position of other flowitems on the conveyor at any given time in the simulation. If there is a delay between the time that the MergeSort becomes ready to receive a flowitem and the time it actually receives the flowitem, the window of opportunity to receive the flowitem at that entry point may have passed by the time the flowitem arrives. This will cause the MergeSort to convey flowitems incorrectly. If you need to transport flowitems into the MergeSort, then you should only transport them into the very first entry point on the conveyor. Otherwise, transport flowitems onto regular conveyors that feed into the MergeSort. Also, the ability to use transports when exiting the MergeSort has been disabled. Again, use regular conveyors, feed the MergeSort into those conveyors, and use transports exiting those conveyors.

### Entry/Exit Point Visualization

In your Ortho/Perspective view, the positions of the entry/exit points are drawn as red or green arrows. Input locations have arrows drawn pointing into the conveyor. Exit locations have arrows pointing out of the conveyor. A green arrow means either the MergeSort can receive through that entry point, or is waiting to send through that exit point. A red arrow means the MergeSort is not available at the entry point, or there is now flowitem waiting to exit through the exit point. To hide these arrows, set the object to not show ports, either from its Properties window, or from the Edit Selected Objects menu.

### States

**Empty:** There are no flowitems on the conveyor.

**Conveying:** All flowitems are conveying down the conveyor.

**Blocked:** A flowitem on the conveyor has reached an exit point with a blocking parameter of 1, but has not left the conveyor yet, so all flowitems on the conveyor are stopped.

## Parameters tab-pages

MergeSort

Conveyor

Layout

ConveyorTriggers

## MultiProcessor



### Overview

The MultiProcessor is used to simulate the processing of flowitems in sequentially ordered operations. The user defines a set of processes for each MultiProcessor object. Each flowitem that enters will go through each process in sequence. MultiProcessors may call for operators during their process steps.

### Details

The MultiProcessor is a sub-class of the FixedResource. It receives one flowitem, puts the flowitem through its sequence of processes, then releases the flowitem. Once the flowitem has exited the MultiProcessor, it receives another flowitem and goes through the processes again. Only one flowitem will ever be in the MultiProcessor at one time.

For each process that you define, you can specify a name for the process, a process time, a number of operators to use for that process, priority and preempting values for the tasks sent to the operators, and the operator or dispatcher to send operator tasks to. At the beginning of each process, the MultiProcessor calls the process time field, sets its state to the name of the process, and calls operators if the number of operators value is greater than 0. When the process is finished, the MultiProcessor releases all operators called for that process, and calls the process finish trigger. It also passes the process number into the process finish trigger as parval(2).

### Context

You would use the MultiProcessor if you have one station that involves several operations with separate process times and/or different resources. You can also use the MultiProcessor as a shared station for different types of operations. For example, itemtype 1 needs to go through operations A, B, C, and D, and itemtype 2 needs operations E, F, G, and H, but both itemtypes must share one station for their processes. Give the MultiProcessor 8 processes: A - H, and for itemtype 1 have processes E - H have 0 process time, and for itemtype 2 have processes A - D have 0 process time.

Note that the MultiProcessor does not accommodate piping of processes. Piping happens if, once a flowitem gets finished with process 1 and moves to process 2, another flowitem can take up process 1. Thus several flowitems can be "flowing down the pipe" at any given time. If you would like to simulate this, use several Processor objects connected in sequence.

### States

**Idle** - There are no flowitems being processed

**User-defined states** - These are states defined by the user, one for each process.

**Blocked** - The MultiProcessor has finished all processes for a flowitem, released it,

but there is no downstream object ready to receive it.

**Waiting for Operator** - The MultiProcessor is waiting for operator(s) to arrive in order to start a process.

**Waiting for Transport** - The MultiProcessor has finished all processes for a flowitem, released it, and there is a downstream object ready to receive it, but the flowitem has not been picked up by a TaskExecuter yet.

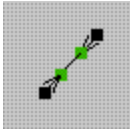
### Parameters tab-pages

MultiProcessor

Flow

MultiProcessorTriggers

## NetworkNode



### Overview

NetworkNodes are used to define a network of paths that transporters and operators follow. The paths can be modified using spline points to add curvature to the path. By default, objects traveling on a network will follow the shortest path between their origin and destination. To learn how to begin using NetworkNodes, refer to Lesson 3 of the Flexsim tutorial.

### Details

Connection of travel networks is done in three steps:

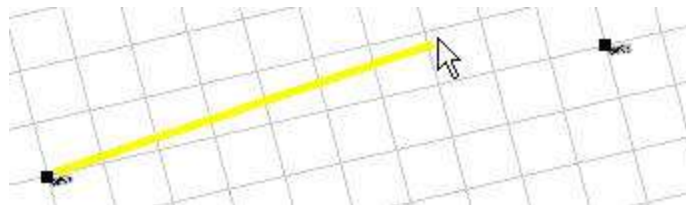
1. Connecting NetworkNodes to each other.
2. Connecting NetworkNodes to the objects they will act as gateways to.
3. Connecting TaskExecutors to the NetworkNodes they will be stationed at for the beginning of the simulation.

### Connecting NetworkNodes to Each Other

Each NetworkNode can have several paths connecting it to other NetworkNodes. Each path represents two single-direction travel paths. Each path direction can be configured independently.

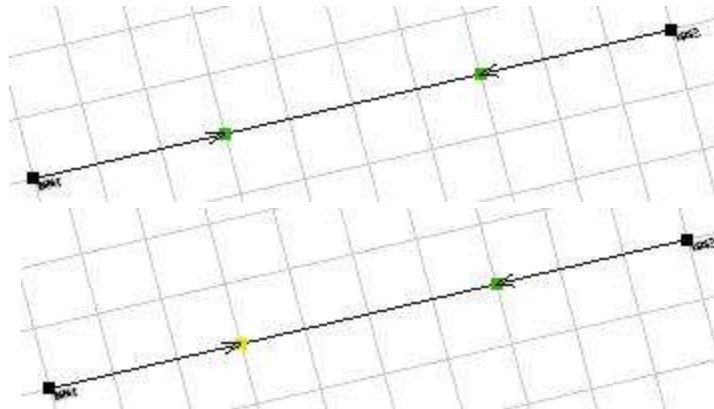
### Configuring Paths in the Ortho/Perspective View

To create a path between two network nodes, hold down the 'A' key, click on one network node, and drag to the other node.



This will create two one-way, passing connections between the nodes. The path is drawn as a green tape between the two nodes. The tape is divided into two sides. Each side describes one direction of the path. Subsequent 'A' drag connections will toggle one direction of the path between passing and non-passing (green and yellow). The direction you toggle is determined by which node you drag from and which node you drag to. The diagram below shows two paths. The first is a passing connection in both directions. The second is a passing connection going right-to-left, and non-passing going left-to-right. The sides of the tape and which direction they

describe is determined in the same manner as the American road system: you drive down the right side of the road.



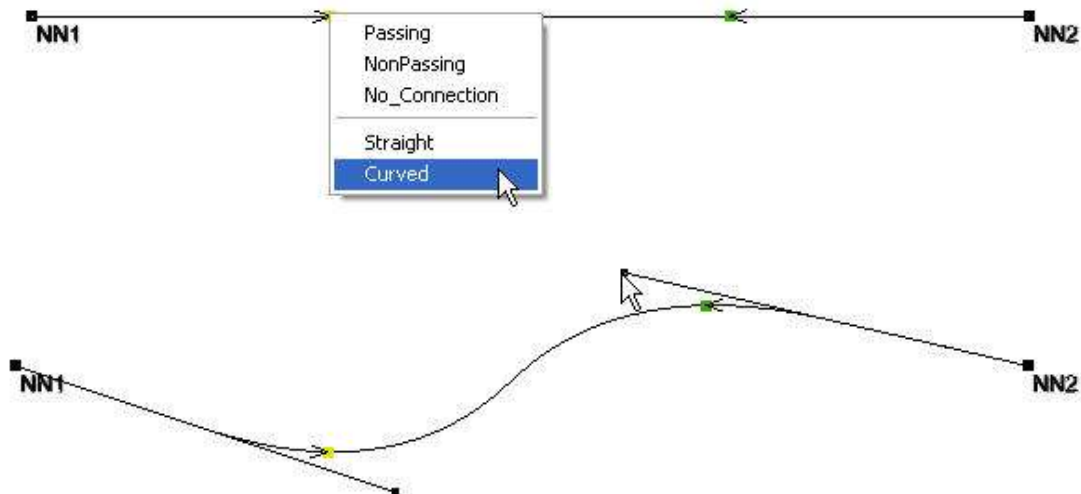
A 'Q' drag connection will toggle one direction of the path as "No Connection", which means travelers aren't allowed to travel in that direction. This type of connection is drawn in red. The diagram below shows a non-passing connection going left-to-right and a no connection going right-to-left. If a 'Q' connection is made in both directions, the whole connection will be deleted.



You can also change the connection type of a given colored box by right clicking on the box and selecting a menu option, or by holding the 'X' key down and clicking on the box.



By default, connections are made as straight paths between nodes. You change these connections to be curved connections by right-clicking on one of the connection's colored boxes and selecting Curved. Two smaller boxes, called spline control points, will be created, and you can move them to create a curved path.



You can also configure how network node connections are made by default through the Travel Networks tool panel in the ortho or perspective view's toolbar.

### Configuring Paths Through the NetworkNode's Path Tab Page

When you open a NetworkNode's parameters window, the Paths page allows you to configure all one-directional paths that extend **out of** that node. If you want to configure paths going **into** the node, then edit the parameters windows of the nodes that connect to it. For each path going out of the NetworkNode, you can give it a name, specify the type of travel connection it is, the spacing, the speed limit, and a "virtual" user distance:

**Name** - The name of a connection is simply for semantic purposes, and has no effect on model logic.

**Connection Type** - There are three connection types: No Connection, Passing, and Non-passing. No connection means that no traveler should travel along this path, in the given direction. If this is selected, then the path will be colored red down the corresponding side of the path. Passing means that travelers do not back up along the path, but simply pass each other if speeds vary. Non-passing means that travelers along this path will actually back up, using the spacing value as a buffer distance between them.

**Spacing** - Spacing only applies if the path is Non-passing. This is the distance to keep between the back of one traveler and the front of another traveler on the path.

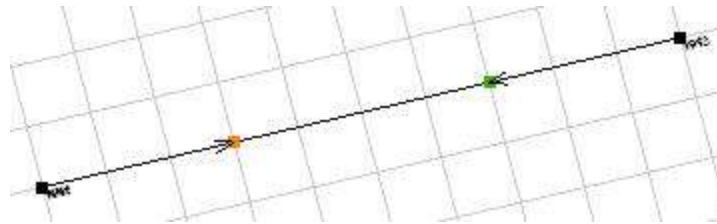
**Speed Limit** - This is a speed limit for the path. Travelers will travel the minimum of their own speed and the speed limit of the path. If the path is a Passing connection, then travelers will accelerate or decelerate to the appropriate speed once they get on the path. If it is non-passing, however, then travelers will immediately change their speed to the appropriate speed, without using acceleration or deceleration.

**Virtual Distance** - Here you can enter a user-defined distance for the path. This would be used if you want to set a specific distance, and override the path's actual distance, or if it is a very large distance, but you don't want to have to put the other NetworkNode in a remote part of the model. If 0 is entered, then it will use the actual distance of the path. Otherwise it will use the distance you enter.

Each node's connection has a number associated with it. This is the same as the order of the listing of the connections in the path tab page. The first connection in the list is associated with connection 1, the second with connection 2, and so forth. To get a reference to the NetworkNode that is connected to the node through a given connection number, just use the `outobject()` command, with the specified connection number.

### Dynamically Closing Node Edges

You can dynamically close node paths during the simulation, using the `closenodeedge` and `opennodeedge` commands. In these commands, you specify the networknode, and either a number rank or a name of the edge. A closed node edge does not allow any more travelers onto the edge until it has been opened again. However, travelers already on the node edge when it is closed will continue and can exit the edge. A closed node edge is drawn with an orange color, as show below. When the model is reset, all node edges that were previously closed will be opened again.



### Acceleration/Deceleration on Node Edges

Acceleration and deceleration will work on network edges. Objects will accelerate to their maximum speed as they begin travel on the network. They will also decelerate when approaching the destination. Also, if a traveler is traveling at its maximum speed and it arrives at an edge whose speed limit is less than its maximum speed, then it will decelerate to the speed limit. The deceleration starts once the traveler starts moving along that edge with the lower speed limit, not before it gets there.

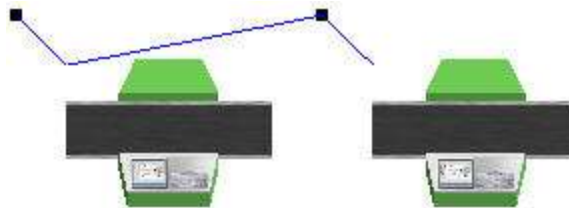
### Connecting NetworkNodes to Objects

To connect a NetworkNode to some object in the model for which you want the NetworkNode to act as a travel gateway, make an 'A' drag connection between the NetworkNode and the object. The NetworkNode will draw a blue line between it and the top left corner of the object. Making this type of connection means that any TaskExecutor traveling on the network that wants to travel to that object will travel to the NetworkNode that it is connected to.





You can connect several NetworkNodes to the same object. This will cause a TaskExecutor wanting to travel to the object to travel to the NetworkNode nearest to itself that is connected to the object. You can also connect several objects to the same NetworkNode. These capabilities are shown below. The Processor on the left is connected to both the NetworkNode on the left and the NetworkNode on the right. The NetworkNode on the right is also connected to both the Processor on the left as well as the Processor on the right.



If you connect a node to a station, but don't see the blue line, try moving the NetworkNode around, as the blue line might just be covered up by a grid line.

### Connecting NetworkNodes to TaskExecutors

To connect a NetworkNode to a TaskExecutor that you want to travel on the network, make an 'A' drag connection between the NetworkNode and the TaskExecutor. The NetworkNode will draw a red line between it and the top left corner of the object. Making this type of connection means that any TaskExecutor given a Travel task will use the network to get to the destination. It also means that the node that you connect the TaskExecutor to is the first node that it will travel to when it needs to travel through the network. Whenever a TaskExecutor finishes a travel operation, arriving at the node connected to the travel operation's destination object, the TaskExecutor will become *"inactive"* (By *"inactive"* we mean that the TaskExecutor is inactive on the travel network. The object may be actively executing other operations like load, unload, or utilize tasks, but it is not currently traveling on the network.) at that node, and the red line will be drawn to the TaskExecutor while he is doing other operations in that node's area. What this means is, the next time the TaskExecutor gets a travel task, he must return to the NetworkNode he was inactive at in order to get back onto the network.



You can connect several TaskExecuters to the same NetworkNode. All TaskExecuters connected to a NetworkNode will reset their position to the position of the NetworkNode they were originally assigned to when the model is reset.

If you connect a node to a TaskExecuter, but don't see the red line, try moving the NetworkNode around, as the red line might just be covered up by a grid line.

If you want to connect/disconnect a NetworkNode as a travel gateway to a TaskExecuter, use the 'D' and 'E' keys to connect and disconnect. Connecting in this manner, will cause a blue line to be drawn to the TaskExecuter indicating that other TaskExecuters traveling to that TaskExecuter will travel to the NetworkNode connected to it with the blue line.

## Viewing Connections

Once you have built a travel network, you can configure which types of connections you want to be drawn in the Ortho/Perspective view. The network has a set of drawing modes, from showing the most information to showing the least information. These modes are listed below.

- Mode 1: Show nodes, paths, object/TaskExecuter connections, spline control points
- Mode 2: Show nodes, paths, object/TaskExecuter connections
- Mode 3: Show nodes, paths
- Mode 4: Show nodes
- Mode 5: Show only one node

When you hold the 'X' key down and repeatedly click on a NetworkNode, the whole network will toggle through these modes, showing less information with each 'X' click. When you hold the 'B' key down and repeatedly click on a NetworkNode, the whole network will toggle backwards through the list of modes. You can also select a set of NetworkNodes (hold Ctrl key down and click on several nodes) and then 'X' click on one of that set to have the draw mode toggling only apply to the NetworkNodes you've selected. If you have selected a set of NetworkNodes and 'X' click on a NetworkNode that is not selected, then draw mode toggling will only apply to NetworkNodes that are not selected. This can be very useful if you have a very large model and don't need all spline connections to be drawn.

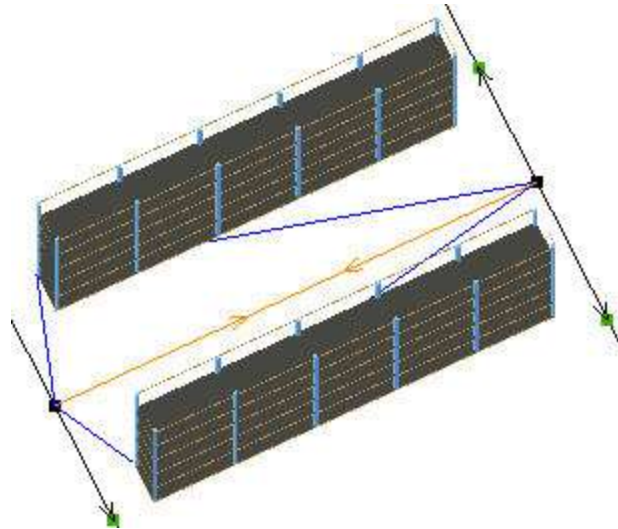
## Maximum Number of Travelers

You can specify the maximum number of travelers allowed to be inactive, or stationary, at the node. An inactive traveler is one that is connected to the network node, and is not performing a Travel task, but rather is doing another task or is idle. You can tell a traveler is inactive by the fact that there is a red line drawn between the traveler and the network node.

If the maximum number of stationary travelers for a network node is set to 1, and a traveler is already stationary at the node, then when another traveler arrives at the node it will have to wait until the first traveler leaves the node before it can finish its travel task. Note that this only applies when the second traveler's destination is that node. If the traveler just wants to pass through the node to another node, then it will not have to wait.

## Virtual Exits

NetworkNodes can also have virtual exits. Above it was mentioned that when a TaskExecutor finishes a travel task, it becomes inactive at the destination NetworkNode. Once it gets another travel task, it must go back to the original NetworkNode it was at in order to get back onto the network. Virtual exits allow you to specify alternative nodes that the TaskExecutor can travel to in getting back onto the network. Virtual exits are created between NetworkNodes. They are created by holding down the 'D' key and dragging from one NetworkNode to another. An example is shown below.



The above figure shows two Rack objects and two NetworkNodes. The NetworkNodes are travel gateways to the Rack objects (blue lines are drawn between the Racks and the nodes). A two-way virtual exit connection has also been made between the two NetworkNodes (orange arrows pointing to either node). This means that if a TaskExecutor arrives at the racks through one of the NetworkNodes, and then needs to get back onto the network, it can "exit" the area through any one of the two, whichever has a shorter total distance. Orange arrows pointing out of a given NetworkNode mean that if a TaskExecutor is inactive at that node, it can exit through any one of the NetworkNodes that node is connected to. If it needs to exit through a different node than it entered, it uses the `reassignnetnode()` command to reassign itself to the new node. Then it simply exits through that node.

To delete a virtual exit, hold the 'E' key down and drag between NetworkNodes in the same direction of the virtual exit connection you want deleted.

## Commands

There are several commands you can use to dynamically manipulate networks and transports. These are as follows. More detailed information can be found in the command summary.

**`reassignnetnode(object transport, object newnode)`** - This dynamically changes the NetworkNode at which a TaskExecutor is stationed.

**`redirectnetworktraveler(object transport, object destination)`** - If a traveler is on a

network traveling to a given destination, and you would like it to change its destination in the middle of the travel operation, you can use this command.

**distancetotravel(object traveler, object destination)** - This command can be used to calculate the distance from a TaskExecuter's current static node to the destination object.

**getedgedist(object netnode, num edgenum)** - This returns the distance of an edge of the NetworkNode.

**getedgespeedlimit(object netnode, num edgenum)** - This returns the speed limit of an edge of the NetworkNode.

## Changes to the Distance Table

The distance/routing table of all network nodes in the model is kept on a global object in the model called "defaultnetworknavigator". The re-calculation is optimized to only be executed if changes have been made to the network. If you have clicked on a NetworkNode in the model, or if you've 'A' or 'Q' dragged between two NetworkNodes in the model, then the next time you reset the model, the distance/routing table will be re-calculated.

## Duplicating NetworkNodes

Be careful duplicating NetworkNodes. If NetworkNodes have been duplicated, you will need to reset the model before editing any of those new NetworkNodes.

### States

The NetworkNode does not implement any states.

### Parameters tab-pages

NetworkNode  
Connections  
NetworkNodeTriggers

### Related Topics

Tutorial: Lesson 3  
TrafficControl

## Operator



### Overview

Operators can be called by objects to be utilized during setup, processing or repair time. They will stay with the object that called them until they are released. Once released, they can go work with a different object if they are called. They can also be used to carry flowitems between objects. Operators can be placed on a network if they need to follow certain paths as they travel.

### Details

The Operator is a sub-class of the TaskExecutor. It implements offset travel depending on whether there's in an involved item for the offset. If there is no item, then it implements the offset exactly like the TaskExecutor. It travels so that its x/y center and z base arrives at the destination location. If there is an involved item, then the Operator only travels along the x/y plane. It also only travels up to the point where its front edge is at the edge of the flow item, instead of its x/y center. This done by decreasing the total travel distance by  $(xsize(operator) / 2 + xsize(item) / 2)$ .

The Operator also animates walking using the setframe and getframe commands. For any 3ds or wrl file, you can specify frames of that 3d file by creating different 3d models, and saving them as <original file name>FRAME<frame number>.3ds. For example, the operator's primary 3d file is Operator.3ds. This is drawn if its frame is set to 0. Its other frames are defined in OperatorFRAME1.3ds, OperatorFRAME2.3ds, etc. You can also specify your own 3d files and frames. However, if you are going to use the Operator, you should know that the Operator automatically updates frames. If you want to use the Operator and still define your own frames, they should conform with the Operator's frame numbers. These numbers are described as follows.

**Frame 0** - Standing with arms at side.

**Frames 1-6** - Walking with arms at side.

**Frames 7-12** - Walking with arms extended to hold a flowitem.

**Frame 13** - Standing with arms extended to hold a flowitem.

**Frame 14** - Sitting with arms at side.

**Frame 15** - Sitting with arms extended to hold a flowitem.

If the Operator is traveling (either on a travel task or doing offset travel), then the frame will be updated automatically to a frame between 1 and 12. Otherwise, it won't update frames at all. You can set the frame that you want when the Operator isn't traveling. If you don't want the Operator updating its frames even while it's traveling, then return a 1 in the Operator's draw trigger, and the Operator won't do any frame updating at all.

### Context

As mentioned above, the xsize of the item is queried to decrease the total offset distance. This may not work perfectly if you have flowitems whose x size is very different from their y size, and the operator approaches from the y side of the flowitem. If this is the case, and you want it to look more exact, you could try switching x and y sizes of the item and rotate it 90 degrees right before the operator picks it up, then undo those changes from the operator's load/unload trigger.

### States

This object follows TaskExecutor states.

### Parameters tab-pages

TaskExecutor  
Collision  
Dispatcher  
TaskExecutorTriggers

## Processor



### Overview

The processor is used to simulate the processing of flowitems in a model. The process is simply modeled as a forced time delay. The total time is split between a setup time and a process time. The processor can process more than one flowitem at a time. Processors may call for operators during their setup and/or processing times. When a processor breaks down, all of the flowitems that it is processing will be delayed.

### Details

The Processor is a sub-class of the FixedResource. It is also a super-class of the Combiner and Separator. It continues to receive flowitems until its maximum content is met. Each flowitem that enters the Processor goes through a setup time followed by a process time. After these two processes have finished, the flowitem is released. If the maximum content value is greater than one, then flowitems will be processed in parallel.

**Note on the maximum content value:** Be very careful about setting a maximum content value greater than 1. The Processor will not correctly request operators if you are processing more than one flowitem at a time. MTBF/MTTR times will also not be calculated correctly if the maximum content is greater than one. Also, state statistics won't be recorded correctly. The maximum content value is only there to allow you to create multiple very simple processors in parallel without having to drag that many into your model. If you want to edit any parameters other than the simple setup, process times and flow page parameters, then you should drag several Processors out, and have each just receive one flowitem at a time.

### Setup/Process Operators

If the Processor is set to use operators during its setup or process time, then at the beginning of the respective operation it will call the user-defined number of operators using the requestoperators command with the Processor as the station, and the item as the involved object. This will cause the Processor to be stopped until the operators have arrived. Please be aware of how a requestoperators task sequence is constructed, as described in the requestoperators command documentation. Also know how the stopobject command works as explained in the command summary. Once all operators have arrived, the Processor will resume its operation. Once the operation is finished, the Processor will release the operators it called. If the Processor is set to use the same operators for both setup and process time, then the Processor won't release the operators until both the setup process times are finished.

### States

**Idle** - The object is empty.

**Setup** - The object is in its modeler-defined setup time.

**Processing** - The object is in its modeler-defined process time

**Blocked** - The object has released its flowitem(s), but downstream objects are not ready to receive them yet.

**Waiting for Operator** - The object is waiting for an operator to arrive, either to repair a breakdown, or to operate on a batch.

**Waiting for Transport** - The object has released a flow item, and a downstream object is ready to receive it, but a transport object has not picked it up yet.

**Down** - The object is broken down.

### Parameters tab-pages

ProcessTimes

Processor

Flow

ProcessorTriggers

Operators



## Queue



### Overview

The queue is used to store flowitems when a downstream object cannot accept them yet. By default, the queue works in a first-in-first-out manner, meaning that when the downstream object becomes available, the flowitem that has been waiting for that object the longest will leave the queue first. The queue has options for accumulating flowitems into a batch before releasing them.

### Details

The Queue is a sub-class of the FixedResource. It will continue to receive flowitems until it reaches its maximum content. If batching is disabled, the Queue will release the flowitem as soon as it arrives in the Queue, and the OnEndCollecting trigger will be called right before each flowitem is released.

### Batching

If batching is enabled, then the Queue will wait until it receives its target batch of flowitems, then it will release all the flowitems in the batch at the same time. By default the maximum wait time is 0. A max wait time of 0 means that there is no max wait time, or the queue will wait indefinitely to collect its batch. If the max wait time is non-zero, then when the first flowitem arrives, the Queue will start a timer. If at the end of the timer the batch is still not met, the Queue will finish collecting the batch and release all flowitems it collected. The OnEndCollecting trigger is called just before the flowitems are released, a reference to the first flowitem in the batch is passed as item, and the number of items collected is passed as parval(2). If the Queue is configured to flush contents between batches, then it will close its input ports as soon as it ends collecting a batch, and will wait until the full batch has exited before opening the input ports again. If the Queue doesn't flush contents between batches, then it will immediately start collecting another batch as soon as it finishes collecting each batch. This means that you can have several finished batches still in the queue at any given time, waiting to leave.

**Note on the target batch size:** Setting a target batch size does NOT nullify the queue's maximum content value. This means if you set your target batch size to a value higher than the maximum content, the queue will never meet its batch because its maximum content is too small.

### States

**Empty** - The Queue is empty

**Collecting** - The Queue is collecting flowitems for a batch.

**Releasing** - The Queue has finished collecting its batch(es) and is releasing the flowitems that it has. Also if the Queue is not batching, and has flowitems in its queue, it will be in this state.

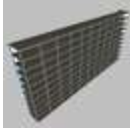
**Waiting for Transport** - The queue has flowitems that have been released and are

ready to move downstream, but are waiting for a Transport to come and pick them up.

### Parameters tab-pages

Queue  
Flow  
QueueTriggers

## Rack



### Overview

The rack is used to store flowitems as if they were in a warehouse rack. The number and size of bays and levels in the rack can be defined by the user. The user can specify the bay and level to place entering flowitems. If a transporter object is used to deliver or pickup flowitems from a rack, the transporter will drive to the specific cell in the rack where the flowitem is assigned. The rack can also be used as storage on the floor of a warehouse, using the bay number to specify an x position to place a flowitem on the floor, and the level to specify the y position to place the flowitem.

### Details

The Rack is a sub-class of the FixedResource. It will continue to receive flowitems until its maximum content value is met. When each flowitem enters the Rack, it executes the minimum dwell time function for that item. This function returns the minimum stay time for that flowitem. The Rack starts a timer for that amount of time. After the timer expires, the Rack releases the flowitem.

### Extra Parameters in Entry/Exit Triggers

The Rack passes extra parameters into the entry and exit triggers. Within the triggers, `parval(3)` is the bay that the flowitem is in, and `parval(4)` is the level the flowitem is in.

### Place in Bay, Place in Level Functions

The timing of the place in bay and place in level function calls can depend on the configuration of the rack in the model. It depends on whether flowitems are being transported to the rack or if they are being moved directly from upstream objects. If they are moved from upstream objects, the placement functions are called when they are received (in the OnReceive event). If they are being transported into the Rack, then the placement functions are called when the transport finishes its travel task and starts the offset travel of the unload task. For more information on task sequences, see the task sequence section. At this point, the transport queries from the Rack where to place the flowitem. The Rack calls the placement functions so that it can tell the transport to travel to the correct bay and level. If the placement functions are called when the transport asks to place them, then they will not be called again when the flowitem actually enters the Rack. This new functionality is different than the functionality in Flexsim version 2.6 and earlier, which would call the place in bay and place in level functions both when a transport requested it as well as when the flowitem entered.

### Flowitem Placement

If the rack is a vertical storage rack, the flowitems entering the rack will be placed in their given bay and level and against the y fringe of the rack (`yloc(rack) - ysize(rack)`). They will stack backwards into the rack from that point. If the rack is being used for

floor storage, then flowitems will be place in their given bay and level on the floor, and will stack vertically from that point.

## Visualization

The rack can have several viewing modes to better see products in the rack. In addition to the opacity value, you can hold the 'X' key down and repeatedly click on the rack, and the rack will toggle through different viewing modes. These modes are listed as follows.

1. Full Draw Mode: This mode shows each cell as a level with a platform to put flowitems on. This is the realistic representation of the rack.
2. Back Face Draw Mode with Cells: This mode shows only the back faces of the rack, so that you always see into the rack. It also draws a grid on the back face representing the bays and levels of the rack. This allows you to better view items in the rack, as well as which bays and levels the items are on.
3. Back Face Draw Mode: This mode is like the previous mode, except that the grid of bays and levels is not drawn. This mode allows you to easily view items in the rack.
4. Wire Frame Draw Mode: This mode draws a wire frame around the shape of the rack. This mode allows you to see flowitems in several back-to-back racks. Once a rack is in this draw mode, you will need to 'X' click on the actual wire frame in order to toggle back to mode 1.

## Commands

There are several commands you can use to query information on the bays and levels of a rack. These commands are as follows. For more detailed information, refer to the command summary.

**rackgetbaycontent(obj rack, num bay)** This command returns the total number of flowitems that are in the given bay.

**rackgetbayofitem(obj rack, obj item)** This command returns the bay number that the flowitem is in.

**rackgetcellcontent(obj rack, num bay, num level)** This command returns the number of flowitems that are in a given bay and level.

**rackgetitembybayandlevel(obj rack, num bay, num level, num itemrank)** This command returns a reference to an item in a given bay and level.

**rackgetlevelofitem(obj rack, obj item)** This command returns the level number that the flowitem is in.

**rackgetnrof bays(obj rack)** This commands the total number of bays the rack has.

**rackgetnrofllevels(obj rack [,num bay])** This command returns the number of levels in the given bay of the rack.

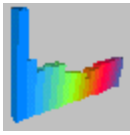
## States

The Rack doesn't implement any states. Use the content graph to get statistics.

## Parameters tab-pages

Rack  
SizeTable  
Flow  
RackTriggers

## Recorder



### Overview

Recorders are used to record and/or display information graphically in your model. More specifically the Recorder used to capture Table Data, Standard Data, and User-defined Data. All data types can be displayed graphically in you model and written to tables within Flexsim for export to Excel, Access, or any ODBC database. Since the Recorder object is a powerful data presentation tool, an example of how to use the Recorder is found in the tutorials section of this manual.

### Details

Refer to the Recorder Parameters page for more information on the Recorder.

## Reservoir



### Overview

The Reservoir is used to store flowitems as if they were in a fluid reservoir or tank. The flow rate in and out of the reservoir can be defined by the user. Events can occur when the level in the reservoir rises or falls to certain values that the user can define.

### Details

The Reservoir is a sub-class of the FixedResource. It continues to receive flowitems until its volume reaches its maximum content value. When each flowitem enters, it first executes the incoming flow rate function, which sets the flowitem's volume as well as returns the time before receiving the next flowitem. If there is still room to receive another flowitem, the Reservoir creates an event in the time returned by the function to receive the next flowitem. Then, if the flowitem is the first flowitem in the Reservoir, it calls the outgoing flow rate function, and creates an event for that returned time, after which it releases the flowitem.

Whenever a flowitem leaves, the Reservoir first checks if it was previously full. If it was full, and now has room to receive another flowitem, it calls the incoming flow rate function, and creates an event in that time to receive the next flowitem. Then, if there is another flowitem in the Reservoir, it calls its outgoing flow rate function, and creates an event in the returned time to release the next flowitem.

### States

States for the Reservoir are defined based on the last thing that happened to the Reservoir.

**Empty:** There are no flowitems in the Reservoir.

**Collecting:** The last thing that happened to the Reservoir was an entry event, and there is still room to receive more products.

**Blocked:** The last thing that happened to the Reservoir was an entry event, and the Reservoir is full.

**Releasing:** The last thing that happened to the Reservoir was an exit event, and there are still flowitems left to release.

### Parameters tab-pages

Reservoir  
Flow  
ReservoirTriggers

## Robot



### Overview

The robot is a special transport that lifts flowitems from their starting locations and places them at their ending locations. Generally, the robot's base does not move. Instead, the arm rotates as it carries the flowitems. The arm on the robot has two segments that move to reach either the flowitem being moved or its destination. The length of these arms can be set by the user. The speed at which the robot rotates and extends the arm can also be set by the user.

### Details

The robot is a sub-class of the TaskExecuter. It implements offset travel by extending its arm to the destination location. Note that its x/y/z location of the robot does not change at all when it does offset travel. Only its z and y rotations and arm extension change as it travels to the destination. If the destination location is further away than the robot's maximum arm extension, then the robot will only extend its arm up to its maximum extension. The robot uses its z, y rotation speeds and arm extension to get to the destination for offset travel. The travel time for an offset is the maximum of the time to extend its arm, the time to rotate around the z-axis, and the time to rotate around its y-axis. It does not use the standard TaskExecuter maximum speed, acceleration and deceleration values.

The robot by default does not connect itself to a navigator, which means that it does not execute travel tasks unless you explicitly connect it to a network.

### States

The Robot follows TaskExecuter states.

### Parameters tab-pages

- Robot
- TaskExecuter
- Collision
- Dispatcher
- TaskExecuterTriggers

## Separator



### Overview

The separator is used to separate a flowitem into multiple parts. This can either be done by unpacking a flowitem that has been packed by a combiner or by making multiple copies of the original flowitem. The splitting/unpacking is done after the process time has completed. The separator can be set to require operators during its setup, processing and repair times.

### Details

The separator is a sub-class of the Processor, which is in turn a sub-class of the FixedResource. It receives one flowitem, then executes the setup and process times for that flowitem. If the separator is in unpack mode, then once the setup and process times are finished, the separator moves the unpack quantity out of the flowitem and into itself. Then it releases all flowitems that it unpacked. Once all unpacked flowitems have left the separator, it releases the container flowitem. If the separator is in split mode, once the setup and process times are finished, the separator duplicates the flowitem so that the resulting total number of flowitems is the split quantity. Then it releases all of the flowitems. For both unpack and split modes, once all flowitems have left the separator, the separator immediately receives the next flowitem.

**Note on the split/unpack quantity:** This quantity value has subtle differences for unpack mode versus split mode. When in unpack mode, the separator unpacks the exact number of flowitems specified by this quantity. This means the total number of flowitems that result is one more than the unpack quantity (the unpack quantity + the container flowitem). When in split mode, however, the separator duplicates the flowitem split quantity - 1 number of times. This means the total number of flowitems that result is the exact same as the split quantity.

**Note on unpacking order:** When the Separator is in unpack mode, it unpacks the container flowitem from **back-to-front**, meaning it pulls the last item in the container out first, then the second to last, and so on. If you want flowitems to be unpacked out in a certain order, then set their rank in the entry trigger.

### States

**Idle:** The object is empty.

**Setup:** The object is in its modeler-defined setup time.

**Processing:** The object is in its modeler-defined process time

**Blocked:** The object has released its flowitems, but downstream objects are not ready to receive them yet.

**Waiting for Operator:** The object is waiting for an operator to arrive, either to repair a breakdown, or to operate on a batch.

**Waiting for Transport:** The object has released a flow item, and a downstream object



is ready to receive it, but a transport object has not picked it up yet.

**Down:** The object is broken down.

### Parameters tab-pages

ProcessTimes

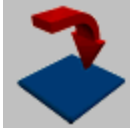
Separator

Flow

ProcessorTriggers

Operators

## Sink



### Overview

The sink is used to destroy flowitems that are finished in the model. Once a flowitem has traveled into a sink, it cannot be recovered. Any data collection involving flowitems that are about to leave the model should be done either before the flowitem enters the sink or in the sink's OnEntry trigger.

### Details

The sink is a sub-class of the FixedResource. It receives flowitems always, and either destroys them as soon as they enter, or recycles them back to the flowitem bin. Since it destroys all flowitems it receives, the Sink does not have any sendto logic in its Flow tab.

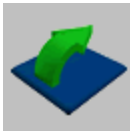
### States

The Sink does not implement any states. Refer instead to its input statistics.

### Parameters tab-pages

- Sink
- Flow
- SinkTriggers

## Source



### Overview

The source is used to create the flowitems that travel through a model. Each source creates one class of flowitem and can then assign properties such as itemtype or color to the flowitems it creates. Models should have at least one source in them. Sources can create flowitems per an inter-arrival rate, per a scheduled arrival list, or simply from a defined arrival sequence.

### Details

The source is a subclass of the FixedResource, although it does not receive flowitems. Instead it creates the flowitems and releases them. Hence it has no input section in its flow parameters page. The source can operate in one of three possible modes:

- Inter-Arrivaltime Mode:** In inter-arrivaltime mode, the source uses its inter-arrivaltime function. This function's return value is the time to wait till the next arrival of a flowitem. The source waits this amount of time, then creates a flowitem and releases it. Once the flowitem has exited, it calls the inter-arrivaltime function again and repeats the cycle. Note that the inter-arrivaltime is defined as the time between the exit of one item and arrival of the next item, not as the time between the arrival of one item and the arrival of the next item. If you would like to make the inter-arrivaltime be defined as the true time between arrivals, then use a downstream queue with a large capacity to make sure that the source immediately sends on flow items when they are created. You can also specify whether the inter-arrivaltime should be executed for the first arrival, or if the first flowitem should be created at time 0.
- Arrival Schedule Mode:** In arrival schedule mode, the source follows a schedule defined by the user in the schedule table. Each row of the table specifies an arrival of flowitem(s) at a given time in the simulation. For each arrival entry, you can specify the arrival time, name, itemtype, number of items to create, and additional item labels for that arrival. The arrival times should be ordered correctly in the schedule table, meaning each entry's arrival time should be greater than or equal to the previous entry's arrival time. If the source is set to repeat the schedule, then it will immediately loop back on the first arrival after the last arrival, causing the first entry's arrival to happen at the exact same time as the last entry's arrival. Note here that, when repeating the schedule, the first entry's arrival time only applies to the very first loop through the schedule. This allows you to have an initial arrival time that is executed once, but not repeated. If you would like the source to wait a given amount of time between the last arrival and the repeated first arrival, then add an extra entry to the end of the table, give it an arrival time greater than the previous entry's arrival time, but have 0 flowitems arrive for that new entry.
- Arrival Sequence Mode:** Arrival sequence mode is like arrival schedule mode, except that there is no time associated with the arrivals. The source will create the flowitems for a given table row, and then as soon as the last flowitem for that entry has exited, it will immediately go to the next row in the table. You can also repeat the arrival sequence.

## States

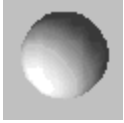
**Generating:** There are no flowitems in the Source. It is waiting until its next creation event to create flowitems.

**Blocked:** Flowitems have been created and are waiting to leave the source.

## Parameters tab-pages

Source  
Flow  
SourceTriggers

## TaskExecutor



### Overview

TaskExecutor is the top level class for several objects in the library. Operators, Transporters, ASRSvehicles, Cranes and other mobile resources inherit from the TaskExecutor class. All of these objects can travel, load flowitems, unload flowitems, act as shared resources for processing stations, and perform many other simulation tasks.

To learn more on how to begin using TaskExecutors, refer to Lesson 2 of the Tutorial.

### Details

TaskExecutors and their sub-classes are able to execute task sequences, perform collision detection, and execute offset travel.

The TaskExecutor is also a sub-class of the Dispatcher class, and thus a TaskExecutor can actually act as a team leader that dispatches task sequences to other team members. Its handling and dispatching logic have subtle differences from the Dispatcher, however. When the TaskExecutor receives a task sequence, it first checks to see if it already has an active task sequence. If there is no active task sequence, or if the newly received task sequence is preempting and has a priority greater than the currently active task sequence, then it will start executing the new task sequence, preempting the active one if needed. Otherwise it will go through the normal Dispatcher logic. If the task sequence is not passed on immediately, then it will queue up in the TaskExecutor's task sequence queue, and if the task sequence is still in the queue when the TaskExecutor finishes its active task sequence, the TaskExecutor will then execute the task sequence.

### User-Defined Parameters

All TaskExecutors have the following fields that can be defined by the modeler.

**Capacity:** This parameter defines a value for the maximum content of the object. In default operation, the object will never load more flowitems than this value specifies.

**Note for advanced users on the capacity value:** this value can be deceiving if you create your own task sequences. Since the TaskExecutor's first and most important responsibility is to execute task sequences, if you give the object a task sequence to load more flow items than its maximum content, then it will load the flowitems anyway. The only real instance in which the maximum content value is used is for the TASKTYPE\_BREAK task. If the TaskExecutor comes to a break task, and it has reached its maximum content, then it will not perform the break, and will continue on its current task sequence instead of breaking to another task sequence. This works in the default case when task sequences are created automatically because each task sequence is responsible for the loading of just one flowitem.

**Maximum Speed, Acceleration, Deceleration:** These define the TaskExecutor's maximum speed, acceleration, and deceleration. Maximum speed is defined in units of length per unit of time, while acceleration and deceleration are defined in units of length per squared unit of time. If you are defining your model in meters and seconds, for example, the speed value is in m/s, etc. These values are used in defining the object's peak speed and change in speed while executing the task types such as TASKTYPE\_TRAVEL, TASKTYPE\_TRAVELTOLOC, etc.

**Travel Offsets for load/unload tasks:** This value determines whether the TaskExecutor should execute offset travel to the load/unload location when it loads or unloads a flowitem. For example, if this is not checked, and the TaskExecutor is traveling on a network, then it will only travel to the network node that is at the load/unload station. It will remain on that node while doing the load.

**Rotate while traveling:** Here you can specify if you want the object to rotate in the direction that it is traveling. This will have no effect on model output. It is only for visual purposes.

**Load Time:** This field is executed at the beginning of each load task. Its return value is the delay time that the TaskExecutor will wait before loading the flowitem and moving on to the next task. Note that if the TaskExecutor is configured to travel offsets for load/unload tasks, then it will first travel the correct offset, and then start the load time. Thus the load time is added onto the end of the offset travel time; it is not part of the offset travel time.

**Unload Time:** This field is executed at the beginning of each unload task. Its return value is the delay time that the TaskExecutor will wait before unloading the flowitem and moving on to the next task. Note that if the TaskExecutor is configured to travel offsets for load/unload tasks, then it will first travel the correct offset, and then start the load time. Thus the load time is added onto the end of the offset travel time; it is not part of the offset travel time.

**Break to Requirement:** This field is executed when the TaskExecutor comes to a break task or callsubtasks task. The return value is a reference to a task sequence. The logic within this field should search the TaskExecutor's task sequence queue, and find a task sequence that is appropriate to break to.

## Parameters tab-pages

TaskExecutor  
Collision  
Dispatcher  
TaskExecutorTriggers

## States

The TaskExecutor's states are purely dependent on the types of tasks that the TaskExecutor performs. Many tasks are associated with a hard-coded state, but with some tasks the modeler can specify an explicit state for the TaskExecutor to be in when executing that task. Here are some of the states that you will see often with a TaskExecutor. For more information on tasks and task sequences, refer to Task Sequences.

**Travel Empty:** The object is traveling to a destination object and doesn't contain any flowitems. This state is exclusively associated with the TASKTYPE\_TRAVEL task.

**Travel Loaded:** The object is traveling to a destination object and has load one or more flowitems. This state is exclusively associated with the TASKTYPE\_TRAVEL task.

**Offset Travel Empty:** The object is performing offset travel and doesn't contain any flowitems.

**Offset Travel Loaded:** The object is performing offset travel and has loaded one or more flowitems.

**Loading:** The object is loading a flowitem. This state corresponds to the TASKTYPE\_LOAD task, and applies only to the time when the object has finished its offset travel and is waiting its modeler-defined load time before loading the item.

**Unloading:** The object is unloading a flowitem. This state corresponds to the TASKTYPE\_UNLOAD task, and applies only to the time when the object has finished its offset travel and is waiting its modeler-defined unload time before unloading the item.

**Utilize:** The object is being utilized at a station. This state is usually used for an operator, when the operator has arrived at the station and is being utilized for a process, setup, or repair time. The utilize state is usually associated with a TASKTYPE\_UTILIZE task, but that task can also specify a different state. Also, other task types, like TASKTYPE\_DELAY, can use the utilize state.

## Offset Travel

Offset travel is a mechanism by which different types of objects can travel differently, yet use the same interface for traveling. For example, an object wants to place an item into a given bay and level of a Rack. The way in which the object travels to the correct location to drop off the item depends on the type of object it is. An operator walks to the bay's location and places the item in the level. A Transporter travels to the bay, but must also lift its fork to the proper height of the level. It can travel in both the x and y direction, but only its fork can travel in the z direction. An ASRSvehicle will only travel along its own x axis, lifting its platform to the height of the level, and then pulling the item from the Rack. Hence, each object implements its travel differently, but the interface is the same: travel to the right spot to place the item in the Rack. Offset travel is essentially the only thing that distinguishes each sub-class of the TaskExecutor. For information on how each sub-class implements offset travel, refer to the "Details" section of an object's help page. Offset travel is used in load and unload tasks, in `traveltoloc` and `travelrelative` tasks, and in `pickoffset` and `placeoffset` tasks.

The offset travel interface is very simple. Every type of offset request translates into an x,y, and z offset distance, and sometimes a reference to a flow item. For example, if an object is given a `traveltoloc` task for the location (5,0,0), and its current location is (4,0,0), then it automatically translates that task into an offset request for (1,0,0), meaning it needs to travel one unit in the x direction. A `travelrelative` task translates directly. For example, a `travelrelative` task for (5,0,0) tells the object to travel 5 units in the x direction. Load and unload tasks also use offset travel if the "Travel Offsets for Load/Unload Tasks" checkbox is checked. When an object needs to load a flowitem from a station, it queries the station for the location of the item. Also, when it needs to unload an item, it queries the station for the location to unload the item. The station returns an offset x/y/z distance, and the TaskExecutor uses this distance to

travel its offset. Also, for a load and unload task, the TaskExecutor has a reference to the item in its offset request. This may or may not affect the way the object travels, depending on the type of object. For example, the Transporter's offset travel mechanism is implemented so that if there is an item, or in other words, if the Transporter is loading or unloading an item, the Transporter will lift its fork in the z direction. If there is no item, or in other words, if the Transporter is doing a `traveltoloc` or `travelrelative` task, then it will actually travel in the z direction, instead of lifting its fork.

Offset values should be calculated relative to the x/y center of the object, and the z base of the object. For example, a robot is positioned at location (0,0,0). It has a size of (2,2,1). From this the x/y center and z base can be calculated as the location (1,-1,0) (Note: y size extends along the negative y-axis). All offset calculations should be made from this (1,-1,0) location. While giving the robot a `traveltoloc` task will automatically make this calculation for you, sometimes it is necessary to calculate this location manually and use a `travelrelative` task. If the robot is given a `travelrelative` task of (1,0,0), this means that the correct position to travel to is one unit to the right of the robot's x/y center and z base. This translates to the location (2,-1,0). Note that this does not mean that the robot will travel such that its own location is (2,-1,0). Neither will it travel such that its x/y center and z base are at that location. Because it is a robot, it will rotate and extend its arm so that the end of the arm is at the location (2,-1,0). Its actual location will not change at all. Thus the calculation from the object's x/y center and z base allows you to specify a desired destination location which is the same for all objects, but which allows each type of object to handle that destination location differently.

## Collision Detection

The TaskExecutor and its sub-classes have the capability of detecting collisions with other objects. Collision detection is performed by adding collision members to a TaskExecutor, then adding collision spheres to it and its collision members, then executing logic when one of the spheres of the TaskExecutor collides with one of the spheres of one of its collision members. Each collision sphere you specify has a location in the TaskExecutor's frame of reference, and a radius. The TaskExecutor repetitively checks for collisions at time intervals that you specify. At each collision check, the TaskExecutor checks for collisions on all of its collision spheres with all of the collision spheres of all of its collision members. If a collision is found, then the TaskExecutor fires its collision trigger. It does not fire the collision trigger of the object with whom it is colliding. The other object's collision trigger will fire if and when it does its own collision checks. Note that the collision trigger fires for a specific sphere-to-sphere collision. This means that within one collision check the collision trigger can fire several times, once for each sphere-to-sphere collision encountered in the check.

Be aware that you can very easily cause your model execution speed to decrease significantly if you are not careful with collision detection. For example, if a TaskExecutor has 5 collision spheres and 5 collision members, and each of its collision members has 5 collision spheres, then each collision check will need to check for 125 sphere-to-sphere collisions. If all 6 TaskExecutors are checking for collisions, then 750 sphere-to-sphere checks are being made at each collision check



interval of the model. This can slow the model down considerably, especially if your collision interval is very small.

You can turn collision detection on and off for a given TaskExecuter by using the `setcollisioncheck()` command. See the command summary for more information on this command.

## TrafficControl



### Overview

The TrafficControl is used to control traffic in a given area of a travel network. You build a traffic controlled area by connecting NetworkNodes to the traffic control object. These NetworkNodes then become members of the traffic controlled area. Any path between two nodes that are both members of the same traffic control object is designated as a traffic controlled path, and travelers are only allowed onto that path if given permission by the traffic control object. The traffic control object can be in a mutual exclusion mode, where it only lets a certain number of travelers into the area at any given time, or it can use un-timed traffic modes to only allow travelers onto certain path sections at once.

### Details

To connect a NetworkNode to a traffic control, hold down the 'A' key and drag from the traffic control to the node. This will draw a red line between the node and the traffic control. If two NetworkNodes have a path between them, and both NetworkNodes are members of the same traffic control object, then that path is designated as a traffic controlled path, or a member path.



All travelers entering a traffic controlled area must get permission from the traffic control. A traffic control's area consists of all traffic controlled paths as well as the NetworkNode members themselves. What this means is, "entering" a traffic control area is defined as either entering a path that is a member of the traffic controlled area, or arriving at a final destination whose NetworkNode is a member of the traffic control area. However, a traveler is not considered entering the area if it is passing over a NetworkNode that is a member of the traffic controlled area and continuing on a path that is not a member of the traffic controlled area. Travelers will not need permission in this case. A traveler "exits" a traffic controlled area in one of two ways. Either the traveler is coming from a member path to a path that is not a member of the traffic controlled area, or the traveler is coming from an "inactive" state at a member NetworkNode and continuing on to a path that is not a member of the traffic controlled area. Whenever a traveler exits a full area, room is created for other travelers to enter the area. You can also have an inactive traveler exit a traffic controlled area by calling `reassignnetnode()`, assigning the traveler to a NetworkNode that is not a member of the area. The table below shows the entry/exit definitions.

### Traffic Control Area Entry/Exit Criteria

	<b>Coming From</b>	<b>Continuing To</b>
<b>Area Entry</b>	Non-member path	1. Member path OR 2. Member final destination node
<b>Area Exit</b>	1. Member path OR 2. Inactive state at member node	Non-member path

The traffic control object screens entries into its area using one of two modes: mutual exclusion or un-timed traffic modes.

### Mutual Exclusion

When the traffic control is in mutual exclusion mode, it will only allow a certain number of travelers into its area at any given time. Once the area is full, travelers requesting to enter must wait at the edge of the traffic controlled area until another traveler leaves the area and frees up room.

### Un-Timed Traffic Modes

When the traffic control is using un-timed traffic modes, it screens entries into the traffic control area based on its table of modes and the entry path of travelers requesting to enter the area. Each row in the mode table represents one mode. Each mode includes a set of entry paths that the traffic control will allow entry for.

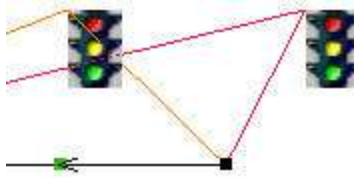
The traffic control selects modes based on entry requests. If there are no travelers in the area, then the traffic control simply waits. When the first traveler requests to enter a certain member path of the area, the traffic control searches its table to find a mode that includes that path. If found, the traffic control goes into that mode, allowing the traveler into the area. Once the traffic control is in a certain mode, it will stay in that mode until all travelers have exited the area. Then it will wait for the next "first traveler" request, and will repeat the cycle.

### Dynamically Changing Modes

If the traffic control is set to search for the best mode, then it may change modes dynamically without emptying the area first. The traffic control keeps a record of which paths in the current mode have been traveled on. When a traveler enters on a path, the traffic control marks that path as "dirty" or "traveled on" so to say. Even if the traveler leaves the area later on, the dirty record remains. The record is only reset when the traffic control area is totally emptied. When a traveler requests to enter on a path that is not a member of the current mode, the traffic control searches the rest of the table to see if there is another mode that includes all paths that are currently dirty as well as the path that the traveler is requesting. If it finds one then it changes to that mode and allows the traveler into the area. Otherwise the traveler must wait.

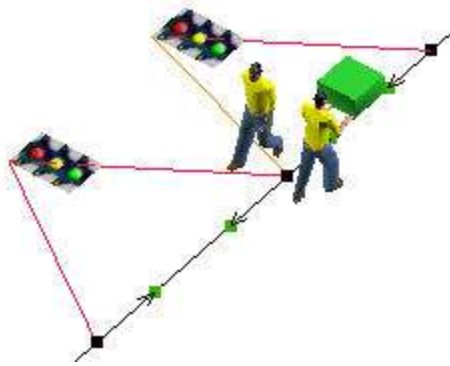
### Using Several Traffic Controls

Each NetworkNode can be connected to up to 50 traffic control objects simultaneously. The figure below shows a network node connected to two traffic controls.



Notice that the line drawn from the node to the traffic control on the left is orange, whereas the line to the traffic control on the right is red. These colors show the ordering of the traffic controls in the node's member list. The color ordering is done in ROYGBIV fashion (red, orange, yellow, green, blue, indigo, violet). The first traffic control for that network node has a red line drawn to it, the second an orange line, and so forth. This ordering is very important to the functionality of the model, as well as in avoiding gridlock. When a traveler comes to a network node where it must enter more than one traffic control area, it will request entry into the traffic control areas in the order that those areas are listed on the node. It will only request one traffic control at a time. Once a traffic control gives permission to enter, the traveler has technically entered that traffic control area, even though it may still require permission for entry into other traffic control areas. When transferring between two paths, a traveler will first enter all traffic control areas corresponding to the new path before exiting traffic control areas corresponding to the old path.

**Note on gridlock when using several traffic controls:** Although using several traffic controls in your model can increase the flexibility and capability of your travel networks, it can also very easily cause gridlock. Traffic control gridlock is usually caused by circular wait. Circular wait happens when one traveler is in a traffic control area that is full, waiting to enter another traffic control area, but the other traffic control area is also full and waiting for a second traveler to exit, but the second traveler cannot leave because it must first enter the area that the first traveler is in. This is a simple example of circular wait. Circular wait can span several travelers and traffic controls and is very difficult to debug. Hence, be very careful when using several traffic controls. Experience has shown that it is beneficial to hierarchically order traffic controls, with some traffic controls in charge of large areas, and some traffic controls in charge of smaller areas. Partial intersection of traffic control areas is strongly discouraged. Either a traffic control should be completely contained within a larger traffic control area, or should be the only traffic control for its area. Travelers should request entry into larger areas before requesting entry into smaller areas. Still, even following these guidelines exactly does not guarantee that a model will not experience gridlock. Much of this type of model building is simply trial and error. The figure below shows a very simple model that still causes gridlock. Notice that the green traffic control on the left is full by the fact that the operator holding the flowitem is in its area, waiting to get permission to enter the blue traffic control's area on the right. But the blue traffic control's area is also full because the operator with no flowitem is in the area waiting to get into the green traffic control area on the left.



## Interacting with Traffic Controls

You can do several things to edit traffic control objects in the orthographic/perspective views. Hold down the 'X' button, and repeatedly click on a traffic control object, and all traffic control objects in the model will toggle between hiding their node connections and showing them. If you only want to hide one traffic control's connections, select the traffic control using the shift key, and then 'X' click on the object and it will hide its connections. While running your model, you can hold the 'V' key down and click on the object, holding the mouse button down. Holding the 'V' key down operates the same as described in the keyboard interaction section. This will draw a line to all travelers that are currently requesting entry to the traffic control area, but have not been given permission yet. If the traffic control's color is not white, then it will draw these lines in its color, allowing you to better distinguish between different traffic control area entry requests.

## Resetting a Model with Traffic Controls

Currently traffic controls are not configured to correctly handle travelers that are placed within a traffic control area when the model is reset. You will need to have all travelers reset to a `NetworkNode` that is not a member of any traffic control areas. Usually this means adding a `NetworkNode` off to the side of your model, 'A' connecting all `TaskExecutors` to that `NetworkNode`, and then having them enter the model at the beginning of each simulation run.

## Manipulating Speeds with Traffic Controls

You can also use Traffic Controls to modify speeds of travelers as an area becomes more crowded. As the traffic control's content increases, entering travelers will modify their speeds based on the Traffic Control's speed table. For example, if you have entered a row in the table that is a 0.6 speed multiple for a content of 3, then as soon as the content of the traffic control's area becomes 3 or greater, all travelers' max speeds will be decreased to 60% of their normal max speed. Note that the speed change does not take effect until the traveler reaches its next node in the network. If you have an area with multiple traffic controls, then the minimum speed multiple of all of the traffic controls will be applied.

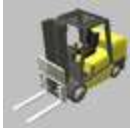
## States

The traffic control does not implement states at this time.

## Parameter tab-pages

Traffic Control

## Transporter



### Overview

The transporter is used mainly to carry flowitems from one object to another. It has a fork lift that will raise to the position of a flowitem if it is picking up or dropping off to a rack. It can also carry several flowitems at a time if needed.

### Details

The transporter is a sub-class of the TaskExecuter. It implements offset travel in two ways. First, if there is an involved flowitem for the travel operation, then it travels so that the front of the fork lift is positioned at the destination x/y location and raises its fork to the z height of the destination location. Second, if there is no involved flowitem for the offset travel operation, then it travels so that its x/y center and z base arrive at the destination location.

### States

The transporter follows TaskExecuter states.

### Parameter tab-pages

- TaskExecuter
- Collision
- Dispatcher
- Transporter
- TaskExecuterTriggers

## VisualTool



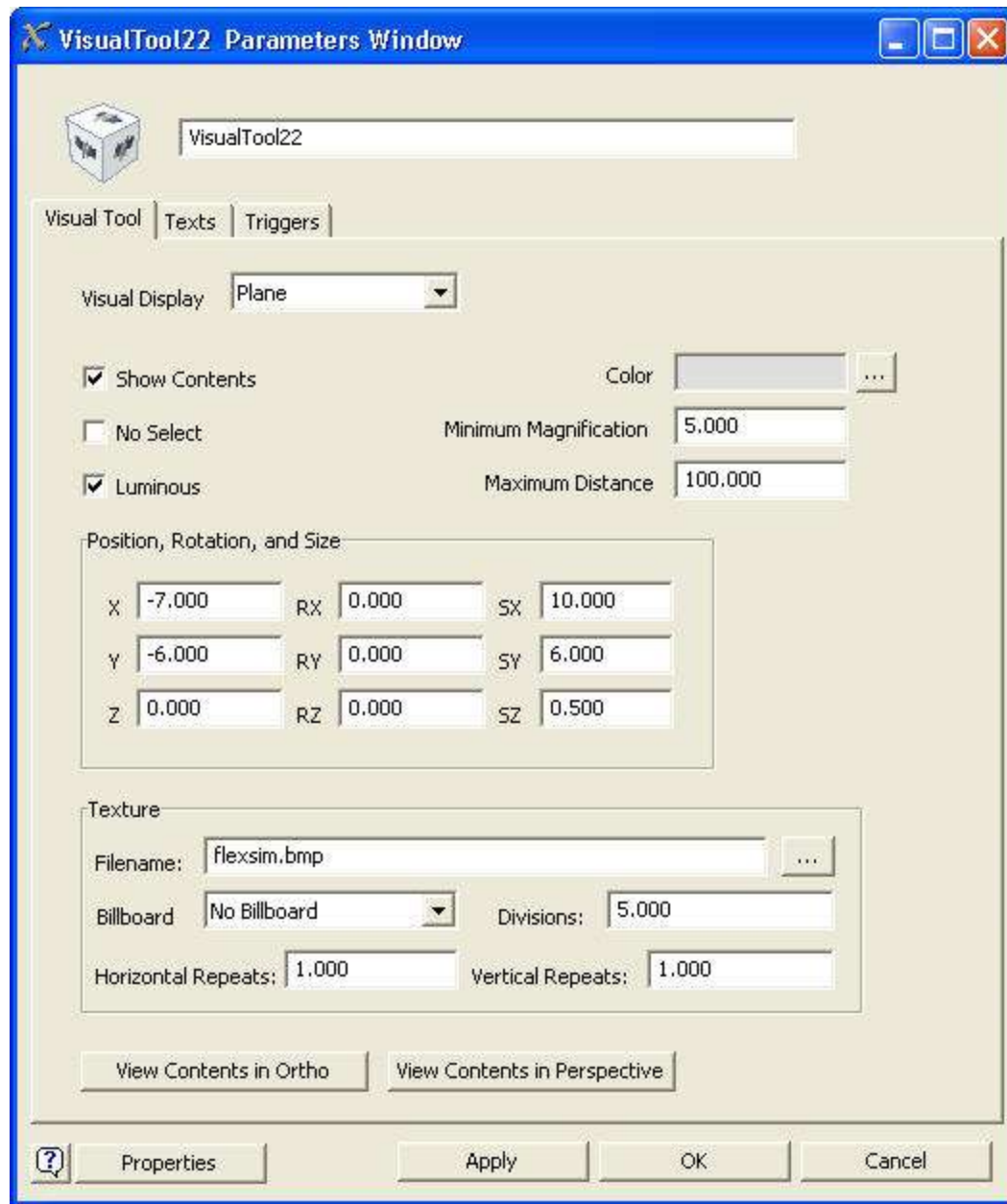
### Overview

VisualTools are used to decorate the model space with props, scenery, text, and presentation slides in order to give the model a more realistic appearance. They can be something as simple as a colored box, background, or something as elaborate as an imported 3D graphic model, presentation slide. Another use of the VisualTool is as a container for other objects in your model. When used as a container, the VisualTool becomes a handy tool for hierarchically organizing your model. The container can also be saved in a User Library acting as a basic building block in the development of future models.

### Details

There are many ways you can use the VisualTool object in your model.

- As a container or submodel
- As a plane, cube, column or sphere
- As an imported shape
- As text
- As a presentation slide
- Other settings



The VisualTool replaces the VisualObject, and the VisualText in previous versions of Flexsim. The VisualTool plays a more expanded role than the VisualObject, and VisualText. The VisualTool now serves as a container to encapsulate sub-models for hierarchical modeling structure. Since the VisualTool works differently than other Flexsim objects, a detailed description of how it is used will now be explained.

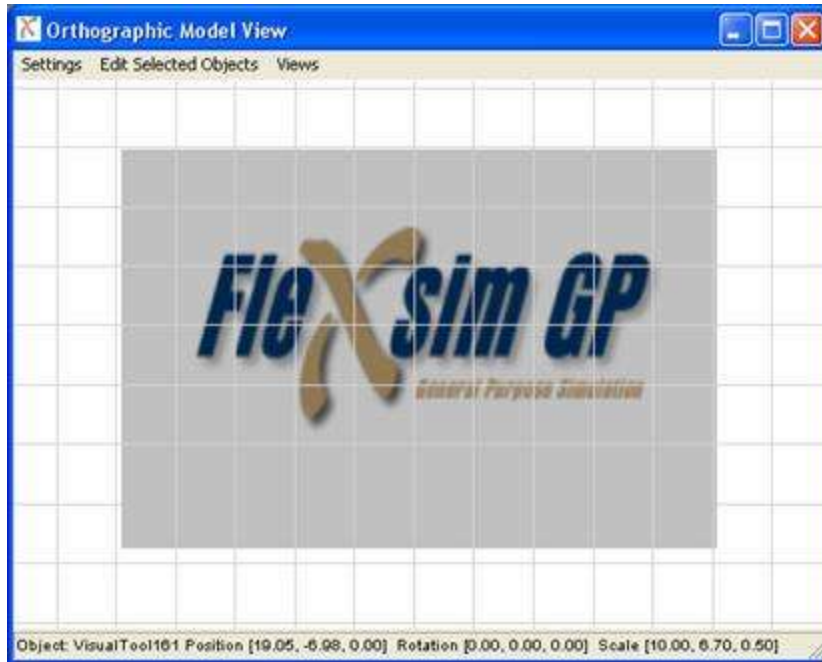
### Using the VisualTool as a Container

The default setting for the VisualTool is a plane. When placed in the model the VisualTool will display a plane with a Flexsim GP bitmap texture. The size and location of the plane can be set graphically in the Orthographic or VR model view window, or by using the VisualTool parameters tab (the use of the parameters tab is explained in the section Using the Visualtool as a Plane, Cube, Column, or Sphere).



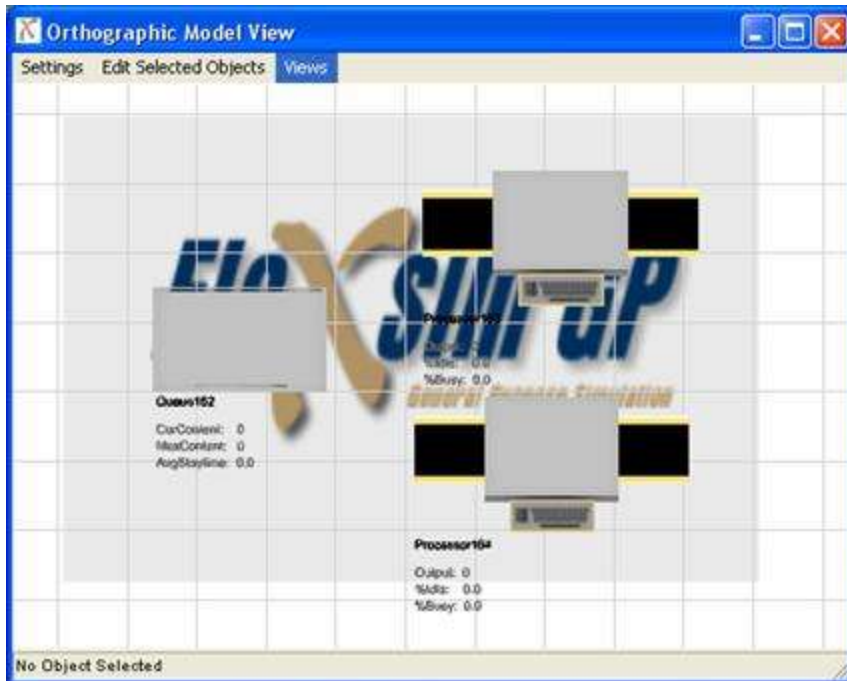
When using the VisualTool as a container it is recommended that you use the default view (a Plane) setting when you start. You can change the visual representation at a future time. For this example we will build a container that has 1 queue, and 2 processors. Flowitems will be passed into the container from a Source that is located outside. The processors will send flowitems to a sink located outside the container.

### Step 1: Place a VisualTool into the model view



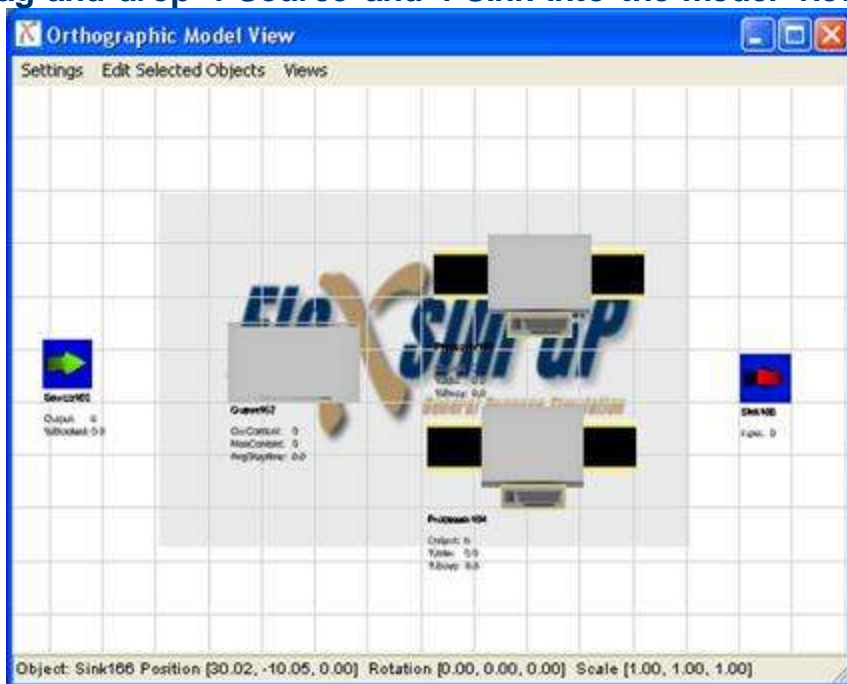
The VisualTool will be displayed with the Flexsim GP bitmap texture. To add objects inside the container simply drag them from the library and place them on the VisualTool object.

### Step 2: Drag-and-drop 1 Queue and 2 Processors into the VisualTool



When you place an object on the VisualTool it will automatically be placed inside the VisualTool object. You can test this by selecting the VisualTool and moving its location with your mouse. When you move the VisualTool the objects inside will move as well.

### Step 3: Drag-and-drop 1 Source and 1 Sink into the model view



When placing the Source and the Sink in the model make sure you do not place them on the VisualTool, you want to make sure they are outside.

Before you make the port connections for this example it might be helpful to enlarge the port connections graphically to help you understand the 2 methods that can be used when working with containers.

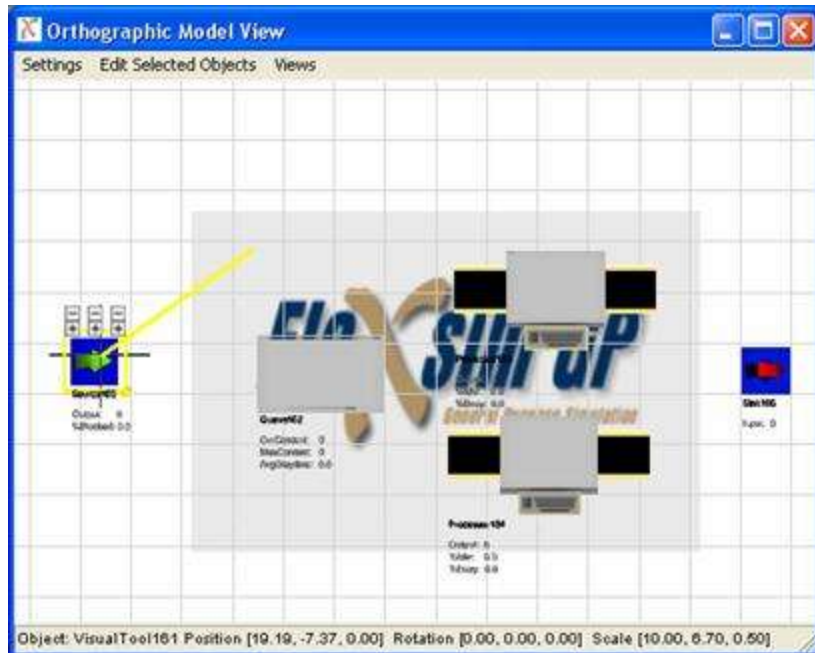
#### Step 4: Enlarge the port connections graphically

This is done by selecting the Settings menu on the Model View window. This will bring up the following input page:

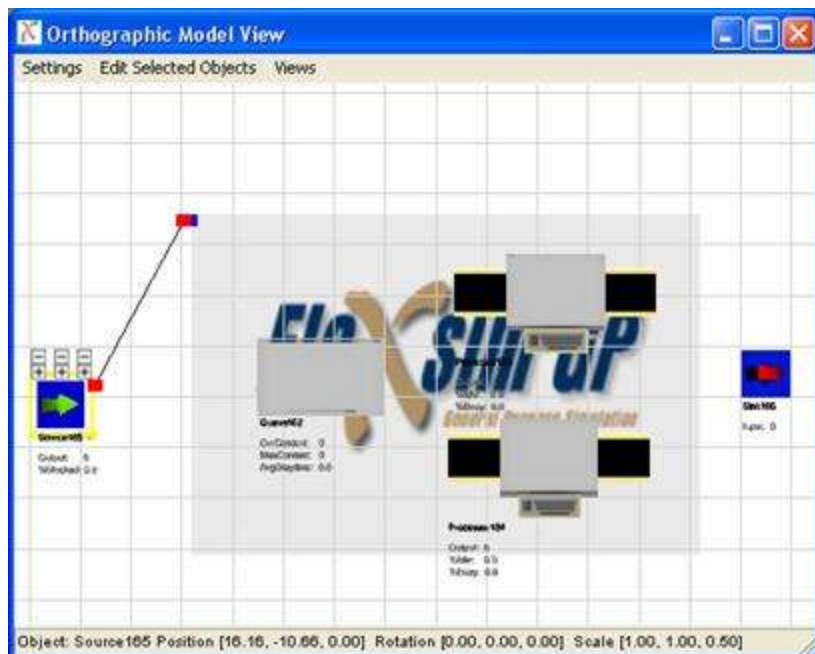


Set the Connector size to 0.30. Then select the OK button.

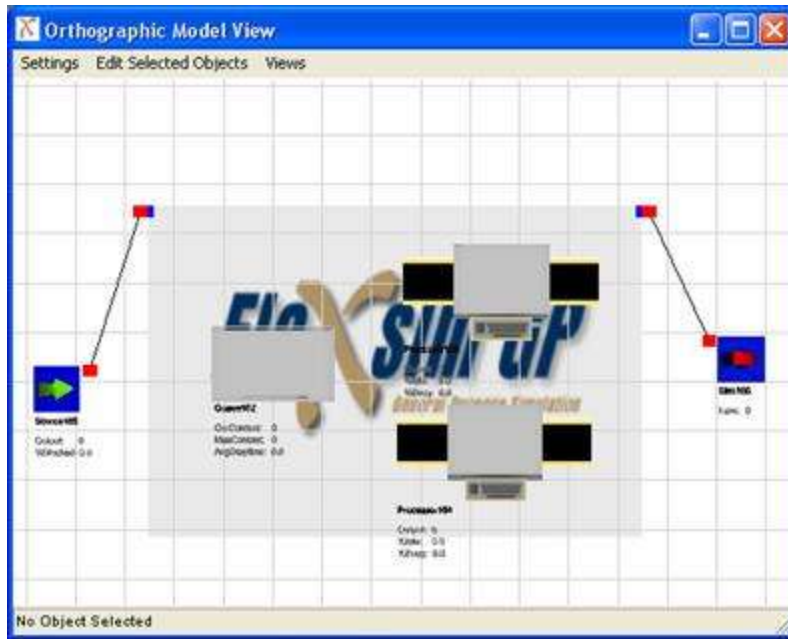
#### Step 5: Connect to Source to the VisualTool, and the VisualTool to the Sink



While pressing the "A" key on the keyboard, click-and-drag a connection from the Source to the VisualTool (not the Queue). When you release the left mouse button you will see a connection made between the Source and the VisualTool as shown below.



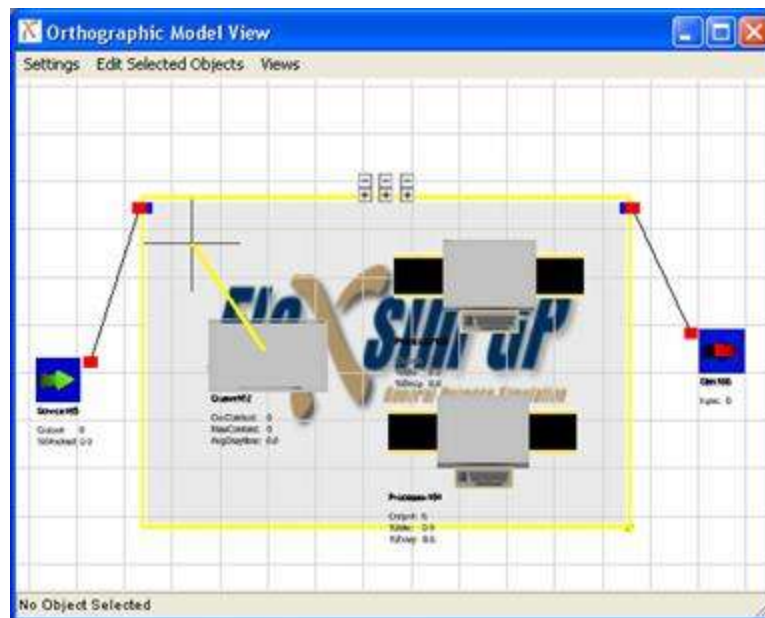
Now make a connection from the VisualTool to the Sink as shown below.



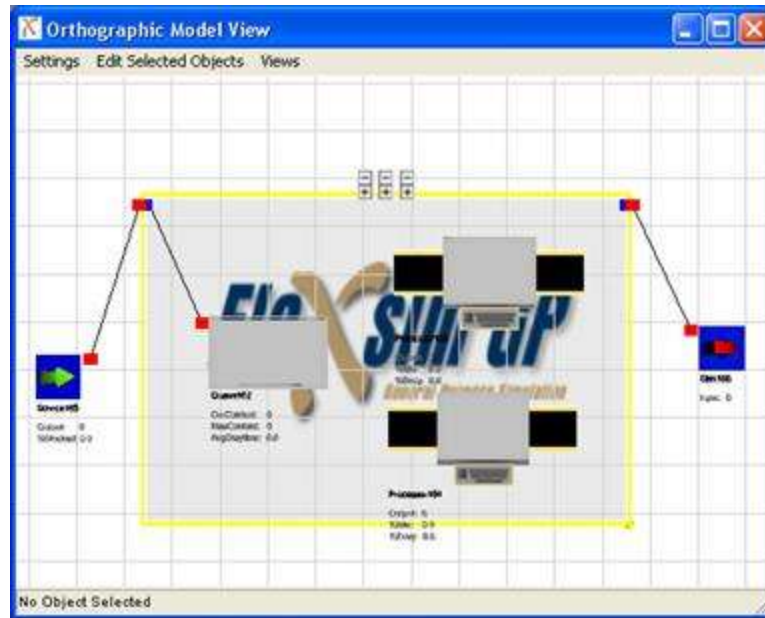
At this point the Source and the sink are connected to the container (VisualTool).  
Now we will connect the container to the Queue.

### Step 6: Connect the Container to the Queue

Drag a connection from the container to the Queue.

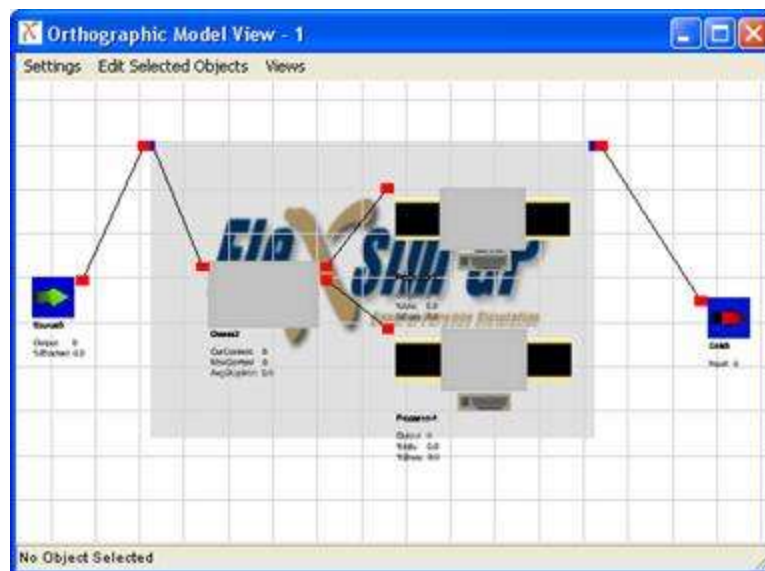


When you release the left mouse button you will see a connection from the internal port (blue) to the Queue.



### Step 7: Connect the Queue to the Processors

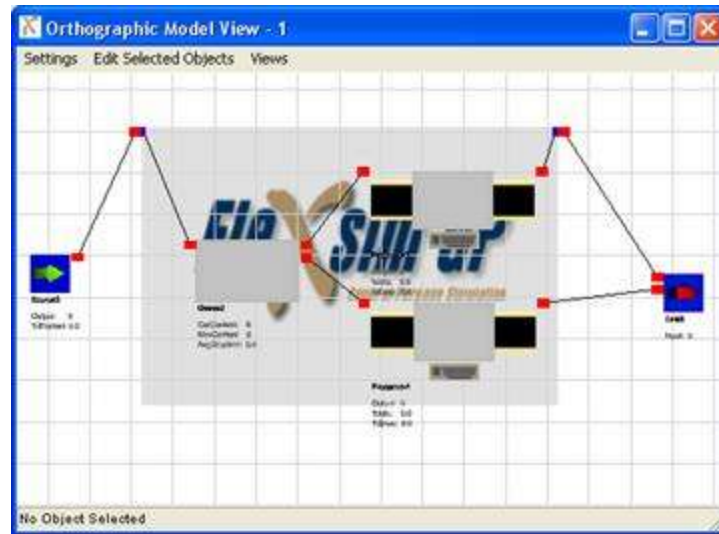
Following the same procedure connect the Queue to the 2 processors.



### Step 8: Connect the Processors to the container or directly to the Sink

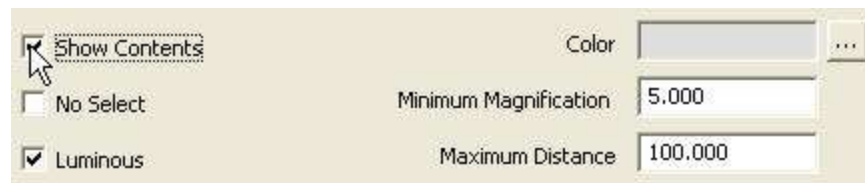
There are 2 way to connect &ldquo;into&rdquo; or &ldquo;out&rdquo; of a container. The first way was shown in step 5 when a connection was made from the Source to the container and then from the container to the queue. You can however connect directly from the Processor to the Sink by clicking and dragging a link. For this example the first Processor will be connected to the container which then connects to the Sink, and the second Processor will connect directly to the Sink.





### Step 9: Set the display option for the Container

At this point you now have a functioning container that holds a sub-model. How you display the container is now up to you. If you would like the contents of the container to be hidden during the simulation run you can toggle off the content display by unchecking the Show Contents box.



You can also have internal input and output ports of the visual tool be connected automatically when you create external port connections. To do this, go to the Visual Tool's parameters page, and use the drop-down boxes at the bottom of the window to select which object inside of the Visual Tool should be connected when an external connection is made to the Visual Tool. This allows you to hide the contents of the Visual Tool and make it into a completely encapsulated black box.



You can also use any of the Visual Display options to present your view of the container such as a box, building, or text. The contents of the container can be viewed at any time by right-clicking on the VisualTool in the ortho window and selecting "View Contents in Ortho".

Statistics on the container can be viewed the same as on other objects by view the statistics tabs in the properties dialog.

## Using the VisualTool as a Plane, Cube, Column, or Sphere

To use the VisualTool as a visual prop in your model is a simple process. You simply choose the type of prop you would like and define the parameters.

### Plane

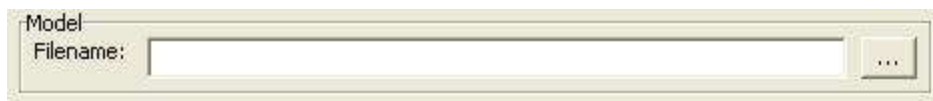
A Plane can be defined as a background such as an Autocad layout, a texture or picture, or a patch of color for a specific section of your model. The Plane is the default view for the visual tool. You simply set the size of the Plane and then choose the Texture. The Texture can be repeated in both the vertical or horizontal direction.

### Cube, Column, or Sphere

The cube, column, or spheres are simple shapes that can be assigned a texture much like the plane.

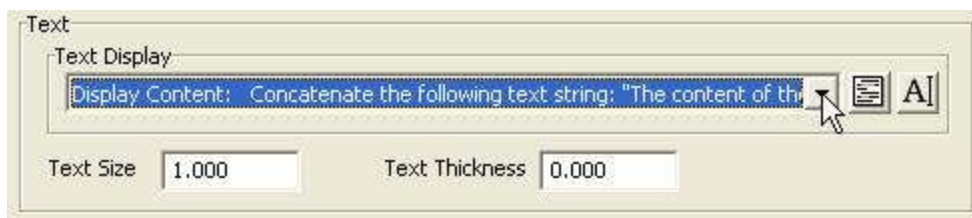
## Using the VisualTool as an Imported Shape


When using the VisualTool for an imported shape you need to have a 3D model or object that you wish to bring into the model. Flexsim supports a variety of 3D shape formats such as 3D Studio Max (.3ds,.max), VRML (.wrl) 1.0, 3D DXF (.dxf), and Stereo Lithography (.stl).



## Using the VisualTool as Visual Text

3D visual text can be added to your model to display labels, statistics, or other information in your model. When the visual display is set to visual text you are presented with a pick list of options for the visual text you want to display.



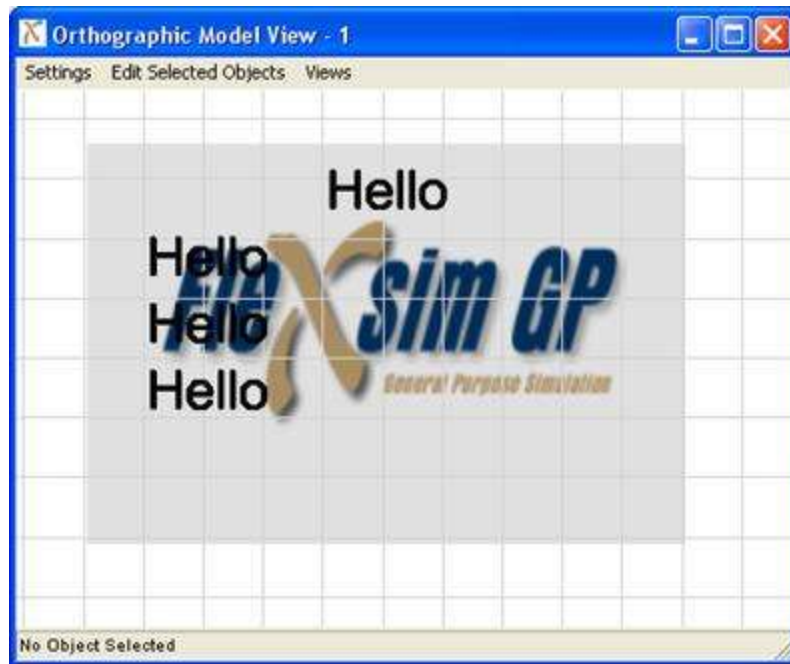
Pick options include simulation time, content, state, outputs, inputs, and many others. If any statistics are selected in the pick list the user must connect the VisualTool center port to the object that you want to display the information about. The text can be edited by selecting the code template button .

## Using the VisualTool as a Presentation Slide

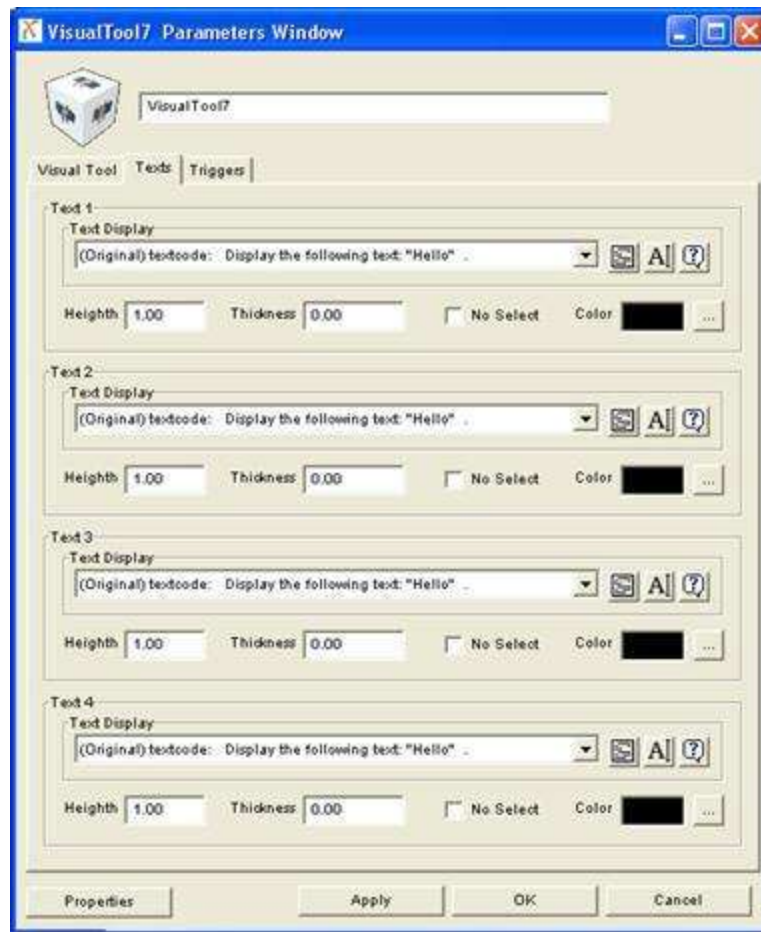
The VisualTool may also be used as a presentation slide much like you would use a slide to develop PowerPoint presentations. Presentation slides are placed in your model to present data, model findings, and presentation points. You can develop a "FlyTo" sequence to view the slides by using the Presentation Builder found in the "Presentation" menu. A tutorial on how to build a presentation is found in the tutorials section of this manual.



When the visual display is set to "Presentation Slide" you can drag additional VisualTool object "onto" the slide to create text for the slide. Each VisualTool that is placed on the presentation slide will be toggled to visual text and will be formatted on the slide. The first VisualTool added will be the slide header, the second will be item 1 and so on. For example if you were to drag 4 VisualTools onto the presentation slide you would see the following:

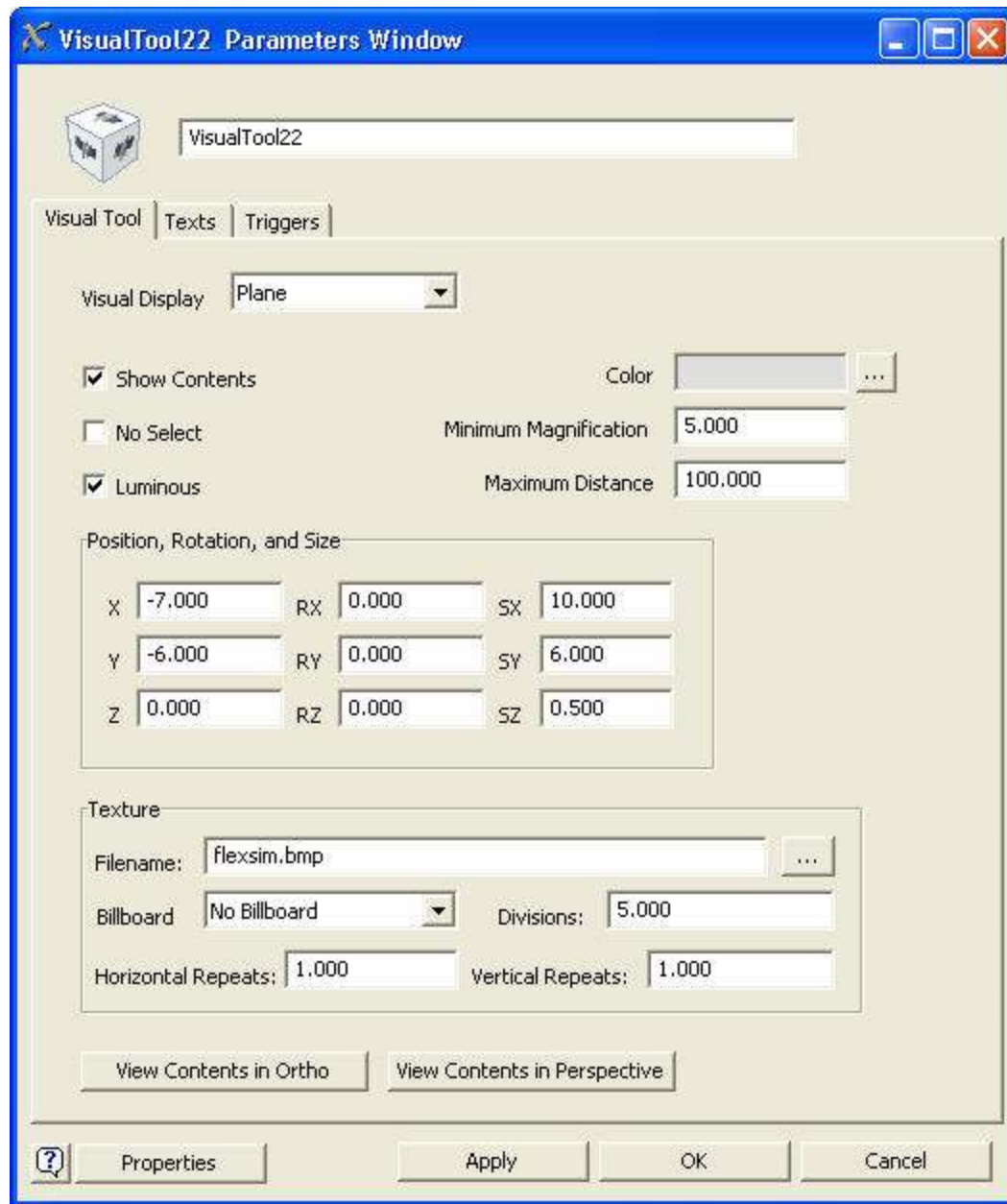


Each text is given the default location on the slide as shown. When you double click on the presentation slide to view the parameter screen you will see a new tab at the top left called "Text". By selecting this tab you will be able to edit the text to display your desired text.



You can apply any desired side background by selecting a texture on the &ldquo;Visual Tool&rdquo; tab, or remove the background by deleting the default Flexsim logo in the texture field.

## Other Visual Settings for the Visualtool



**Visual Display** - This selects the display type the VisualObject will be. The available types are: Plane, Cube, Column, Sphere, Imported Shape, Text, or Presentation Slide.

**Show Contents** - If checked the contents of the VisualTool will be displayed. Contents refer to text or objects that may be inside the object.

**No select** - If this box is checked, the object cannot be selected using the mouse in the Ortho or VR views.

**Luminous** - If this box is checked, the object will appear to give off its own light.

**Minimum Magnification** - This is the minimum magnification that the object will be visible for.

**Maximum Distance** - This is the maximum distance that the object will be visible for. If your view is further than this distance the object will not be displayed.

**Position, Rotation, and Size** - This area allows you to define the location, size, and rotation parameters of the object. It is the same interface as that on the Properties page.

**Texture** - These parameters are used to define how a texture is drawn on the object.

**Filename** - This file is the bitmap image that will be textured on the object.

**Billboard** - This option will display the texture as a billboard (flat surface). The billboard will always face the viewer.

**Divisions** - This number is used to define the number of sides on the object if it is a column and the "roundness" of the object if it is a sphere. If the object is a sphere, this number should be relatively high (~20).

**Horizontal repeats** - This number defines how many times the texture image will be repeated horizontally across the image.

**Vertical repeats** - This number defines how many times the texture image will be repeated vertically across the image.

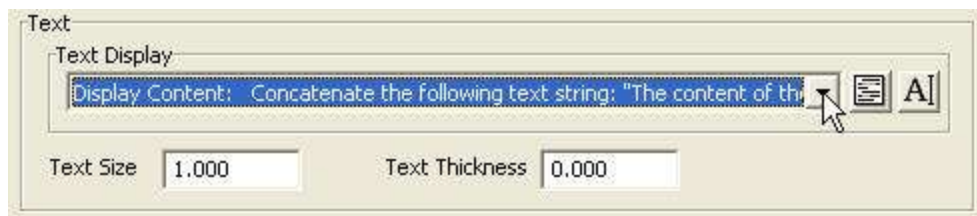
**Model** - These parameters only appear if the prop type is set to Imported Model (Imported shape must be selected to this to show).

**Filename** - This file is the .3ds or .wrl file that will be drawn for this object.

**View Contents in Ortho** - This button opens an orthographic view showing the contents of the VisualTool as a container.

**View Contents in Perspective** - This button opens an orthographic view showing the contents of the VisualTool as a container.

## Adjusting Visual Text



**Text Size** - This number defines the height of the text in the object. The width of the text will be automatically adjusted to keep the text easily legible.

**Text Thickness** - This number defines the thickness of the text in the object.

## Fluid Objects Library

### Overview

The Flexsim Fluid Library consists of eleven objects. These objects are designed to aid in simulating systems that move material as a fluid. The material does not necessarily need to be liquid, but can be nearly anything that is measured by weight or volume. In contrast, the standard Flexsim library is designed to move material that is in discrete amounts (boxes, pieces, etc). There are fluid objects that are designed to be used as an interface between the other fluid objects and the discrete objects. They allow the modeler to turn flowitems into fluid and fluid into flowitems.

Because of the discrete event nature of Flexsim, the fluid objects are not what might be considered “continuous objects.” Instead, they break time down into small sections called “ticks”. The length of a tick is called the tick time. At the end of each tick, all of the fluid objects in the model evaluate how much material they received and sent during the tick. Material only moves at the end of each tick. As the tick time gets longer, the model will run faster, but may lose accuracy. As the tick time gets lower, the model will slow down, but may be more accurate. It is the modeler’s responsibility to make sure that the tick time for the model is small enough that the simulation is still accurate.

The amount of material that moves between objects at the end of each tick is based on the input and output rates defined on each object, as well as the amount of material that is available to move and the amount of space available to move it to. The fluid objects have a standard way of moving material between themselves. The input and output rates are each defined using three values: the object’s maximum total rate, the maximum port rate, and a scaling factor for each port. The object’s maximum rate is the maximum amount that will be allowed in or out through all of the ports combined in a single time unit (not a single tick). Because fluid movement is evaluated one port at a time, this value may be used to stop certain ports from sending or receiving. The maximum port rate is the maximum amount of material that will be allowed in or out any one port in a single time unit. This value applies to all of the input or output ports on the object. Each individual port is then assigned a scale factor. This scale factor is multiplied by the port rate to calculate the maximum rate for that specific port. These scale factors are used to restrict flow through one or more ports without affecting the maximum flow through the others. These three values are applied separately to the input and output ports.

All of the fluid objects use this system to transfer material. However, because of the specialized nature of some of the objects, not all of these values may be available to the modeler for editing. Instead, some limited subset of the values will be presented and the object will maintain the other variables behind the scene. All of the values that the user can edit are presented in the GUI’s for each object, and will be discussed in more detail later.

Each object also keeps track of a value called the Product ID. This is a number that is used to identify the type of material that the object is currently holding. The product may contain sub-components. This is a list of all of the possible sub-components in the model. The list is recorded as a set of percentages of each sub-component. The Product ID number of an object does not represent any of the sub-components. It is

simply a number that the modeler assigns to a specific material. The Product ID of the objects, as well as the sub-components percentages will always be maintained by the fluid objects. The user only has to specify initial values on the objects that create fluid.

Most of the fluid objects have a system for displaying the current content of the objects as a percentage of the maximum content. This system is a colored bar called the level display or level indicator. The bar consists of two layers. One is dark gray and is generally drawn above the other. This represents the amount of empty space in the object. The other is the color of the object and represents the current content. If a bar is displayed as being fully gray, that object is empty. If a bar is fully colored, that object is full.

The level indicator bar can be moved, resized and rotated for each object in the model. The modeler can also state whether the bar is rectangular or cylindrical. The size of the bar is measured as a percentage (from 0-1) of the size of the object. The location is relative to the size of the object as well. The point (0,0,0) is one corner of the object's bounding box while (1,1,1) is the opposite corner. This flexibility allows the modeler to position the level indicator bar in such a way that it appears to be part of the object's 3d shape.

## Fluid Objects

- Ticker
- FluidGenerator
- FluidTerminator
- FluidTank
- FluidMixer
- FluidBlender
- FluidSplitter
- FluidPipe
- FluidProcessor
- FluidToItem
- ItemToFluid

## FluidBlender



### Overview

The FluidBlender is used to mix materials from multiple input ports based on percentages the user defines (not fixed amounts). It is most commonly used for in-line blending where the mixing is not done in batches.

### Details

The FluidBlender receives material from more than one input port based on a series of percentages that the modeler defines. Once the material has been pulled into the Blender it is ready to be pulled out immediately by downstream objects. The modeler defines the ProductID of the material that is released by the Blender. The sub-components of the mix will be a mixture of the sub-components of the material that was pulled in. The sub-component percentages are based on how much of each incoming product is mixed together.

The percentages that the Blender uses are defined in a table called the Blender Recipe. There is one row in the table for each input port that the Blender has. The rows in the table are not visible in the Parameters GUI until there are objects connected to the Blender's input ports. Each row has two columns: the ingredient name and the percentage. The name is a text string that the modeler uses to identify the material that is being pulled in. It is for the modeler's benefit only, it does not affect how the Blender will work. The percentage value is a number between 0 and 100, indicating what percentage of the incoming material will come from the port represented by the row.

The Blender always makes sure it is pulling the correct percentages. Each tick, the Blender will calculate how much from each port it needs to either fill itself or pull at its maximum input rate. If there is not enough of a material to meet this demand, the amount it pulls from the other ports will be reduced to keep the percentages correct.

Because the Blender controls how much it pulls from each port at any point in time, the modeler does not have access to the maximum port rate or the input port scale factors. They can, however, edit the object's maximum input rate. The user has complete control over the output rates and scale factors. They can change these values with the AdjustOutputRates function, which fires every tick. This allows them to update the output rates and scale factors as the model is running.

### States

- Empty - The Blender has no material in it.
- Mixing - The Blender has received material that it mixed together.
- Blocked - The Blender has received material, but it cannot send it downstream.

### Parameters tab-pages



FluidBlender  
FluidBlenderRecipe  
FluidLevelDisplay  
FluidBlenderTriggers

## FluidGenerator



### Overview

The FluidGenerator provides an infinite supply of fluid material for a model. The Generator can be set to refill at a fixed rate (that can be faster or slower than the outgoing rate) or it can refill itself a set amount of time after it becomes empty.

### Details

The FluidGenerator is used to create fluid material for a model. The modeler defines the capacity of the Generator, as well as the amount of material in it when the model is reset. They can also define the ProductID and sub-component mix for the initial product.

The Generator creates material in two ways. The first is at a constant rate that the modeler defines. This rate can be faster or slower than the output rate of the object. If it is faster, then the Generator will always be full. If it is slower, then the Generator will eventually become empty. Even if the Generator becomes empty, it will make more material during the next tick, but the downstream objects may not be able to receive it at the full rate.

The second way to create material is to fill the Generator to its maximum amount instantly, but only after a certain amount of time has elapsed since the Generator became empty. The modeler can define how long this wait time is. This is used to simulate situations where material is available on a regular basis, but is not available all of the time. For example, trucks full of raw materials that arrive once a day could be simulated with this technique.

The modeler is given control over all of the rate variables that affect the Generator's output rate, as well as a function called "AdjustOutputRates". This function fires every tick and allows the modeler to change the output rates during a model run. The modeler is not given control over the variables controlling the input rate, as no material ever comes into a Generator.

### States

**Empty** - The Generator has no material in it.

**Not Empty** - The Generator has some material in it.

**Full** - The Generator's maximum capacity has been reached.

### Parameters tab-pages

FluidGenerator

FluidLevelDisplay

FluidGeneratorTriggers

## FluidMixer



### Overview

The FluidMixer is used to combine products together into a single, new product. The different materials can either be pulled sequentially or in parallel. The Mixer always works in batches. It does not send any material until it has received and processed all the material that it was set to receive.

### Details

The FluidMixer pulls material from one or more input ports and mixes it together. The modeler defines the ProductID of the material that is released by the Mixer. The sub-components of the mix will be a mixture of the sub-components of the material that was pulled in. The sub-component percentages are based on how much of each incoming product is mixed together.

The modeler defines a series of steps that the Mixer will go through. These steps are defined in the Step Table. Each step can pull material from zero or more input ports at the same time. A delay time can also be defined that begins after all the material for the step has been collected. The next step will not begin until after the delay time is over. In addition, the modeler is given a trigger that fires before the delay at a step a trigger that fires after the delay (but before the next step starts). These triggers can be used for things like calling an operator to perform work during the delay. The modeler can assign a text description to each of the steps in the table. This description is displayed near the object's name in the model view window. It does not affect the behavior of the object.

The modeler defines how much material comes in by using the Mixer's Recipe Table. Each row of the table represents material coming from a single input port during a single step. Each row has four columns: the ingredient name, the port number, the step number and the amount. The name is a string that describes the material being pulled in by that row. It is for the modeler's benefit only. The Mixer ignores the value. The port number is the input port that the material will be pulled from. The step number is the step in the Step Table the Mixer must be in to pull this material. Once the Mixer has pulled the correct amount for a given row, it will not pull any more material for that row, even if the other rows in the same step are not complete yet. The amount is the actual amount of material that will be pulled from the specified port during the specified step.

Different materials will be pulled in parallel if they have the same step number. They will be pulled in series if they have different step numbers. It is possible (and often very useful) to have a recipe that calls for some ingredients to be pulled in parallel and others to be pulled in series. There is no limit on the number of steps or ingredients that can be defined. There is also no limit on the number of ingredients that can be pulled during any single step. If the modeler wishes to have material

pulled from the same input port during multiple steps, they have to define multiple rows in the Recipe Table.

Because the Mixer controls which ports it will pull from at any point in time, the modeler does not have access to the input port scale factors. They can, however, edit the object's maximum input rate and maximum port rate. It is very important that they make sure that the maximum object rate is high enough to allow input from multiple ports if their recipe requires that. Once the delay after the final step is complete, the user has control over the output rates and scale factors. They can change these values with the AdjustOutputRates function. This function is not called until the Mixer has finished collecting everything. Once the Mixer is finished collecting and processing a batch, the function is called during every tick. It is not called while the Mixer is still working through the Step Table.

The Mixer provides a visual display of the material that has been received at any point in the process. The level indicator bar will not show the Mixer's color until the delay time of the last step is complete. Before that time, the level indicator is a series of layers of different colors. There is one layer for each ingredient in the Recipe. The colors of the layers are the colors of the upstream objects that the Mixer is receiving material from. The size of each layer is the percentage of the total batch that has been pulled for that ingredient. This multi-color bar is a good indicator of what is happening in a Mixer at any given time.

## States

**Empty** - The Mixer has nothing in it and is waiting to start step 1.

**Filling** - The Mixer is receiving material for its current step.

**Starved** - The Mixer has not completed its Step Table, but there is no material coming into it.

**Releasing** - The Mixer has completed the Step Table and is sending the finished product downstream.

**Blocked** - The Mixer has completed the Step Table, but is unable to send material downstream.

## Parameters tab-pages

FluidMixer

FluidMixerSteps

FluidLevelDisplay

FluidMixerTriggers

## FluidPipe



### Overview

The Pipe is used to simulate the time required to move material from one object to another. It can appear as either a cylindrical pipe, or as a simple conveyor.

### Details

The Pipe carries material from one point in the model to another. It is most often needed when the modeler has to take into account the time required to move material from one point to another. It is also used if the modeler needs to send material from multiple objects to a single input port on another object or when material from one output port needs to be split.

The modeler specifies a maximum content and maximum flow rate for the Pipe. The amount of time that it takes material to travel through the Pipe is based on these two values. The maximum flow rate is used as the maximum input and output rate. The actual output rate is based on the rate that material came into the Pipe. The material will go out of the Pipe at the same rate it came in, unless the Pipe “backs up”. If one of the output ports does not receive all of the material that the Pipe tries to send, the material in the Pipe “backs up” and more is available to be sent during the next tick.

The modeler also selects an output flow mode. There are three modes available. The first flow mode is called “Flow Evenly.” In this mode, the Pipe attempts to divide the output rate evenly between the output ports. The second flow mode is called “First Available.” In this mode, the Pipe tries to send all of the material that is ready to be sent to the first output port. If that object can not receive it all, the Pipe tries to send it to the next port, and so on. The third mode is called “User-Defined.” This mode allows the modeler to edit the maximum port input and output rates, as well as the port scale factors. The modeler also has access to the `AdjustInputRates` and `AdjustOutputRates` functions. Unlike the other fluid objects, the modeler can read but can not change the object input and output rates using these two fields.

The Pipe has no level indicator bar, but does have some visual indications of its state. When the Pipe is empty, it is shown as a solid gray color. When material is moving, the Pipe is shown in the color assigned to it, but that color is fading in and out. When the Pipe is blocked and unable to send, it is drawn in its assigned color and does not change.

The modeler can change the look of the Pipe by editing the Pipe’s Layout Table. Each row of the table represents a single, straight section of the Pipe. The modeler can define how long the section is and what its diameter is. They can also specify the angles around the Z and Y axis that the pipe will rotate for the next section. There is also a column that allows the modeler to state whether or not the joint between this section and the next one will be drawn. By editing this table, the modeler can make the Pipe look however they need. The modeler also has the option of displaying the

Pipe as a simple conveyor. When the Pipe is drawn as a conveyor, the Layout Table is still followed, only the option to display the joints between sections is ignored.

### States

**Empty** - The Pipe has no material in it.

**Filling** - The Pipe received material, but no material has been sent out recently.

**Starved** - The Pipe has material, but has not sent or received any recently.

**Flowing** - The Pipe has material that is it currently sending downstream.

**Blocked** - The Pipe has material that it is unable to send downstream. The material in the Pipe is "backing up".

### Parameters tab-pages

FluidPipe

FluidPipeLayout

FluidPipeTriggers

## FluidProcessor



### Overview

The FluidProcessor is used to simulate a processing step that continuously receives and sends fluid material (such as a continuous cooker).

### Details

The FluidProcessor receives and sends material based on an over-all rate that the modeler specifies. The modeler specifies the maximum output rate of the Processor. This value is used to determine the input rate as well. The actual output rate is based on the rate that material came in. Material will leave at the same rate that it entered, unless the output is, for some reason, lowered (such as the downstream object closing its input ports or breaking down). If this happens, the material will “back up” in the Processor and more will be available to send when the downstream object can receive more again. Once the downstream object has received all of the “backed up” material, the output rate will go back to the input rate. The amount of time that material spends in the Processor is based on the Processor’s maximum output rate and its maximum capacity.

The modeler is also able to specify a loss value. This value is a number from 0-1 that represents the percentage of incoming material that is lost due to machine inefficiency, evaporation, or any other reason. Whenever any material enters the Processor, it is immediately reduced by this percentage.

Unlike other objects, the Processor can only receive from one port and send to one port during each tick. The Processor has functions called “Receive Port Number” and “Destination Port Number” that the modeler uses to decide which ports will be used. If these functions returns 0, the first input or output port that has material or has available space will be used. If another value is returned it is the number of the input or output port that will be used. This is the more common use of these two fields.

### States

**Empty** - The Processor has no material in it.

**Processing** - The Processor has material in it that it is trying to send downstream, or that has not been in the Processor long enough to be sent.

**Blocked** - The Processor has material in it that it is unable to send downstream.

### Parameters tab-pages

FluidProcessor  
 FluidLevelDisplay  
 FluidProcessorTriggers

## FluidSplitter



### Description

The Splitter is used to send material to multiple output ports in percentages that the modeler specifies. These percentages are specified in a table called the Splitter Percents Table. Each row in the table corresponds to an output port. There are columns that allow the modeler to enter a description of each port (for the modeler's use only) and the percentage (from 0-100) of the outgoing material that will go to each port. When the Splitter sends material out, it will always send in the percentages specified. If the Splitter can not send all the material that it is trying to send, it will reduce the amount that it sends to the other ports to keep the percentages equal at all times. The modeler has complete control over the input rates and scale factors using the AdjustInputPorts function. They also have control over the total output rate. The port output rate and the scale factors will be adjusted by the object itself as needed. It should be noted that the FluidSplitter may not adjust the amount of material to go out correctly if the downstream objects are full, or nearly full. Due to the timing of the calculations, the Splitter may decide there is no room for material it is trying to send even if the downstream object will release enough material to make room for the Splitter's material.

### Overview

The FluidBlender is used to send materials to multiple output ports based on percentages that the modeler specifies.

### Details

The FluidBlender receives material in a normal manner, but sends it out differently. It sends material out in percentages that the user specifies in a table called the Splitter Percents. The table is not visible in the Parameters GUI until the Splitter is connected to downstream objects. Each row of the table corresponds to a single output port, and has two columns: a description and the percentage. The description is a text string that describes the material being sent to that port, it is not used by the object and is for the modeler's benefit only. The percentage is a number between 0 and 100 that indicates what percentage of the outgoing material should be sent to that port.

Every tick, the Splitter calculates the amount of material that should be sent to each of the downstream objects, based on rates and capacities. If there is not enough room in one of the downstream objects to receive the amount the Splitter has calculated, the amount sent to each port is reduced to keep the percentages correct.

Because the Splitter controls how much it sends to each port at any point in time, the modeler does not have access to the maximum port rate or the output port scale factors. They can, however, edit the object's maximum output rate. The user has complete control over the input rates and scale factors. They can change these



values with the AdjustInputRates function, which fires every tick. This allows them to update the input rates and scale factors as the model is running.

### States

**Empty** - The Splitter has nothing in it.

**Not Empty** - The Splitter has material in it that can be sent out.

### Parameters tab-pages

FluidSplitter

FluidSplitterPercents

FluidLevelDisplay

FluidSplitterTriggers

## FluidTank



### Overview

The FluidTank is a simple Fluid Object that can receive and send material at the same time. The modeler decides the maximum capacity of the Tank and up to three points (called "marks") that will cause triggers to fire when the content in the Tank reaches them.

### Details

The FluidTank is the most generic of the Fluid Objects. It can receive and send material at the same time. The modeler has complete access to the variables that control the input and output rates. They are also given two functions that fire at the end of every tick. These functions are called "AdjustOutputRates" and "AdjustInputRates". They are used to changed the values of the input or output rates during a model run.

The Tank can start a model with no content, or with a set amount. If the tank begins with content, that is all it will create during a run. It may continue to receive material from upstream, however. Starting with a fixed content value is very useful for models that have a fixed amount of material that will enter. If the modeler wants a constant or infinite stream of incoming material, they should use a FluidGenerator instead.

The Tank has a maximum capacity that the modeler defines. The content of the Tank will never go above this value. If, at the end of a tick, the Ticker calculates that the Tank should receive more than it can currently hold, only the material required to fill the Tank will be transferred.

The modeler can define three points that will cause triggers to fire when the content of the Tank reaches them. These points are called "Marks". Whenever the content of a Tank passes one of these marks (either rising or falling), a trigger is fired. The user can use that trigger to open or close ports, send messages, change rates, or many other things. If two or more marks are set for the same value, the trigger for only one of them will fire. For example, if the low mark and the mid mark are both set to 10, when the content in the Tank changes from 9 to 10 only the trigger for the low mark will fire.

### States

**Empty** - The Tank has no material in it.

**Not Empty** - The Tank has some material in it.

**Full** - The Tank's maximum capacity has been reached, it will not receive any material unless it can send some out.

### Parameters tab-pages

FluidTank  
FluidTankMarks  
FluidLevelDisplay  
FluidTankTriggers

## FluidTerminator



### Overview

The FluidTerminator is used to destroy fluid material once the model is done with it.

### Details

The FluidTerminator is the object that modelers use when they want to remove fluid material from the model without turning it into flowitems. The Terminator keeps track of how much of each different type of material it receives, with a few restrictions. It can only keep track of up to 14 different Product ID's. These should be integers that are 1 or greater. The data it collects for each product id is displayed in the Class Statistics section of the Properties GUI's statistics tab.

The user has complete control of the input rates of the Terminator. This includes a function called "AdjustInputRates" that fires every tick. This function is used to change the input rates and scale factors during a model run. The modeler is not given control over the output rate because the material the Terminator receives is destroyed and cannot be sent downstream.

### States

**Collecting** - The Terminator is never full and is always able to receive material, so it is always in a collecting state.

### Parameters tab-pages

FluidTerminator  
FluidTerminatorTriggers

## FluidToltem



### Overview

The FluidToltem is an object that is designed to interface between the fluid objects and the discrete objects. It receives fluid and converts it to flowitems that it sends downstream.

### Details

The FluidToltem is used to convert fluid to flowitems that can be sent to any Fixed Resource. The modeler chooses the flowitem class, the default itemtype and name for the flowitems that are created. The modeler also has to specify the amount of fluid material that must be collected before a flowitem can be created. This is done by editing two values. The first is the amount of fluid in each discrete unit. The second is the number of discrete units that each flowitem represents. The total amount of fluid that will be collected before a flowitem is created is found by multiplying these two values together. For example, if a single flowitem represents 20 cans and each can holds 5 gallons of fluid, the number of discrete units per flowitem is 20 and the number of fluid units per discrete unit is 5. Therefore, for each 100 gallons of fluid collected in the FluidToltem a single flowitem will be sent out.

Once a flowitem is created, the standard Send To Port logic is used to send it downstream. This means that the FluidToltem can send flowitems to any of the discrete objects and can call operators to do the transporting. The modeler can define how the input to the FluidToltem works by changing the maximum object input rate, maximum port input rate and the port scale factors using the GUI and the AdjustInputRates function.

### States

<NOT TOO SURE RIGHT NOW - FINISH ME>

### Parameters tab-pages

FluidToltem  
 FluidToltemFlow  
 FluidLevelDisplay  
 FluidToltemTriggers

## ItemToFluid



### Overview

The ItemToFluid is an object that is used to interface between the fluid objects and the discrete objects. It receives flowitems and converts them to fluid material.

### Details

The ItemToFluid is a Fixed Resource object that is designed to interface between the discrete objects and the fluid objects. When it receives a flowitem, it destroys the flowitem and creates fluid that can be sent to any of the other fluid objects. The amount of fluid created is based on two values that the modeler specifies. The first is the amount of fluid that each discrete unit creates. The second is the number of discrete units each flowitem represents. Generally this value will be 1, but often the modeler will use a flowitem to represent multiple physical objects. In that case, the number should be set to the number of objects represented by each flowitem. The total number of fluid units that are created for each flowitem that enters is found by multiplying these two values together. For example, a single flowitem may represent 10 bags that each contain 25 pounds of a fluid material. The discrete units per flowitem in this case is 10, and the fluid units per discrete unit is 25. Each time one of these flowitems enters the ItemToFluid, 250 pounds of fluid are created.

The ItemToFluid has a maximum capacity that the modeler defines. The object will not accept any flowitems if there is not at least enough empty space in it to hold all of the material that flowitem will create. The modeler can also define the ProductID, and sub-component mix of the fluid that is created and sent out.

The flowitems entering can be controlled using standard FixedResource pull logic. The output of fluid is completely controlled by the modeler. They can edit the maximum object rate, maximum port rate and the port scale factors using the GUI and the AdjustOutputRates function.

### States

**Empty** - The ItemToFluid has no material in it.

**Not Empty** - The ItemToFluid has fluid material in it that has not been pulled out yet.

**Full** - The ItemToFluid's maximum capacity has been reached

### Parameters tab-pages

ItemToFluid  
ItemToFluidFlow  
FluidLevelDisplay  
ItemToFluidTriggers

# Modeling Tools

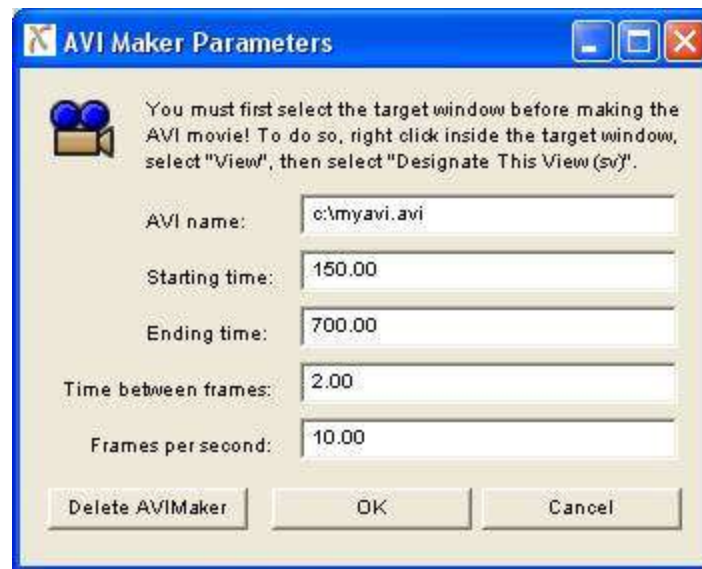
## Modeling Tools

This section contains information about different modeling tools that you will use in building, configuring, and obtaining results from your model. Many of these tools can be accessed through the Tools menu, while others are accessed from other areas. These tools are listed below.

- AVI Maker
- Excel Interface
- Flowitem Bin
- Global C++ Code
- Global Object Pointers
- Global Tables
- Global Time Tables
- Global User Events
- Import Media Files
- Model Startup Code
- MTBF/MTTR
- Multiple Table Excel Import
- Optimizer
- Presentation Builder
- Script Editor
- Simulation Experiment Control
- Single Table Export
- Single Table Import
- Sky Box Editor
- Table Configurator
- User Commands
- Visio Import
- Watch List

## AVI Maker

The AVIMaker is created when the “AVI Maker” option is selected in the Presentation menu. It is a special object in the model that calls the commands to make an AVI file of the model running. It will make this file as long as it is in the model. If the user does not want to make the AVI file, they need to delete the object from the model. Before running the model, the user must designate a view to record. This is done by right-clicking on the view they wish to record, and selecting “View>Designate This View(sv).” The model may run very slowly while the AVI file is being recorded. It will not respond to the speed slider bar in the run control window during that time.



**AVI Name** - This is the name of the file that the AVIMaker will write to. It should have an .avi extension.

**Starting Time** - This is the time that the AVIMaker should start recording the AVI file.

**Ending Time** - This is the time that the AVIMaker will stop recording the AVI file. It is recommended that the model not be stopped before this time, as it may corrupt the file being written to.

**Time between frames** - This is how much simulation time passes in the model between recorded frames.

**Frames per second** - This number defines how many frames per second the AVI file plays back at.

**Delete AVIMaker** - When this button is pressed, the AVIMaker is deleted from the model. The model will run at normal speed again, and the AVI file will not be generated.

### How to Get the AVI Maker Working Properly



The AVI maker can often be tricky to work with. Here are steps to go through to make sure that your AVI is created with as little hassle as possible.

1. If there is already an AVI Maker in the model, delete it by pressing the Delete AVIMaker button.
2. Compile the model.
3. Re-open the AVI Maker window.
4. Fill in the above mentioned fields appropriately.
  1. Make sure the name of the avi file is not the name of a file that already exists.
  2. Set the start and stop times according to the simulation time that you want the avi to be recorded during.
  3. Set the frames per second value to whatever you want your avi playback speed to be. 10 frames per second is usually reasonable.
  4. Set the time between frames value based on the frames per second value you specified. Find an ideal run speed that you want the avi to record for the model (from the simulation run control panel). The time between frames should be calculated as that ideal run speed divided by the frames per second
5. Right click in the ortho or perspective view you want to be recorded and select the "Set as Selected View sv()" option
6. Resize the ortho or perspective view to the resolution you want for the avi movie. Setting a smaller window size can dramatically increase the speed that the avi maker can create the movie.
7. Reset the model
8. A window will pop-up about the codec to use. Enter the codec/compression that you want to use.
9. Run the model
10. **IMPORTANT: Wait until the avi maker is finished.** Once the model gets to the avi's starting time, do not press any buttons or click on anything until the model time is past the ending time you have specified for the avi maker.
11. Once the model has run past the avi maker's ending time, stop the model. **Do not hit reset again until you have deleted the AVI Maker.**
12. Delete the avi maker using the Delete AVIMaker button.

## Excel Interface



**Single Table Import** - Press the Single Table Import button to import the configured table into Flexsim. To configure this table, press the edit button to bring up the Single Table Import editor.

**Single Table Export** - Press the Single Table Export button to export the configured table from Flexsim to Microsoft Excel. To configure this table, press the edit button to bring up the Single Table Export editor.

**Multiple Table Import** - Press the Multiple Table Import button to import several tables into Flexsim. To configure these tables, press the edit button to bring up the Multiple Table Excel Import editor.

**Custom Import** - Click on the Custom Import button to import from excel using your own custom code. To write and edit this custom code, press the edit button to bring up a code editor. Once you have edited this code, you will need to compile before being able to import from excel.

**Custom Export** - Click on the Custom Export button to export to excel using your own custom code. To write and edit this custom code, press the edit button to bring up a code editor. Once you have edited this code, you will need to compile before being able to export to excel.

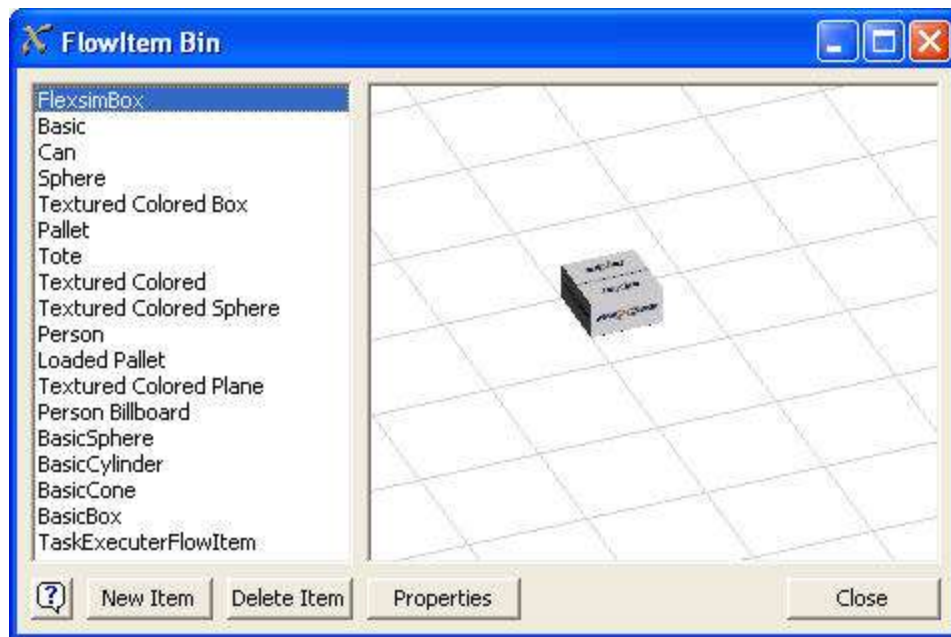
**Global Settings** - Click on the Global Settings button to configure import/export parameters, including the install location of Excel, etc.

## Flowitem Bin

You can get to the flowitem bin through the main tools menu, or through Flexsim's toolbar.

### Flowitems

Flowitems are the simple objects that are created to move through the model. They can represent actual objects, or they can be representative of a more abstract concept. Different classes of flowitems are created in this window and are stored in the Flowitem Bin. The editor is opened by pressing the FlowItem button on the toolbar, or from the Tools menu.



**Flowitem List** - This list contains all of the available flowitem types. When one is selected, it is shown in the main window. You can edit the properties of the flowitem such as name, shape, and size by selecting the flowitem in this list, then pressing the Properties button.

**New Item** - This button adds a new flowitem to the bin. The new flowitem is a copy of the currently selected item in the flowitem list.

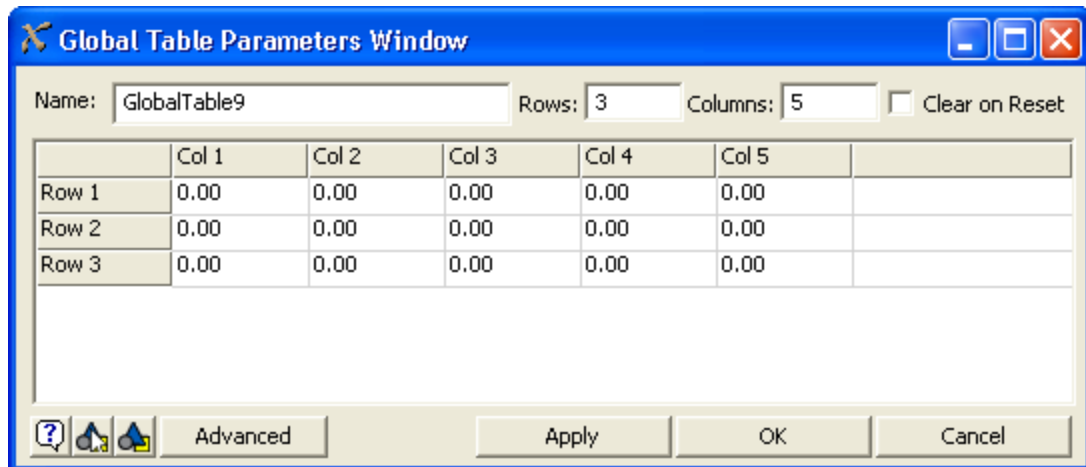
**Delete Item** - This button deletes the currently selected flowitem from the bin. It can no longer be created in the model.

**Properties** - This button opens the currently selected flowitem's Properties window.

## Global Tables

These objects are not dragged out into the model. They are created through special dialog boxes that are reached through the toolbars.

Global tables are accessed through the Tools menu. They can store numerical or string data. This data can be accessed by any object in the model using the `gettablenum()`, `gettablestr()`, `settablenum()`, `settablestr()`, `reftable()` commands. For more information on these commands, refer to the command summary. A model may have any number of Global Tables.





**Name** - This is the table's name. It should be memorable and describe the table's function. The commands to read and write to them access them by name.

**Rows** - This is the number of rows in the table. After changing it, press "Apply" to update the table on-screen. Any new rows that were created can now be edited.

**Columns** - This is the number of columns in the table. After changing it, press "Apply" to update the table on-screen. Any new columns that were created can now be edited.

**Clear on Reset** - If this box is checked, all number cells in the table will be set to 0 when the model is reset.

**Advanced** - This option brings up a table configurator window, allowing you to customize the tables for specific uses.

**Add to User Library** - The  and  buttons let you add this table to the currently active user library. The buttons add it as a draggable icon or as a component for automatic install, respectively. For more information, refer to the user library documentation. Note that this option is not available if you are editing a label table.

### Editing the Table

To edit a cell in the table, click on the cell and type the data in the cell. Press the arrow keys to navigate between cells. Cells hold numbers by default, but can be set to hold string data by right-clicking on the cell and selecting "Insert>Add String Data."

## Global Time Tables

Time tables are accessed in the Tools menu. They are used to schedule state changes, such as scheduled down-time, for specific objects in the model. Each time table may control many objects, and each object may be controlled by many time tables. A model may contain any number of Time Tables.

**Time Table Parameters Window**

Time Table Editor

Name: TimeTable11

Model: TaskExecuterFlowItem

Time Table Members

Rows: 1.00

Repeat Time: 0.00

Graphical Editor

Add Table to MTEI

	Time	State	Duration
Row1	0.00	12.00	0.00

Down Function: Execute stopobject() The stopobject() command is class depe



Resume Function: Execute resumeobject() The resumeobject() command is clas

OnDown: Do nothing

OnResume: Do nothing

Apply OK Cancel

**Name** - The name of the Time Table. This should be descriptive of the function this time table plays in the model. For example: "Weekend" or "Shift Change."

**Adding and Removing TimeTable Members** - At the top of the window, the panel on the left shows objects in the model. The panel on the right shows the list of the TimeTable's members. Select an object in the left panel and click on the  button to add the object to the member list. Select an object from the member list on the right and click the  button to remove that member from the list.

**Rows** - This is the number of rows in the table. After changing it, press "Apply" to update the on-screen table. Each row records the time to change state, what state to change to, and how long to remain in that state.

**Repeat Time** - This number specifies the time from when the first state change begins to when the table is to be repeated. If the first row has a time of 60 and a repeat time of 300, then the first down will occur at time 60, and the table will repeat itself at time 360, 660, 960, etc..

**Time Table** - The table can be viewed and edited here.

**Time** - This is the time since the table began that the state change should occur.

**State** - This is the state that the objects controlled by this table will change into when the time table tells it to go down. If you click on this column, a drop-down box will appear at the top, giving you a list of possible states. Refer to the library objects for more information about what each state means to each object. Refer to the state list for a quick reference of each state's number and macro definition.

**Graphical Editor** - This will open a window where you can specify the time table's downs using a weekly or daily table.

**Add Table to MTI** - This button will add the table to the model's multiple table import.

**Duration** - This is how long the objects will stay in the new state before changing back to their original state.

**Down Function** - This pick list is executed when the objects in the member list go down. It is executed once for each object in the member list. This is where you specify what to do to stop the object.

**Resume Function** - The pick list is executed when the objects in the member list resume their operation. It is executed once for each object in the member list. This is where you specify what to do to resume the object.

**OnDown** - This pick list is fired at the same time as the Down Function, but it is only executed once, instead of once for each object in the member list. See Down/Resume Trigger.

**OnResume** - This pick list is fired at the same time as the Resume Function, but it is only executed once, instead of once for each object in the member list. See Down/Resume Trigger.

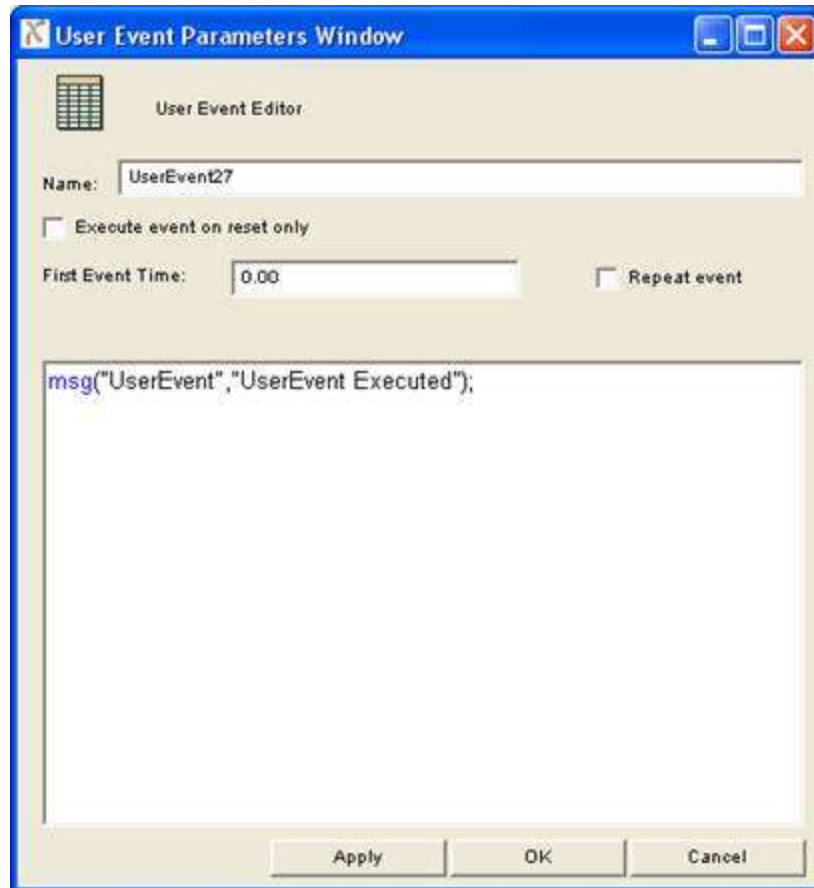
**Add to User Library** - The  and  buttons at the bottom let you add this object to the currently active user library. The buttons add it as a draggable icon or as a component for automatic install, respectively. For more information, refer to the user library documentation. Note that this option is not available if you are editing a label table.

**Note on using several down schedules for the same object:** If one object has several down schedules, each with its own down state, you may come across problems with the state diagrams of your objects. This is caused because of the nature of the `stopobject()` and `resumeobject()` commands. If two entities request the same object to stop, the object does not remember the state for each stop request. For more information, refer to the `stopobject()` command in the command summary.



## Global User Events

User Events are accessed in the Tools menu. They are C++ functions that execute at set times during the model's run, but are not connected with any specific, visible object. They are created in a special node in the model called "Tools," in a sub-node called "UserEvents." A model can have any number of user events.



**Name** - This is the name of the user event. It should be descriptive of what the user event does.

**Execute event on reset only** - If this box is checked the event will only be executed when the reset button is pressed.

**First Event Time** - This is the time that the user event will occur.

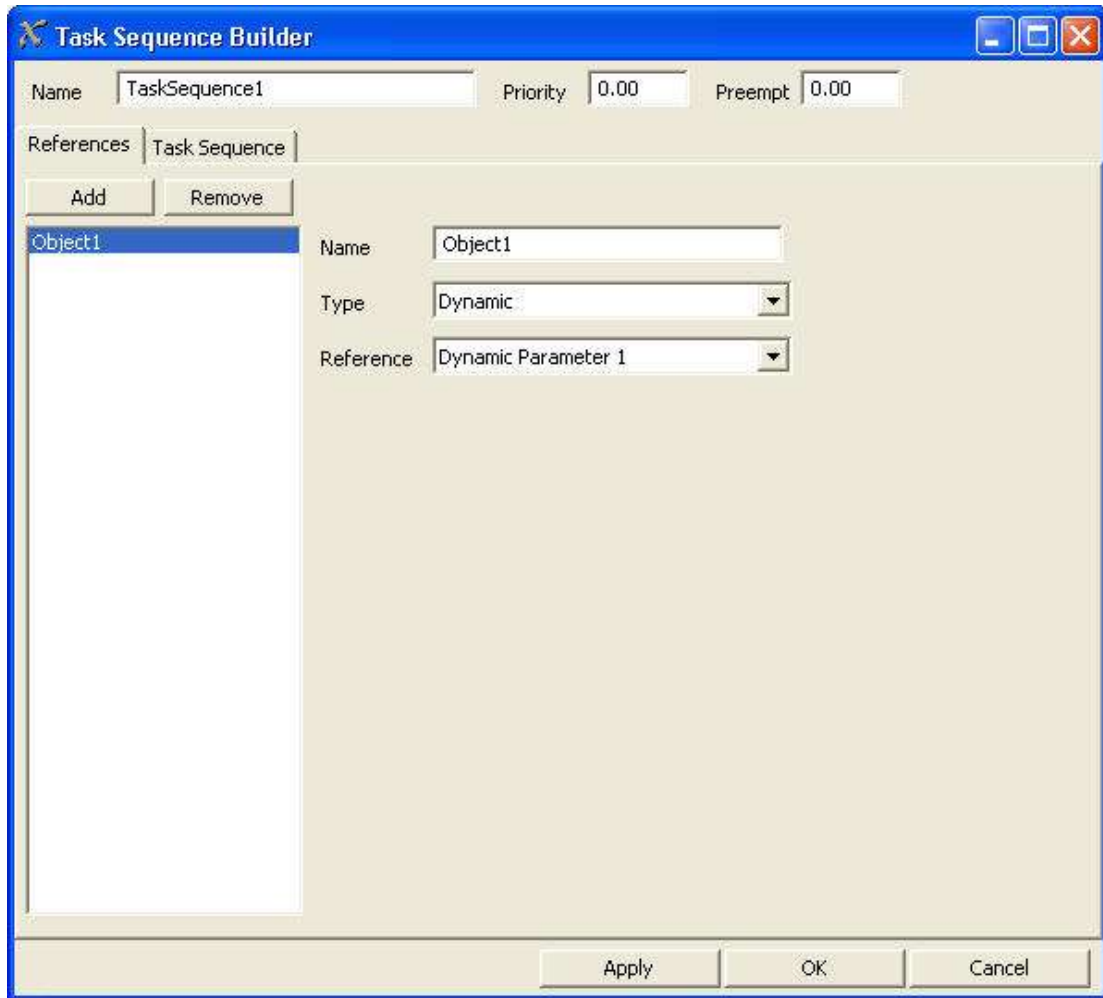
**Repeat Event** - If this box is checked, as soon as the user event finishes, it begins counting towards the execution time again. User events will always repeat in regular intervals defined by the execution time.

**Event Code** - This is where the C++ code for the event is written. Any valid C++ statements can be used in this field. If you have edited this code, the model must be compiled before running the simulation.



## Global Task Sequences

Global Task Sequences let you build Task Sequences through a graphical user interface instead of through code. To create a Global Task Sequence, select the main menu option Tools > Global Task Sequences > Add. The following window will appear.

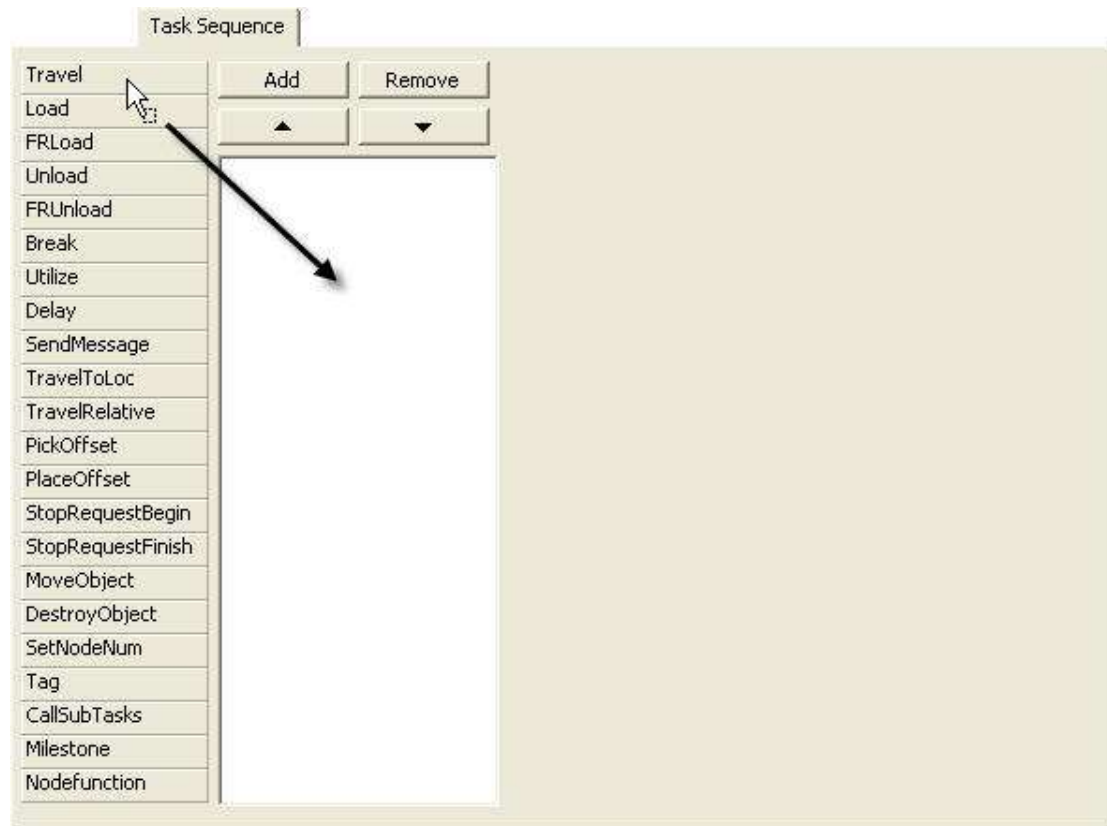


First, give the Task Sequence the appropriate name.

Global Task Sequences are built by first creating object reference aliases, and then by creating tasks associated with those references. The object references can be either dynamic, meaning they are resolved at the time that the instance of the task sequence is actually created by being passed in as dynamic parameter to the `createglobaltasksequence()` command, or they can be static, meaning they stay the same across all create instances of the task sequence. Press the Add and Remove buttons to add or remove object references. For each object reference, you can give it a name and type (either Dynamic or Static). For dynamic types, you can choose the reference to be one of Dynamic Parameters 1-5. These parameters will be passed in to the `createglobaltasksequence()` function when the instance is created.

For static references, click the browse button and select the desired object in the model tree.

Once you've created the references you need, go to the Task Sequence tab.

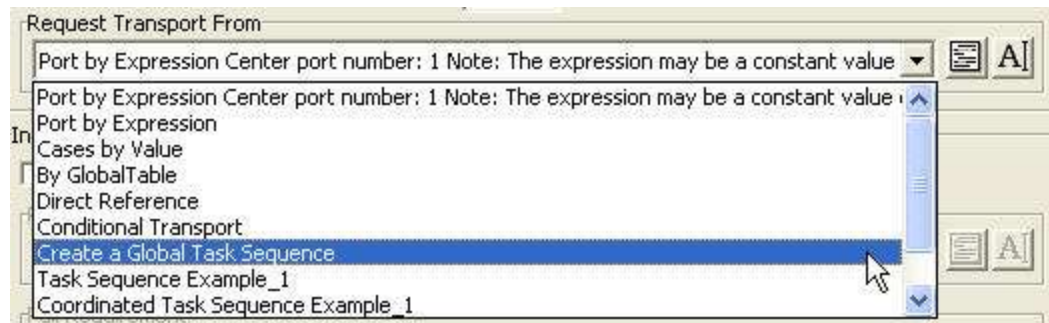


By default the task sequence is empty. On the left is a grid of droppable tasks. Add tasks by dragging them from the grid on the left and dropping them into the list. An attributes panel will appear to the right of the list.

Description	Travel
Task Type	Travel
Destination	Object1
Not Used	No Object
End Speed	0.00
Force Travel	0.00
Not Used	0.00
Not Used	0.00

To edit each task's attributes, just click on the task in the list, and edit the attributes on the right. Use the buttons above the list to relocate or remove the tasks in the sequence.

Once you've created your task sequence, you can select the "Create Global Task Sequence" option in the "Request Transport From" field of the Flow tab in any object's Parameters window.



Just enter the Global Task Sequence's name and the appropriate dynamic parameters.

Create a Global Task Sequence

Name: **"CustomTS"**

Dispatch To: **centerobject(current,1)**

Dynamic Parameter 1: **item**

Dynamic Parameter 2: **current**

Dynamic Parameter 3: **outobject(current, port)**

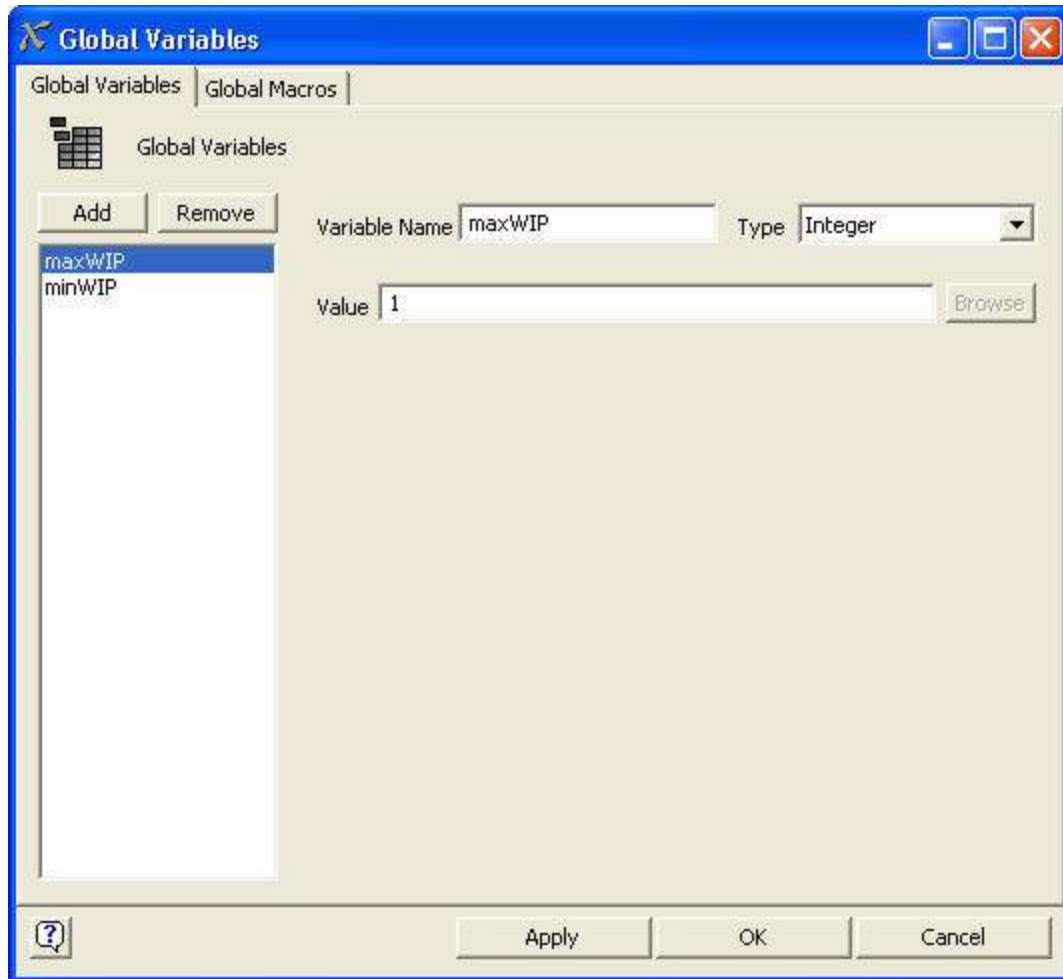
Dynamic Parameter 4: **NULL**

Dynamic Parameter 5: **NULL**

For more information on how Task Sequences work, refer to the Task Sequences section.

## Global Variables

The Global Variables window lets you create global variables that are accessible in flexscript and c++, as well as macro definitions.



Press the Add or Remove buttons to add and remove global variables. Then select the variable you would like to edit, and specify the name, variable type, and initial value(s) in the panel on the right. There are 8 variable types you can use, namely integer, double, treenode, string, integer array, double array, treenode array, and string array. For the array types, you can specify the size of the array and the initial value of each array element.

Once you have specified global variables, you can get and set the values of the global variables in flexscript or c++ code.

**Note on using C++** - If you access global variables in C++, you must make sure that the variables' names are globally unique names, meaning you do not use those names anywhere else in your C++ code except for when you are accessing the global variables themselves. Flexsim uses a macro definition to define these variables, so any other occurrences of the variable name may cause compile errors.

## Global Macros

The global macros page lets you make macro definitions. You do this by using `#define` statements, as follows:

```
#define MACRO_VAL1 5  
#define MACRO_VAL2 6
```

Once you have made these definitions, you can use them in your code.

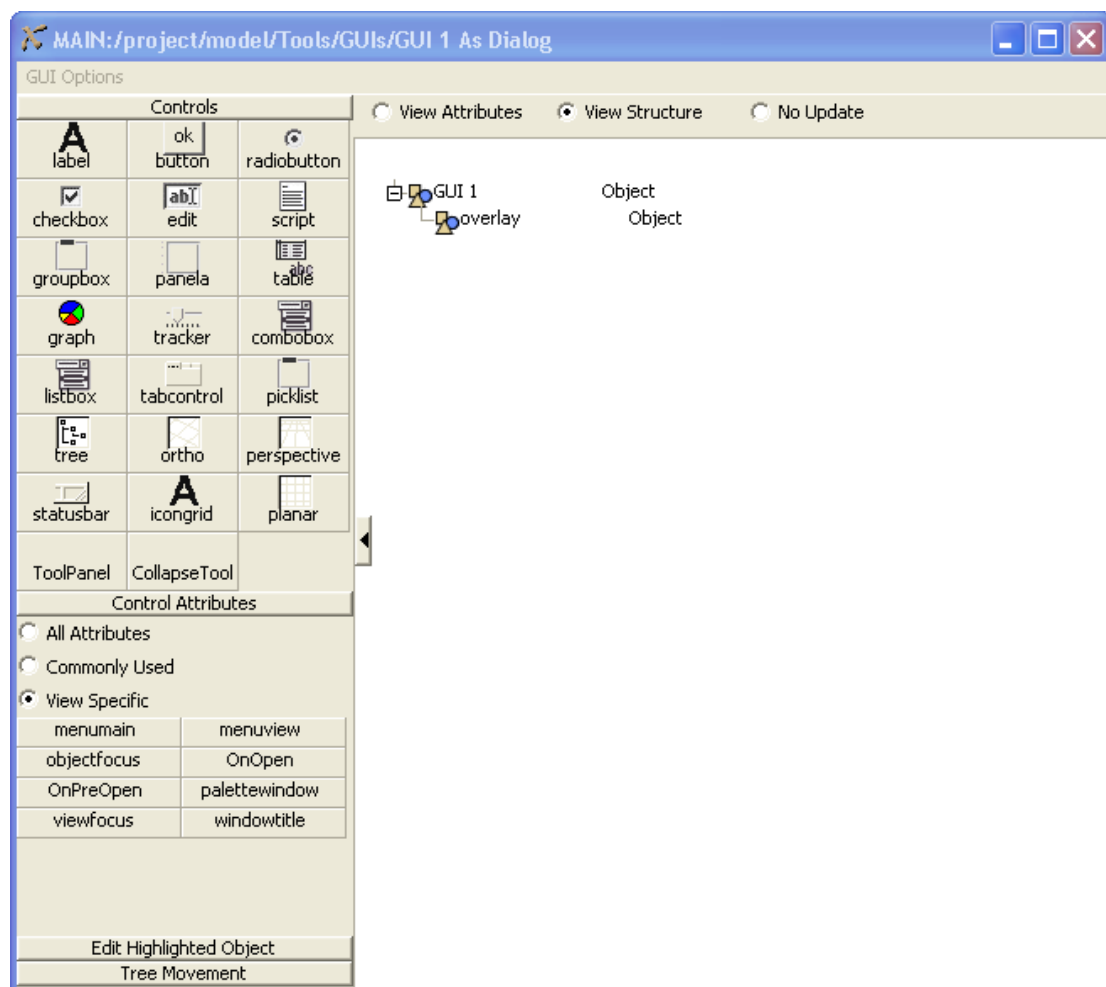
**Note** - The last line of the macro definitions must be a blank line, so make sure you include a hard return at the end of your definitions.

## Graphical User Interfaces

Graphical User Interfaces (GUIs) are accessed from the Tools menu. GUIs allow you to create your own window interfaces for your model and its objects.

Flexsim GUIs are made up of building blocks called views. Views are windows that perform specialized roles and can be combined hierarchically. Essentially views allow you to view and manipulate data in the Flexsim tree structure. Since data takes many forms, there are many types of views.

Once you have added a GUI by clicking the add button, and then click the edit button, two windows will open. The window on the left is called the GUI builder. This window provides you with several tools for building your GUI.



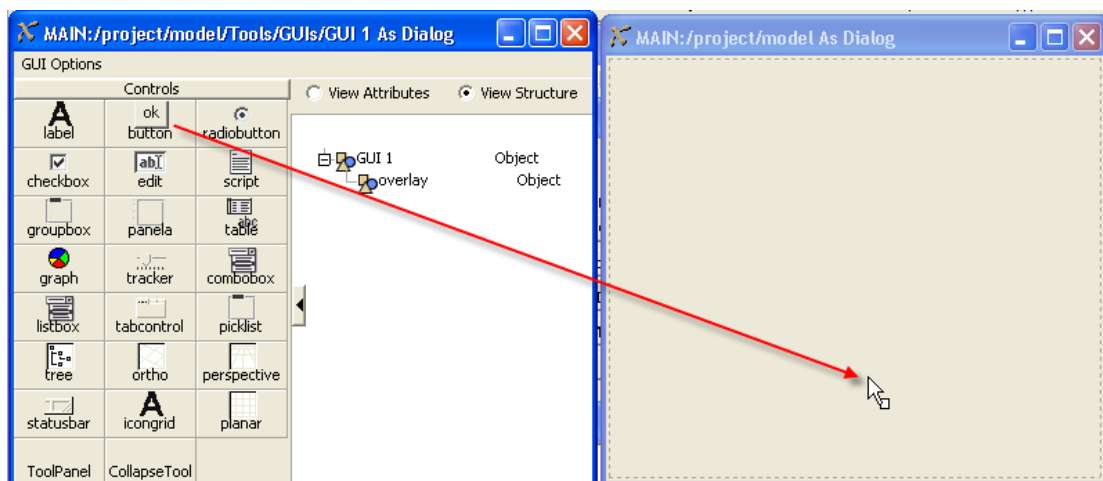
The window on the right is called the GUI canvas. This window shows what your GUI looks like. Initially it is blank, and GUI views are added to it in a drag-drop fashion from the GUI builder window.





## Building a GUI

As an example, we will create a simple GUI that allows you to edit the max speed, acceleration, and deceleration variables of an Operator object. First, let's drag a few simple views onto our GUI canvas. From the top panel of the GUI builder, drag a button onto the GUI canvas.



A button should now show up at the bottom of the GUI canvas.

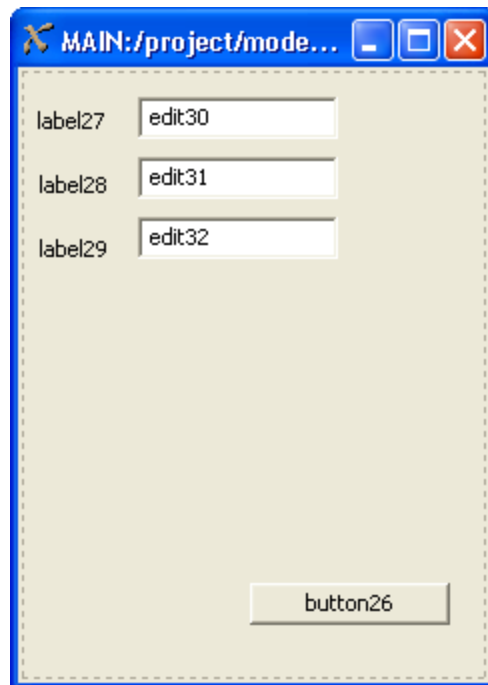


You can now select the button by clicking on it. When the button is selected, you will see a dotted outline around it, as well as a black square to the bottom right of it.



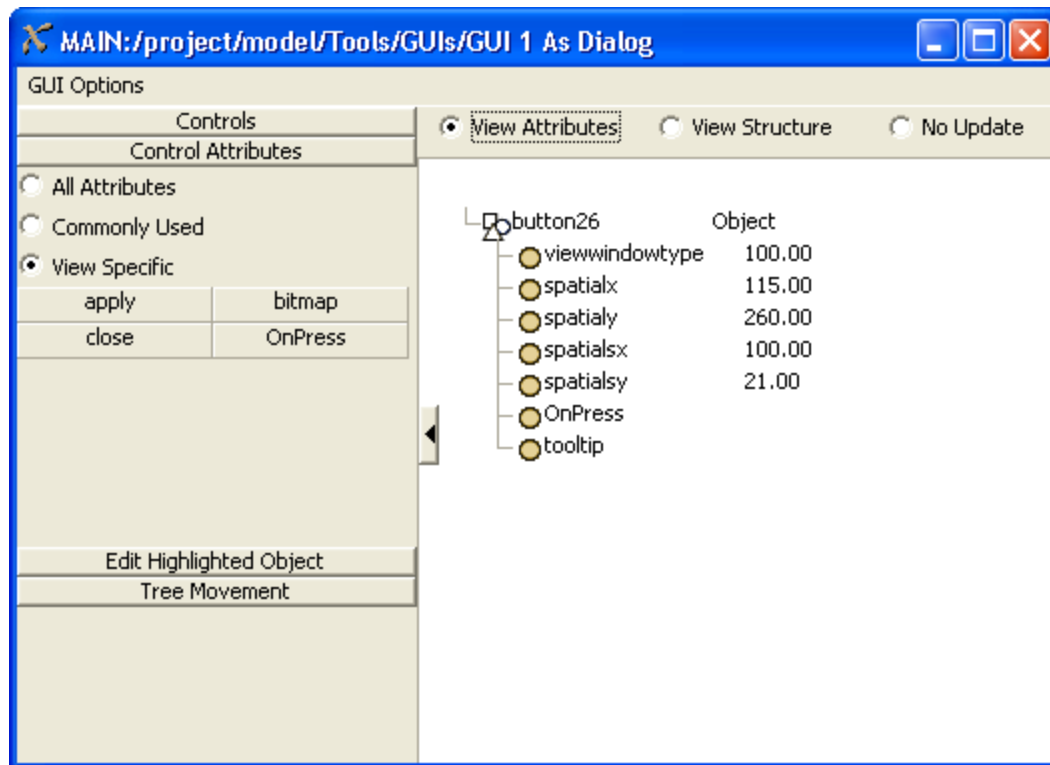
To move the button, just click and drag it to the location you want it at. To change the size of the button, click and drag the black square.

Now add three label views and three edit views onto the GUI canvas, as shown below. Note here that the term label is not the same as with the traditional flexsim label concept. Here a label view is a GUI view that just shows text.

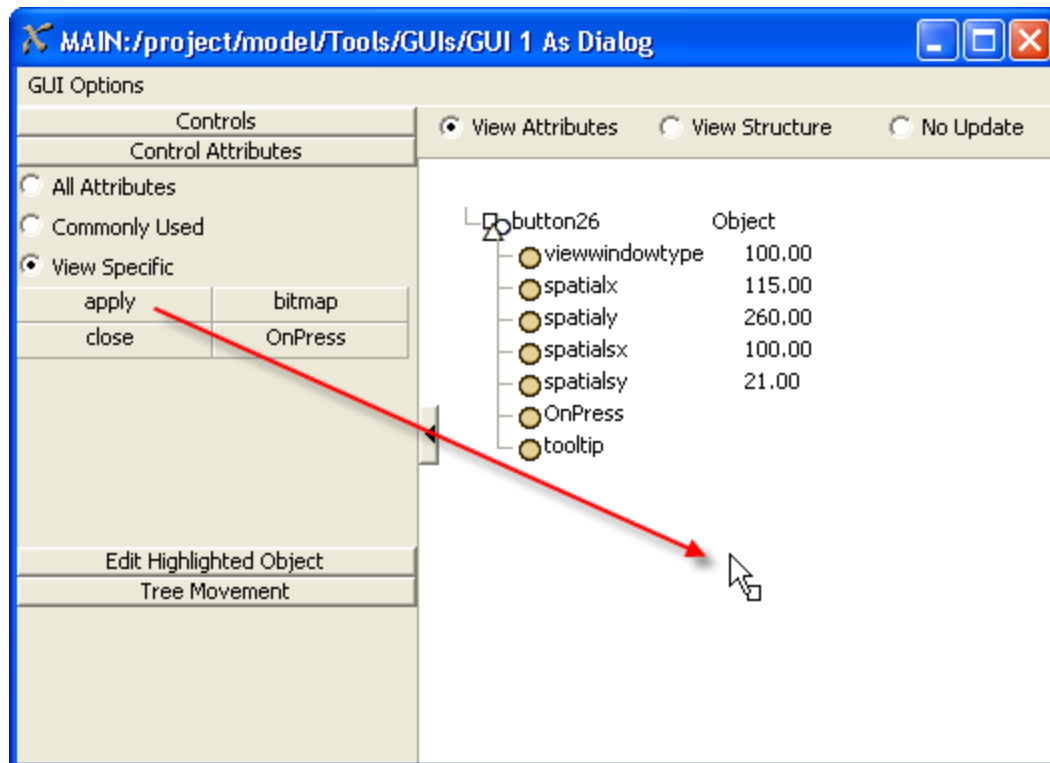


### Change View Names

Now we want to change the names and attributes of our GUI views. Collapse the controls toolbar by pressing on the "Controls" button in the gui builder. First, let's make the button an OK button that will apply the edits the user has made, and then close the window. Click on the button. Then click on the "View Attributes" radio button in the gui builder window. Notice that the button and its attributes are now shown in the tree view of the GUI builder.



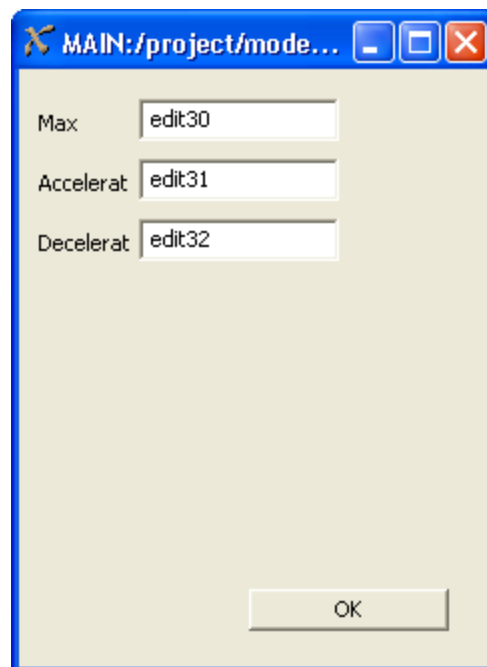
For the name of the button, enter "OK". On the left are some commonly used attributes that can be added to the button. To add an attribute, just drag it from the icon grid on the left to a blank area in the tree view.



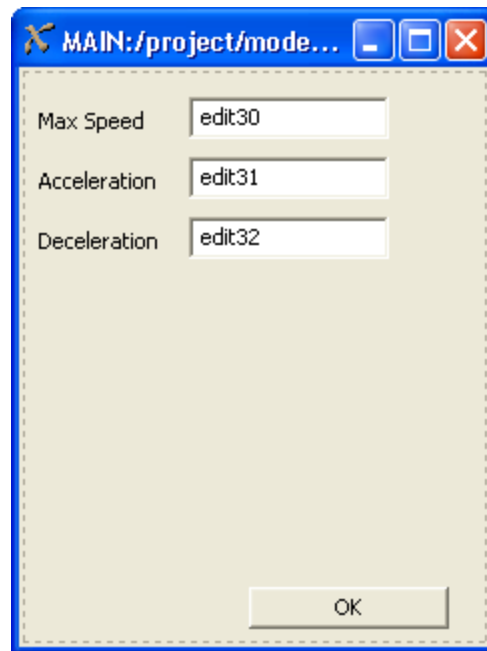
Add an apply attribute and a close attribute to the button. The apply attribute causes all coldlinks and hotlinks found in the view to be applied to their respective destination nodes when the button is pressed. Coldlinks and hotlinks will be explained later. The close attribute causes the window to close when this button is pushed.

Now click on the first label view. Again, the label and its attributes should appear in the tree view of the GUI builder. For the name of the first label view, enter "Max Speed". In the tree view you will need to click on the node once you have changed its name in order for the change to be applied. Now click on the second label view and set its name to "Acceleration". Then click on the last label view and change its name to "Deceleration".

Now that you have made a few changes to the GUI views, refresh the GUI canvas to see these changes. Click on the GUI canvas window and then press the F5 button. This will change the GUI canvas from editing mode to regular viewing mode. Now your window should look like a normal window without the dotted lines around it.

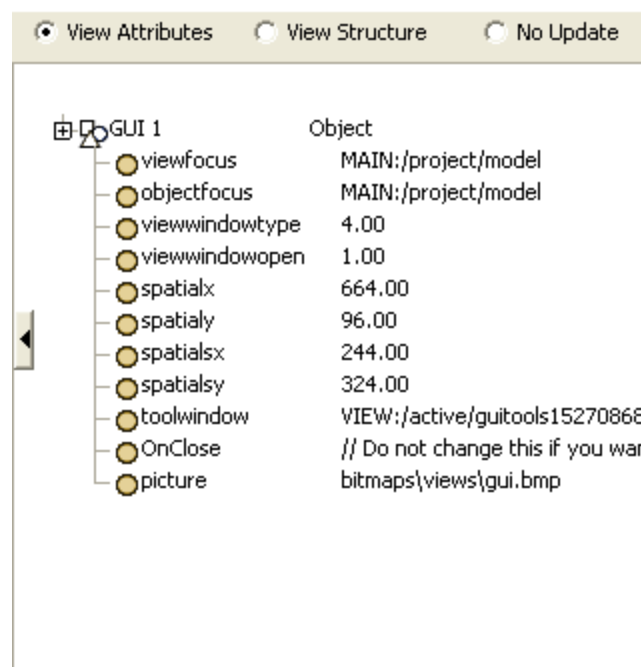


Notice that the label views are now not large enough to fit the text that they are showing. To fix this, go back into editing mode by selecting the GUI canvas window and pressing F5 again. Now rearrange the sizes and locations of your views so that there is enough room to show the entire text of the labels.



### Link the Edit Views

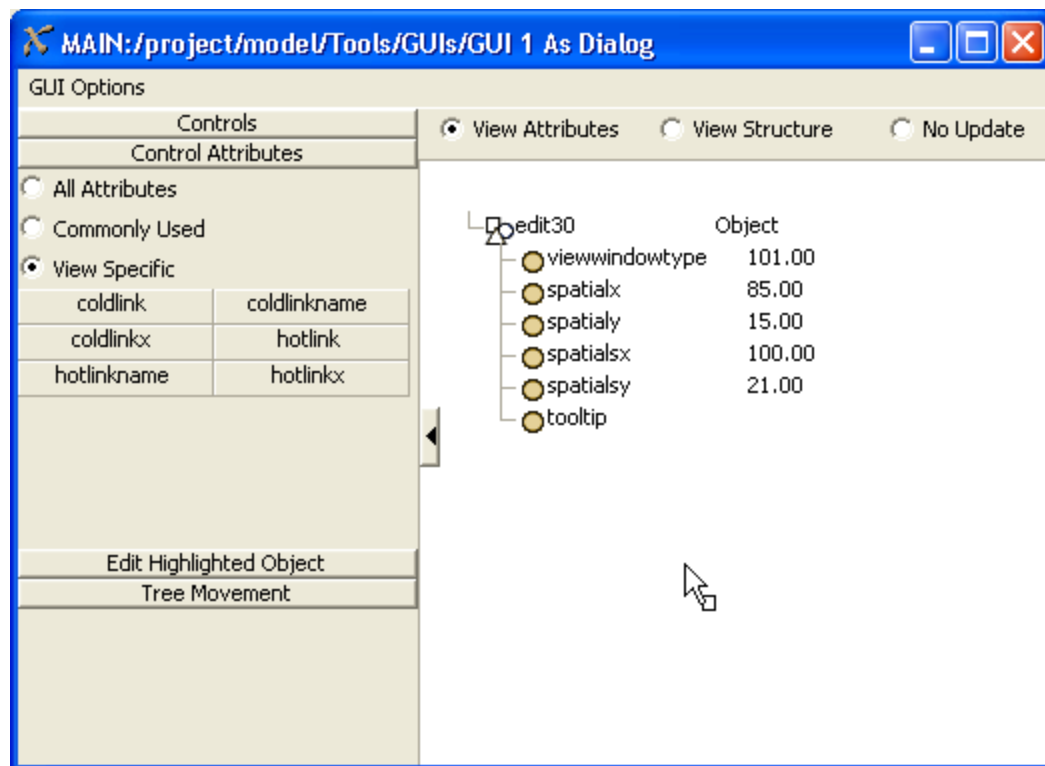
Now let's connect the edit views up to their proper nodes in the model. First let's explain some concepts. Click in a blank area in your GUI canvas. This will cause the tree view of the GUI builder to show the attributes of the main GUI window.



Notice that there is an attribute called `objectfocus`. Right now this attribute shows the path "MAIN:/project/model". Later on, though, once you've associated an Operator with this GUI, and then double-click on the Operator to open this window, the `objectfocus` attribute will be changed. It will specify a path to the Operator you are

editing. For example, you've associated an Operator named Bob with this GUI. When you double-click on Bob, an instance of this GUI is created, and its objectfocus attribute is set to the string path: "MAIN:/project/model/Bob". This is important to know when creating edit fields that are linked to our object. Now let's go back to the first edit field.

Click on the first edit view, and view its attributes in the tree. Add a coldlink attribute from the list on the left.



Now enter the following as the text of the coldlink attribute:  
 @>objectfocus+>variables/maxspeed

 coldlink      @>objectfocus+>variables/maxspeed

A coldlink attribute tells the view's text field to be linked to a certain node in the object's attribute tree. The link is "cold" because it gets the value only once when the window opens, and sets the value only when an apply button is pressed. A "hot" link, on the other hand, would continuously update its text field as the value in the model changes.

The coldlink text specifies a path to a node that the edit field is to be associated with. This path starts at the coldlink node itself, and should specify a path to the maxspeed variable node on the operator. The different symbols in the path are ways of specifying how to traverse the tree to the destination node. Their meanings are as follows:

@ : This symbol tells the traversal to go to the owner view of the current node. For this coldlink node, it is the main GUI window.

> : This symbol tells the traversal to go into a node's attribute tree.

+ : This symbol tells the traversal to read the current node's text as a path to an object, and go to that object.

/ : This symbol tells the traversal to go into the current node's sub-tree.

.. : Although this is not used in this example, this symbol tells the traversal to go to the current node's parent tree, or up one level.

The coldlink you have specified does the following:

1. Starting at the coldlink node, go to its own view, or the main GUI window (@).
2. From there, go into its attribute tree and find the attribute named objectfocus (>objectfocus).
3. Interpret the text of the objectfocus node as a path to a node, and go to that node (+). Remember that when we open this window for our Bob operator example, the objectfocus attribute will be changed to "MAIN:/project/model/Bob". So it will now go to our Bob operator in the model.
4. From there, go into the object's (Bob's) attribute tree, and find the node name variables (>variables).
5. From there, go into the node's sub-tree and find the node named maxspeed (/maxspeed).

Now connect the other two edit views. Click on the second edit view, then in the tree view add a coldlink attribute and specify its text as:

@>objectfocus+>variables/acceleration. Do the same for the last edit view and set its coldlink attribute to: @>objectfocus+>variables/deceleration.

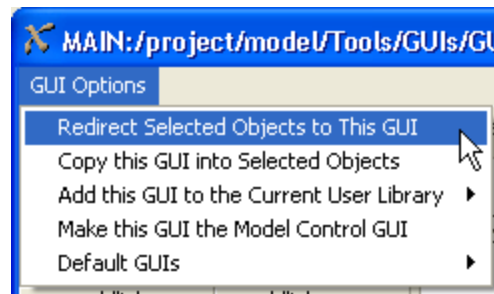
## Direct Model Objects to This GUI

Now that we've finished creating our GUI, let's direct an operator in our model to this GUI. Drag an operator into your model. Select it by holding Control or Shift down and clicking on the operator.

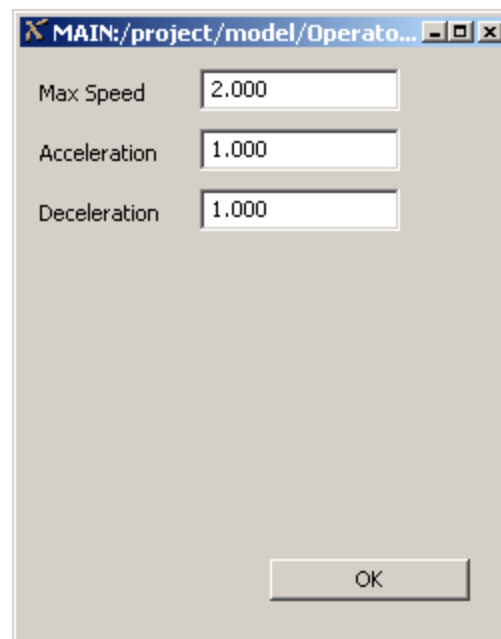


Now go back to the GUI builder window, and from its menu, select "Edit | Redirect Selected Objects to this GUI".





This will redirect all selected objects in the model to use this GUI instead of their usual parameters window. Now close the GUI builder window. This will automatically close the GUI canvas window as well. Now double click on the operator you have selected. This will open the GUI window you have designed instead of the normal parameters window.



Make some changes to the max speed, acceleration and deceleration values, then press the OK button to apply those changes. Open the window again to verify that your changes were applied correctly.

## Other GUI Builder Tools

### Attribute Lists

The list of possible attributes to the left of the GUI builder's tree view has three options for viewing attribute lists. They are: all attributes, commonly used attributes and view specific attributes.

Control Attributes	
<input type="radio"/>	All Attributes
<input type="radio"/>	Commonly Used
<input checked="" type="radio"/>	View Specific
coldlink	coldlinkname
coldlinkx	hotlink
hotlinkname	hotlinkx

**All Attributes** - If this option is selected, all possible attributes are displayed. Usually you will not need to use this option.

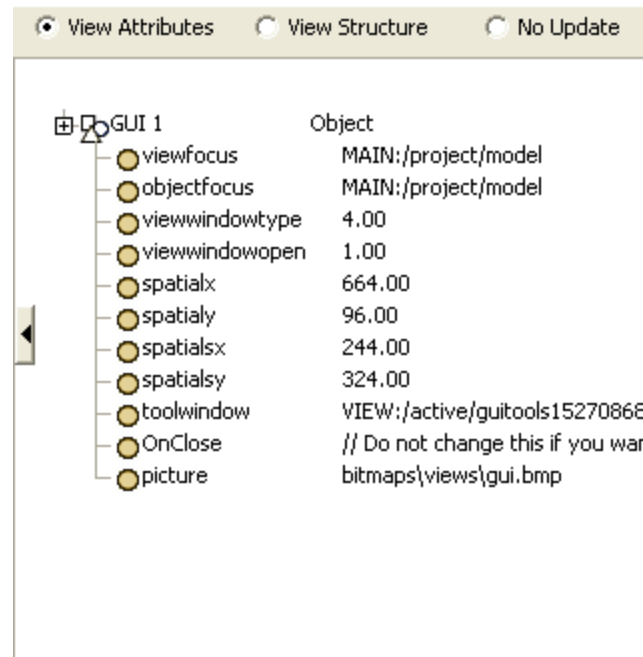
**Commonly Used** - If this option is selected, a list of commonly used attributes will be shown. These include things like alignment attributes, which allow you to anchor a view's position or size to the right or bottom of its container view.

**View Specific** - This option is the default option. If it is selected, then the attribute list that is shown will be specific to the type of view, or view, that you are currently editing. If you click on a button view, an attribute list will appear that is specific to a button. This includes an OnPress attribute, which is executed when the button is pressed. A label view will have a different set of attributes that are used for it, etc.

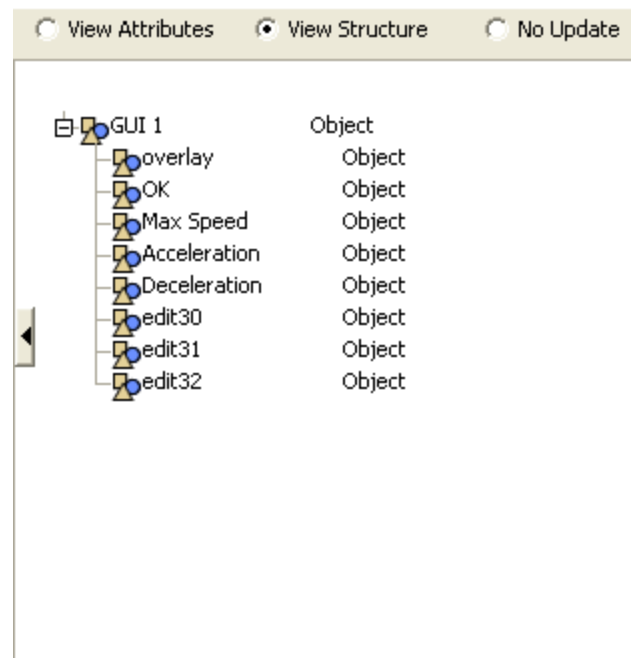
### Tree View - Viewing Attributes vs. View Structure

The GUI builder's tree view also has two options for viewing the tree. They are: view attributes and view structure.

**View Attributes** - This option will view the attributes of the currently selected view.



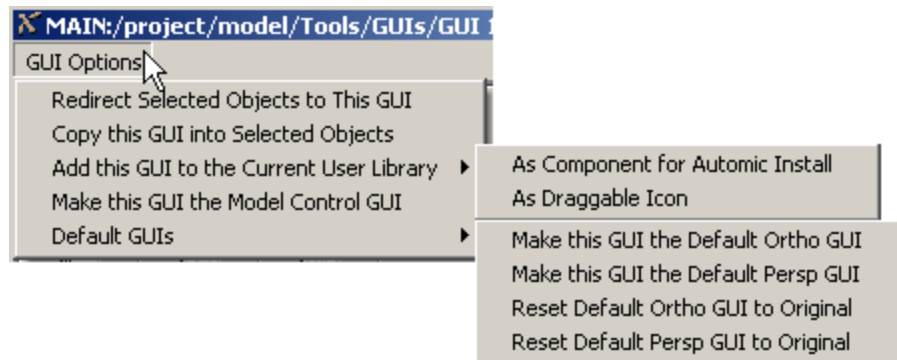
**View Structure** - This option will view the tree structure of the currently selected view. This is useful for rearranging and editing the structure of the GUI's tree.



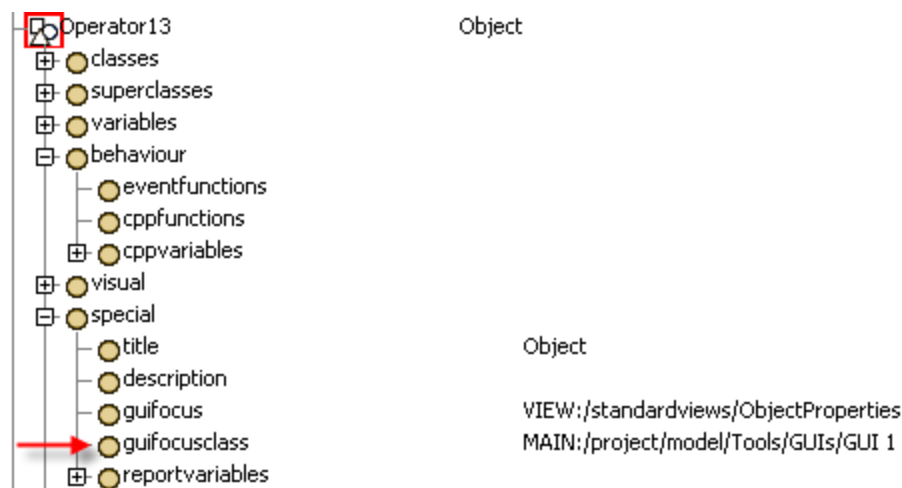
**No Update** - This option will cause the GUI builder to not update the tree focus when you click on an control in the gui canvas. Some users prefer this as it doesn't change the view every time you click in the canvas.

## GUI Builder Menu

The GUI builder's edit menu allows you to do several operations with the GUI once you have created it.



**Redirect Selected Object to this GUI** - This option will redirect the guifocusclass attribute of every selected object in the model to point to this GUI. To illustrate this, explore the model tree, and find the operator that you redirected to the GUI you created. Expand its attribute tree and the expand the node named "special". Inside of special there should be a node named guifocusclass. This node's text specifies a path to a window that will be opened when the object is double clicked on.



Notice that the path for this attribute is "MAIN:/project/model/Tools/GUIs/GUI 1". This is where our GUI is stored. When we selected the redirect option, it changed this path from the Operator's normal parameters page to our page.

**Copy this GUI into Selected Objects** - This option will create a complete copy of the GUI and store it inside of each selected object. To show exactly how this works, let's do it for our GUI. Go back into the toolbox and open the GUI builder and GUI canvas for the GUI you've built. Then, with our original operator still selected, select the menu option: Edit | Copy this GUI into Selected Objects. Now go back into the model tree view and look at what was done with the operator's guifocusclass attribute.



Now the guifocusclass attribute has been changed to ">special/guifocusclass/GUI 1". Also, a copy of the entire GUI that we created has been copied into the guifocusclass attribute.

Although you will not need to use the "Copy this GUI into Selected Objects" option if you are just using this GUI for this model, it is useful for portability purposes. Once you have created this GUI and copied it into the object, you can add the object to a user library, and then drag it into other models, and the GUI will be created with it.

**Add this GUI to the Current User Library** - This option will add the GUI to the currently selected library in the library icon grid. There are two ways by which you can add the GUI to the library, either as a draggable icon, or a component for automatic install. If you select the "As Draggable Icon" option, it becomes a visible icon in the user library. You can then drag it into other models, which will add the GUI to the model toolbox when dropped. You can also drag it from the user library onto an object in your model, which will store it on the object itself. You may want to use this option if you don't like the idea of a complete copy of the GUI stored on each of your objects when they are dragged out from a user library. If you select the "As Component for Automatic Install" option, this will add the GUI to the user library, but it will not be visible in the library's icon grid. Instead, it is added to the library as a component that will be automatically installed to your model when the user library is loaded. For more information on user libraries, refer to the user library documentation.

**Make this GUI the Model Control GUI** - This option causes the "Control" button on Flexsim's main toolbar to open this GUI when it is pressed. This button is called the Model Control button. Its exclusive purpose is to allow model builders to define

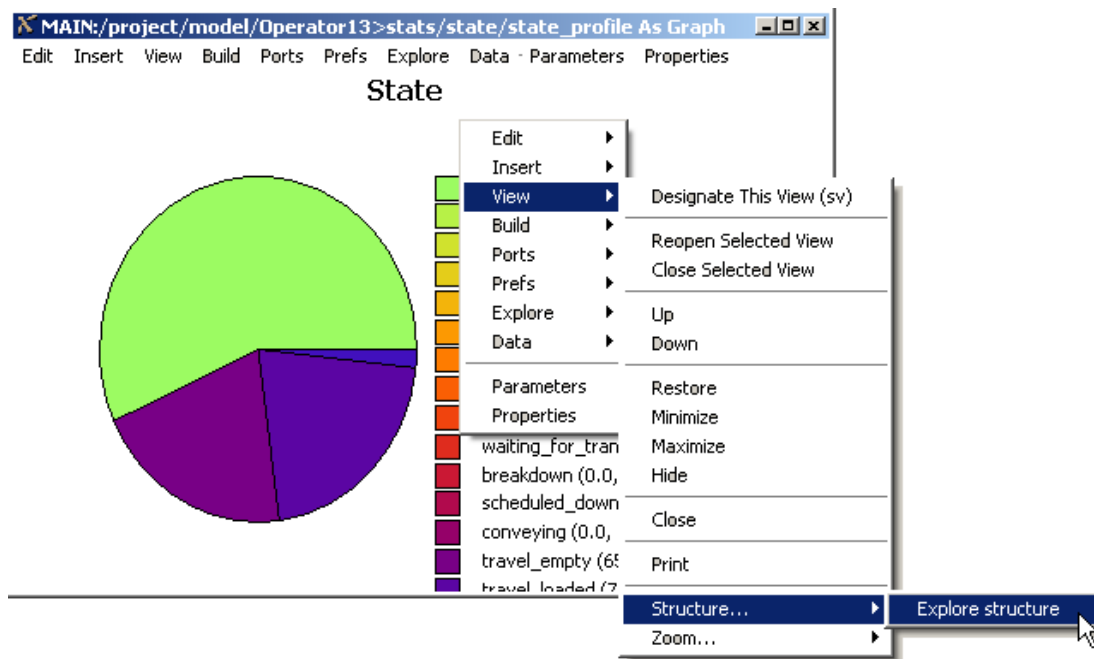
custom GUIs for controlling models and their parameters without having to change Flexsim's view tree. Note that this control button is only visible on computers with resolution above 1024x768.

**Default GUIs** - This sub-menu lets you also edit how other buttons on Flexsim's main toolbar operate. The "Ortho" and "Persp" buttons on the toolbar can be customized to open custom GUIs that you have created. By selecting "Make this GUI the default Ortho/Persp GUI" you can cause the Ortho/Persp buttons on the main toolbar to open your GUI. Select "Reset Default Ortho/Persp GUI to Original" to cause the Ortho/Persp buttons to revert back to their original GUIs. Note that making a custom GUI for the Ortho/Persp buttons only applies to the model you are editing. When you open another model, the Ortho and Persp buttons will reset to the original orthographic and perspective model views.

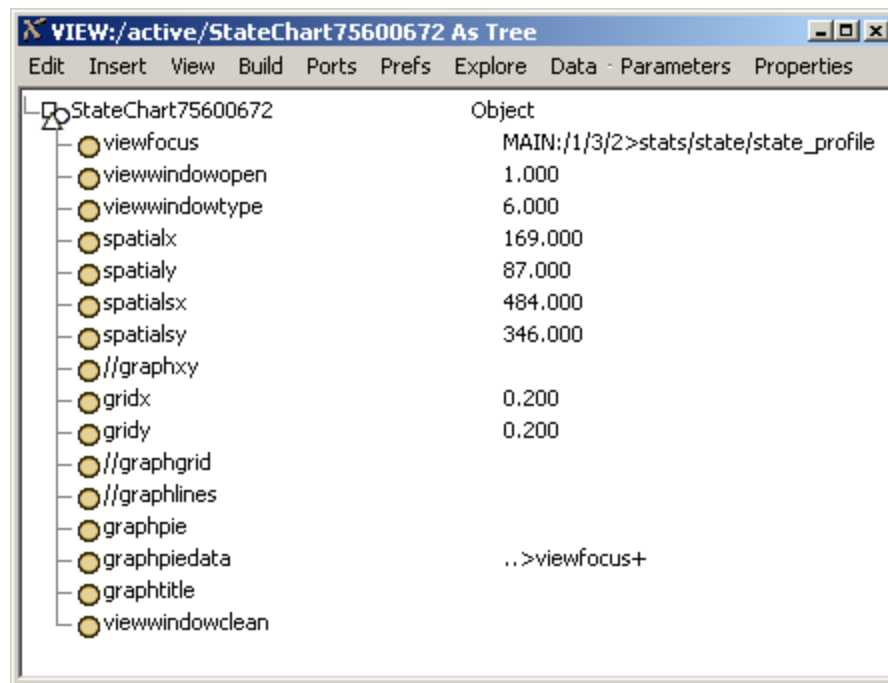
## GUI Building Tips

GUI building essentially consists of adding views to your GUI window and giving those views the appropriate attributes. While the "View Specific" option of the attribute list gives you some good hints of which attributes are appropriate to add, there are also other ways you can become more comfortable and experienced building GUIs.

1. First, and probably most important, look at other GUIs that have already been created. Every window in Flexsim uses the same underlying structure of windowing views and their attributes. If you want to create a GUI view that looks like something you've already seen before in Flexsim, open up that window, right click on it, and select "View | Structure | Explore Structure" from the pop-up menu.



This will open a tree window that shows the structure of the window. From here you can view the attributes of that window to see which attributes are present, so that you can add those attributes to your own GUI.



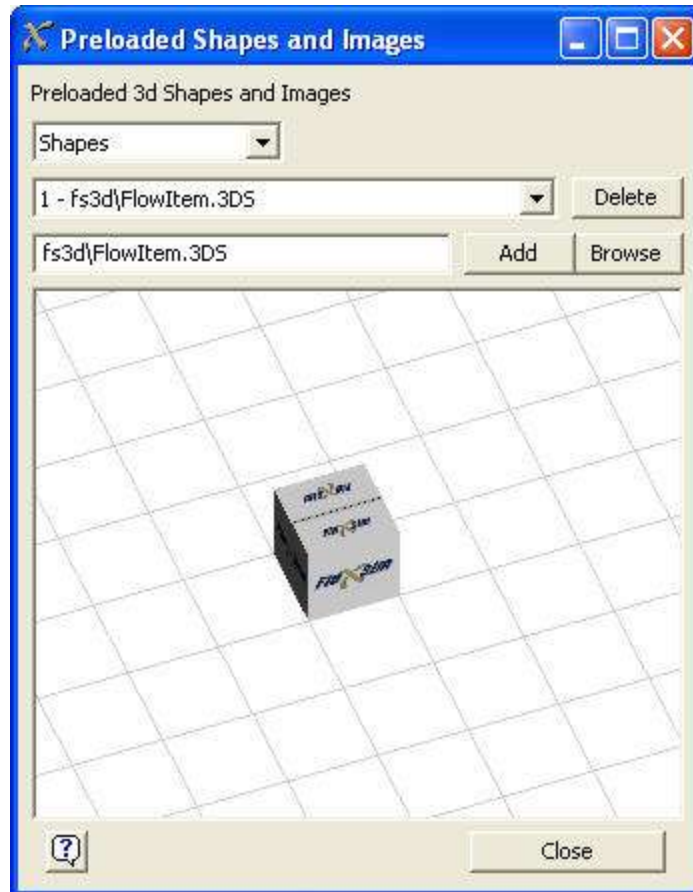
If you see a window view that you want to copy verbatim from an existing window to your GUI, find that view in the tree, right click on it and select "Edit | Copy", then go to your own GUI's tree structure. In the GUI's tree structure, create a new blank node by right clicking on the container view you want to place it in, and select "Edit | Insert Into". Then right-click on the new blank node, and select "Edit | Paste". This will paste the view into your GUI. Press F5 to refresh the view with its added view.

2. The second way you can become more familiar with building GUIs is by simple experimentation. Add an attribute that you've never added or seen before and see how it affects the GUI. If there is no change, then either the attribute doesn't do anything for this type of view, or it's somehow not implemented right. Fiddle around with different settings and values for the attribute to see if and how it changes the GUI. If still nothing changes, then just move on and try something else. If it does, great! That's one more tool that you have in your knowledge base.

3. The third way that you become more familiar with GUI building is just by lots and lots of practice. As you continue to build more and more GUIs, the speed and efficiency with which you do it will increase significantly, and you will be able to get a feel for what GUI styles are more user friendly.

## Import Media Files

This editor allows you to add 3d shapes and images to be pre-loaded to the model, as well as get the path string for shapes that are already loaded. You will usually only need to use this editor if you are dynamically changing the shape of objects during the simulation. Otherwise, just select a shape from an object's Properties window.



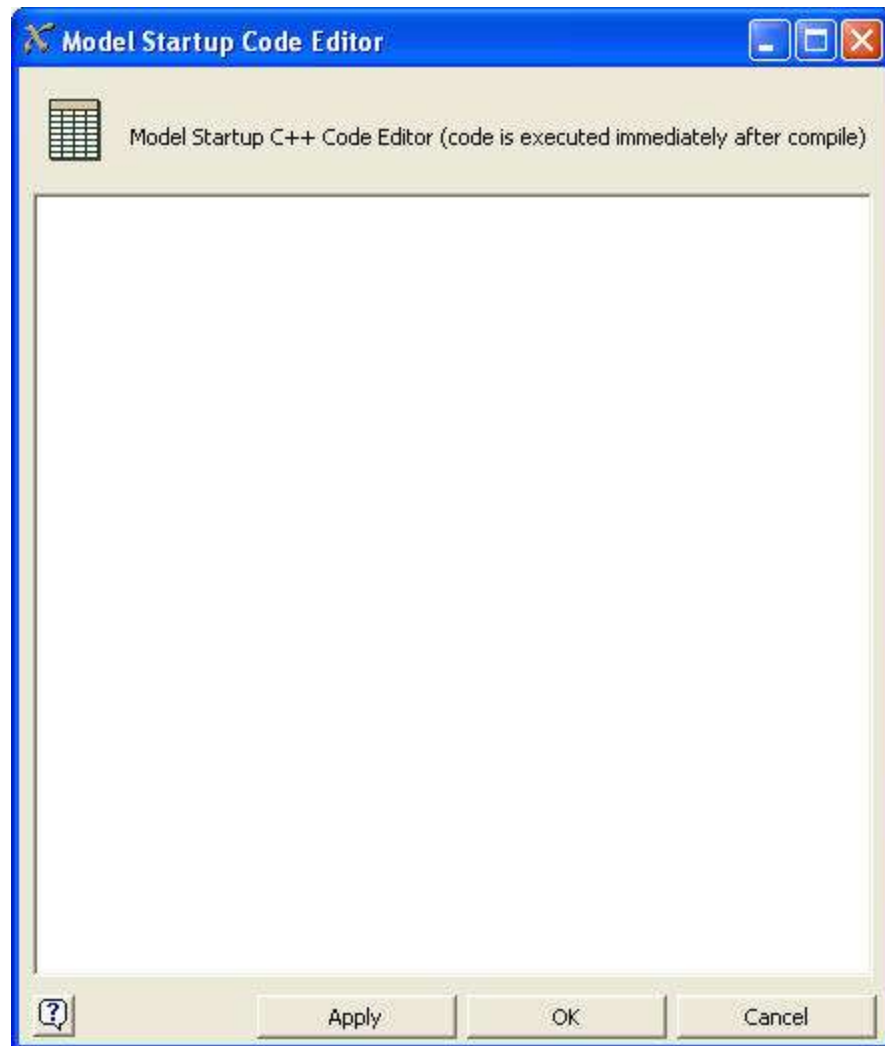
To work in this editor, select either Shapes or Images from the drop-down box at the top. Then select a shape from the second drop-down box. When you select a shape, the text box updates its text to show the path of the currently selected object. To add a new object, click the Browse button, find the .3ds, .wrl, .dxf, or .stl file for shapes, or a .bmp or .jpg file for images, and click Open. Then press the Add button to add the 3d object or image to the pre-loaded list. Press the Delete button to remove shapes that have been added to the list.

The number next to each option in the drop-down list can also be used if you are writing code that references a texture or shape index.



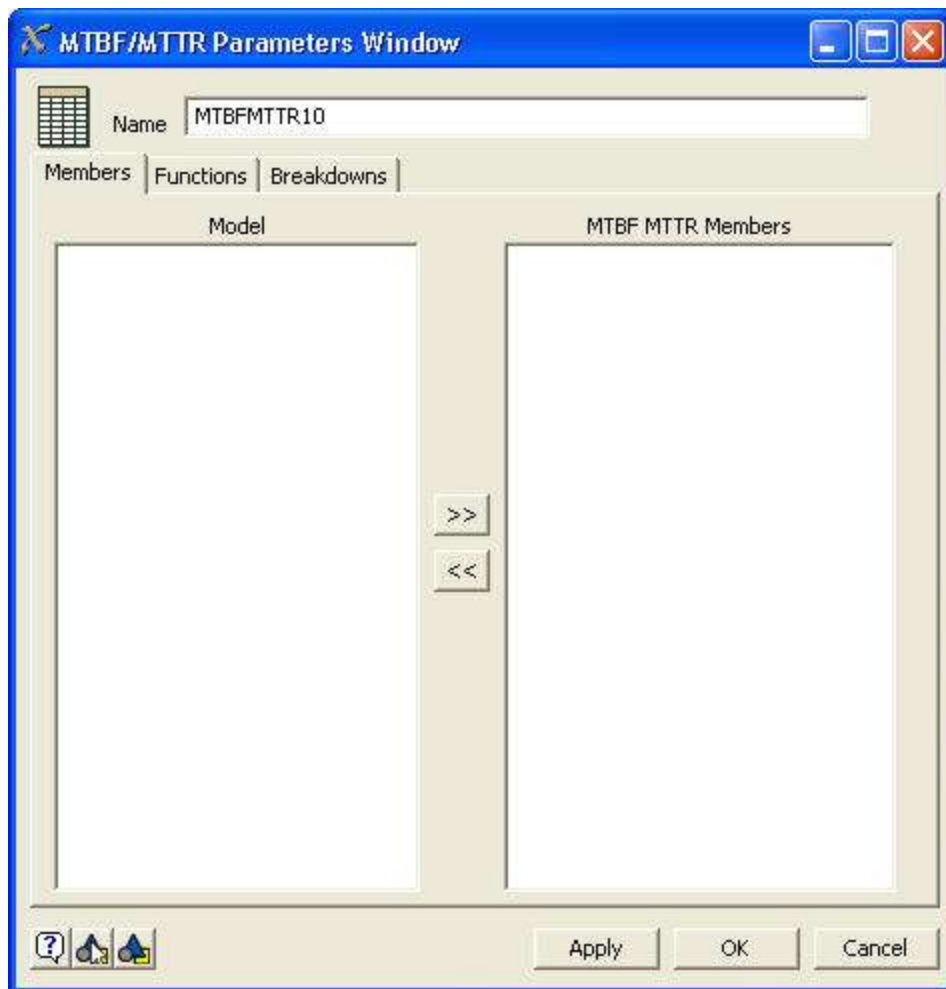
## Model Startup Code

The model startup code editor lets you write code that will be executed immediately after the model is compiled. Note that this is only executed once for each compile, and not every time you reset the model.





## MTBF/MTTR

MTBF MTTR objects are accessed in the Tools menu. They are used to set random breakdown and recovery times for groups of objects in the model. Each MTBF MTTR object can be connected to more than one object in the model. And each object can be controlled by more than one MTBF MTTR object. The MTBF MTTR object allows you to also specify what state the objects will go into when they go down. A model may contain any number of MTBF MTTR objects. The MTBF MTTR window is split into two tab pages.



**Name** - The name of the MTBF MTTR object. This should be something descriptive and easy to remember. For example, “Forklift Control” or “Random Inspection”.

**Add to User Library** - The  and  buttons at the bottom let you add this object to the currently active user library. The buttons add it as a draggable icon or as a component for automatic install, respectively. For more information, refer to the user

library documentation. Note that this option is not available if you are editing a label table.

### Members Page

In this page you can specify the list of member objects for this MTBF MTTR object. The panel on the left is a list of model objects. The panel on the right is a list of the members of this MTBF MTTR object. Select an object from the left panel and click the **>>** button to add the object to the member list. Select a member in the right panel and click the **<<** button to remove it from the list.

### Functions Page

In this page you can specify the times to go down and resume, down and resume triggers, as well as the down and resume functions.

The screenshot shows a software interface for configuring functions. It features a tab labeled "Functions" and seven distinct configuration sections. Each section consists of a title, a text input field, and two small icons (a help icon and an "Apply" button labeled "A").

- First Failure Time:** The text field contains "An Exponential distribution with location value of 0 and scale valu...".
- MTBF:** The text field contains "An Exponential distribution with location value of 0 and scale valu...".
- MTTR:** The text field contains "A Uniform distribution with a minimum value of 500 and a maximu...".
- Down Function:** The text field contains "Execute stopobject() id 1 priority 0".
- Resume Function:** The text field contains "Execute resumeobject() The resumeobject() command is class de...".
- OnBreakDown:** The text field contains "Do nothing".
- OnRepair:** The text field contains "Do nothing".

**First Failure Time** - This pick list returns the time of the first failure. See the time pick list.

**MTBF** - This pick list returns the Mean Time Between Failure for the objects controlled by this MTBF MTTR object. This function determines how long those objects will run before they go into a broken-down state. The MTBF time is specifically defined as the span between the time that the object resumes from its last down period and the time that it starts its next down period. See the time pick list.

**MTTR** - This pick list returns the Mean Time To Repair for the objects controlled by this MTBF MTTR object. This function determines how long they will stay in a broken-down state before resuming normal operations again. All of the controlled objects will go back to their original states at the same time. See the time pick list.

**Down Function** - This pick list is executed when the objects in the member list go down. It is executed once for each object in the member list. Here is where you specify what to do to stop the object.

**Resume Function** - The pick list is executed when the objects in the member list resume their operation. It is executed once for each object in the member list. Here is where you specify what to do to resume the object.

**OnBreakDown** - This pick list is fired at the same time as the Down Function, but it is only executed once, instead of once for each object in the member list. See Down/Resume Trigger.

**OnRepair** - This pick list is fired at the same time as the Resume Function, but it is only executed once, instead of once for each object in the member list. See Down/Resume Trigger.

## Breakdowns Page

The breakdowns page let's you configure more details of how the objects will be broken down.

**Down State** - This specifies the state that the object will go into when it goes down.

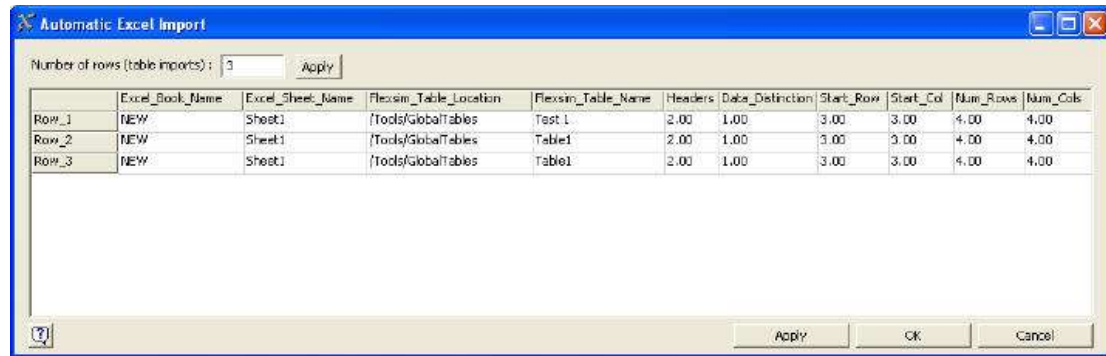
**Break down members individually** - If this box is checked, the MTBF MTTR object will create a separate thread of down and resume events for each member object. If it is not checked, all member objects will go down and resume at the same time.

**Apply MTBF to a set of states** - This box only applies if the MTBF MTTR breaks down members individually. If it is checked, then the MTBF time will only be applied to a subset of the object's state. For example, if machine break down data only applies for when the machine is actively processing, then you would use this field. If checked, you will add a set of states to the list on the right from the list of possible states on the left.

**Accuracy** - This field only applies if the MTBF MTTR uses a subset of the objects' states for its MTBF time. Usually this value will be 100, or 100% accuracy. However, if the subset of states represents a very small portion of the total time of the member objects' state times, then the accuracy value can be used to optimize for run speed. For example, if an MTBF is applied to an object's "Waiting for Operator" state, but the object is only in that state 5% of the time, an accuracy value of 100 would cause the MTBF to perform several checks before bringing the object down. If you then change the accuracy value to 5, then the MTBF will probably need to do much fewer checks before bringing the object down.

**Range Cutoff** - This field only applies if the MTBF MTTR uses a subset of the objects' states for its MTBF time. Usually this value will be 0. However, if the subset of states represents a very small portion of the total time of the member objects' state times, then the range cut-off value can be used in conjunction with the accuracy value to improve run speed. This specifies a range within which the MTBF can go ahead and bring the object down. For example, if the next down is scheduled for when the object's subset of states has reached 10000, and the range cutoff is 100, the MTBF will do a check, and if the state subset is above 9900, it will go ahead and bring the object down.

## Multiple Table Excel Import



### Overview

The Multiple Table Excel Import (MTEI) was designed to make importing multiple worksheets from more than one workbook very fast and easy to do. The MTEI is capable of automating much of the import process in terms of the table size and cell data type. If you allow the MTEI to be more automatic in its implementation it is extremely useful for importing data that will change over time.

### Filling out the Import Table

When you edit the MTEI table you will notice that there are 10 columns that need to be filled out for every import row. These columns define where the data is coming from and going to, as well as, how the data is to be interpreted and formatted.

### Excel\_Book\_Name

The Excel Book Name column is where you define the name of the Excel workbook that contains the information that you want to import. There are four main ways to enter information into this column depending on what spreadsheet you want to use.

#### Unknown workbook name or location (“NEW”)

If the name or location of the workbook that you want to use are unknown or will change over time then you can type in the word “NEW” in all caps. Typing “NEW” into this column will cause the browse window to open prompting the user to find the Excel file they want to use. This is an extremely useful option when the input data will change with different runs or users.

#### Same as the previous location (BLANK)

If you want to continue to use the same workbook as the previous line then you can leave this column blank. This is the recommended option when you are importing information from multiple sheets in the same workbook. **Note:** Do not leave this column blank on the first line

#### Absolute Path (ABSOLUTE)

If the location and name of the workbook will not change for the entire use of the model then you can enter the absolute path of the Excel workbook. For example: c:\tempdirectory\myfile.xls **Note:** The ".xls" extension is essential in order for Flexsim to find the right Excel spreadsheets.

## Relative Path (RELATIVE)

If the name of the workbook will not change and the relative directory of the workbook will not change for the life of the model then you can enter the relative path of the Excel Workbook. The path entered must be relative to the install directory of Flexsim. For example: userprojects\myproject\myfile.xls

## Excel\_Sheet\_Name

The name of the Excel sheet that contains the import information should be entered into this column. For example: "Sheet1". If the MTEI does not find the name of the sheet because it does not exist or has been entered wrong then it will pause the import and alert you of the problem. You will be given the option to exit the import completely or skip the offending import row and continue with the next one. **Hint:** Look for spaces at the beginning and end of the name if you are alerted that a sheet name does not exist.

## Flexsim\_Table\_Location

The path of the Flexsim node that contains the table should be entered into this column. The default path for this column is the path for the Global Tables "/Tools/GlobalTables". An example of a path that will import the data into the schedule of a source is "/Source1>variables". **Note:** Do not enter the name of the table node; that is entered in the next column.

## Flexsim\_Table\_Name

The name of the Flexsim table node is entered into this column. If the target table is a Global Table then you just need to enter the name of the Global Table. If you are trying to import the data into the schedule of a source then you would enter the name "schedule". **Hint:** If you are importing into a Global Table and the table does not exist the MTEI will ask you if it should create the table for you.

## Headers

Implementing headers will cause the MTEI to import the column and or row names for the table. This is useful for helping you to identify the columns and rows later in Flexsim. The values that can be entered in the headers column and their meanings are listed below:

- 0 – Do not import any header information
- 1 – Import the header information for the rows only
- 2 – Import the header information for the columns only
- 3 – Import the header information for the rows and columns

The row or column for the header information is automatically calculated. The header information should always come before any data distinction information or actual data.

## Data\_Distinction

Data distinction refers to the way in which the MTEI will interpret the incoming data and the way that it will format the Flexsim table. The data distinction allows you to import tables with both numeric and text data. The values that can be entered in the data distinction column and their meanings are listed below:

- 0 – No distinction – all data is assumed to be numeric
- 1 – Automatic – data distinction is based on the first character of the table cell
- 2 – Row – data is defined by the row above the first row of data in Excel
- 3 – Col – data is defined by the column before the first column of data in Excel

For options 2 and 3 the data distinction row or column in Excel contains a number value that determines what type the data will become in Flexsim for the entire row or column that the number is in front of. The data distinction row or column should always be placed before the actual data but after any header information. The values that can be entered in the data distinction row or column in Excel are listed below:

- 1 – Numeric data
- 2 – Text data
- 3 – Flexscript data
- 4 – C++ data

The MTEI will automatically format the nodes in the table to be numeric or text and to be built as Flexscript or C++ depending on the data distinction. If the table imports Flexscript or C++ data the MTEI will prompt the user to recompile the model after the import is finished.

### Start\_Row and Start\_Col

The start row and start col columns determine where the MTEI will look on the Excel sheet for the data it needs to import. Enter the starting location for your data in these cells not the location of the headers or data distinction information. If you leave the values for these cells at 0 the MTEI will automatically adjust where it imports the data from. So if you always leave your data at the top left of the worksheet you will never need to enter a value other than 0 in these cells regardless of whether or not you have headers or data distinction information in front of the data.

### Num\_Rows and Num\_Cols

The Num\_Rows and Num\_Cols columns determine the amount of rows and columns that the MTEI will import. So if you have 5 rows of data then you would enter 5 in the Num\_Rows column. If you set these entries to 0 the MTEI will automatically calculate the number of rows or columns for you by searching for the first empty cell in a chosen "search" row or column. The "search" row or column will be based on the header or data distinction row or column if they are available. If there is no header or data distinction row or column, then the "search" row or column will be the first row or column of data. Letting the MTEI calculate the number of rows or columns for you is a great way to allow the developer and or user of the model to add or delete rows and/or columns from the table as they are necessary without having to worry about changing any other values.

**Note on automatic resizing:** The MTEI automatically sizes the Flexsim table that it is importing into to fit the size of the table that it is importing.

### Related Topics



Single Table Excel Import  
Single Table Excel Export  
Excel Interface

## OptQuest Optimizer

The OptQuest optimizer lets you optimize variables in your model to maximize certain output variables.

The screenshot shows the Optquest Optimizer window with the following sections:

- Variables:** A table with columns: Current Value, Best Value, Name, Type, Lower Bound, Upper Bound, Step, and Model Variable. It contains three rows:
 

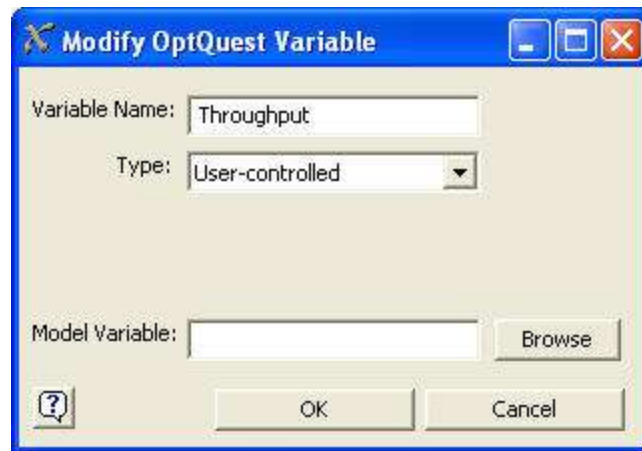
Current Value	Best Value	Name	Type	Lower Bound	Upper Bound	Step	Model Variable
35.000	33.000	QueueContent	OPT_TYPE_INTEGER	0.000	10.000	1.000	maxcontent
865.000	891.000	Throughput	OPT_TYPE_USERCONTROLLED	0.000	10.000	1.000	stats_input
21.000	33.000	MaxReachedContent	OPT_TYPE_USERCONTROLLED	0.000	10.000	1.000	stats_contentmax
- Constraints:** An empty table with Add and Delete buttons.
- Objective Function:** Radio buttons for Minimize and Maximize. The Maximize option is selected, and the expression is  $\text{Throughput} - ((\text{QueueContent} - \text{MaxReachedContent}) * 10)$ .
- Maximum Time for Optimization:** A text box with the value 10000 and an AutoStop checkbox that is checked.
- Scenarios:** Text boxes for Maximum Scenarios (40.000), Current Scenario (0.000), Current Solution (0.000), Best Solution (0.000), and Simulation time per Scenario (3000).
- Replications:** A checked checkbox for Perform Multiple Replications per Scenario. Below it are text boxes for Minimum number of replications (1) and Maximum number of replications (10). Early Exit Criteria is set to Confidence Interval Satisfied, with Percent Confidence at 80% and Error Percent at 10.

Buttons at the bottom include Apply, Optimize, and Close.

### Decision Variables

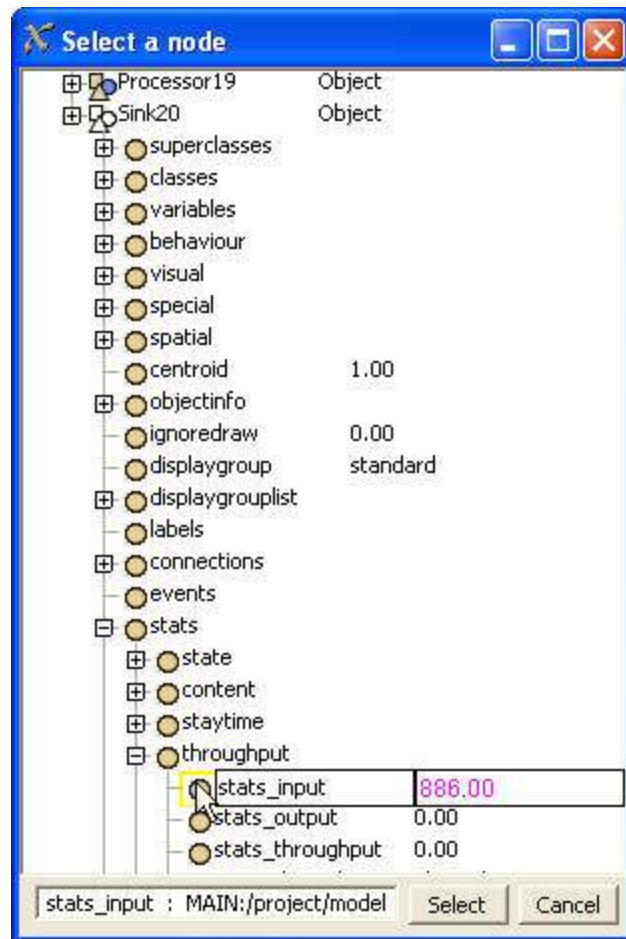
The first part of designing an optimization is to define the decision variables for the model. The main decision variables for an optimization can usually be chosen by restating the problem you want to solve. For example, one problem may be: how many machines do we need in this area to get the best throughput? This statement defines the decision variables for the model: the maximum content value of the processor, and the throughput value of the model. Note that these two variables have different roles. The maximum content is a value we want to change and experiment with, while the throughput is our feedback, reflecting the results of our changes.

To add a decision variable, click on the Add button in the Variables panel. This adds a new variable to the variables table. Select this variable by selecting any cell on the row of the new variable, then click the Modify button. This brings up a window to edit this new variable.



Each decision variable has an associated name, which will be used by OptQuest. Each variable also has an associated type, like continuous, integer, or user-controlled. User-controlled variables are the "feedback" variables. They are not changed by the OptQuest experimentation, but are used as output variables to get feedback for how well different scenarios do. All other variable types will be changed and experimented with during the optimization. Refer to the [OptQuest documentation](#) for more information.

Once you have specified the variable name and type, click on the Browse button to associate this variable with a node in the model. This will open a Tree Browse window to select the node that holds this maximum content value. You must select a node that has number data on it, or the optimization will not work properly.



## Constraints

Once you have defined your decision variables, you will want to define constraints for the optimization. During the optimization, the optimizer will try several scenarios on the decision variables. Constraints are used to nullify certain scenarios if these constraints are not properly met, so that the optimizer doesn't choose an invalid scenario as the optimum. Each constraint has an expression, such as "MaxNrofProcessors < NrofProcessorsUsed + 5". To add a constraint, press the Add button, and then fill in the equations column.

Objective Function

☐ Minimize ☒ Maximize

Throughput - (((QueueContent - MaxReachedContent)\*10)

Maximum Time for Optimization: 10000 ☒ AutoStop

Scenarios

Maximum Scenarios: 40.000 Current Scenario: 0.000

Current Solution: 0.000 Best Solution: 0.000

Simulation time per Scenario: 3000

## Objective Function

The objective function is an expression that you want to maximize or minimize. This could be a simple expression like "Throughput" if you have a decision variable called Throughput. It could also be a revenue vs. cost estimation. For example, if each product produced yields \$5.00, and the cost for each machine (weighted by the length of the simulation run) is \$50, then the objective function could be  $(\text{Throughput} * 5.00) - (\text{MaxNrofProcessors} * 50.00)$ .

## Stop Conditions

**Maximum Time for Optimization** - This is the maximum real time that the optimizer will spend in its optimization.

**AutoStop** - If this box is checked, the optimization stops when the value of the objective function stops improving. The Flexsim current setting is to stop when the objective function value of the best solution found does not vary by at least 0.0001 after 100 iterations.

## Scenarios

**Maximum Scenarios** - This is the maximum number of different scenarios that the optimizer will try. A scenario is one configuration in the optimizer's search.

**Current Scenario** - This is the current scenario number being tested.

**Current Solution** - This is the value of the objective function for the current scenario.

**Best Solution** - This is the value of the object function for the best scenario so far.

**Simulation Time per Scenario/Real Time per Scenario** - This is the maximum simulation time that the optimizer will spend for each scenario. The optimizer stops a scenario as soon as this is met.

## Replications

If you want to run the simulation several times for a given scenario to increase confidence of the mean of the objective function, use the Replications panel to specify how many replications to run.

The screenshot shows the 'Replications' panel with the following settings:

- ☒ Perform Multiple Replications per Scenario
- Minimum number of replications: 3.00
- Maximum number of replications: 100.00
- Early Exit Criteria: Confidence Interval Satisfied (dropdown menu)
- Percent Confidence: 80% (dropdown menu)
- Error Percent: 5.00

**Perform Multiple Replications per Scenario** - If this box is checked, the optimizer will perform more than one replication per scenario.

**Minimum number of replications** - This is the minimum number of replications to run for each scenario. If there is no Early Exit Criterion then the optimizer will always run the minimum number of replications.

**Maximum number of replications** - This is the maximum number of replications to run for each scenario. If there is an Early Exit Criterion then the optimizer will run the scenario until the criterion is met, up to this maximum number, after which the optimizer will stop the scenario.

**Early Exit Criterion** - This allows the optimizer to stop running further replications for the same scenario, based on the criteria you select. If you've selected "Confidence Interval Satisfied" the optimizer will stop replications once it can determine the objective function's true mean for the scenario with the given confidence and error percentages. For example if you specify an 80% confidence and a 5% error, then the optimizer will stop running replications as soon as it is 80% confident that the true mean of the objective function is within 5% of the mean sampled. If "Best Solution Outside Confidence" is chosen, then the optimizer will stop replications for a given scenario if it can determine, within the given confidence and allowable error percentages, that the best solution can never be met with this scenario.

## Optimizing

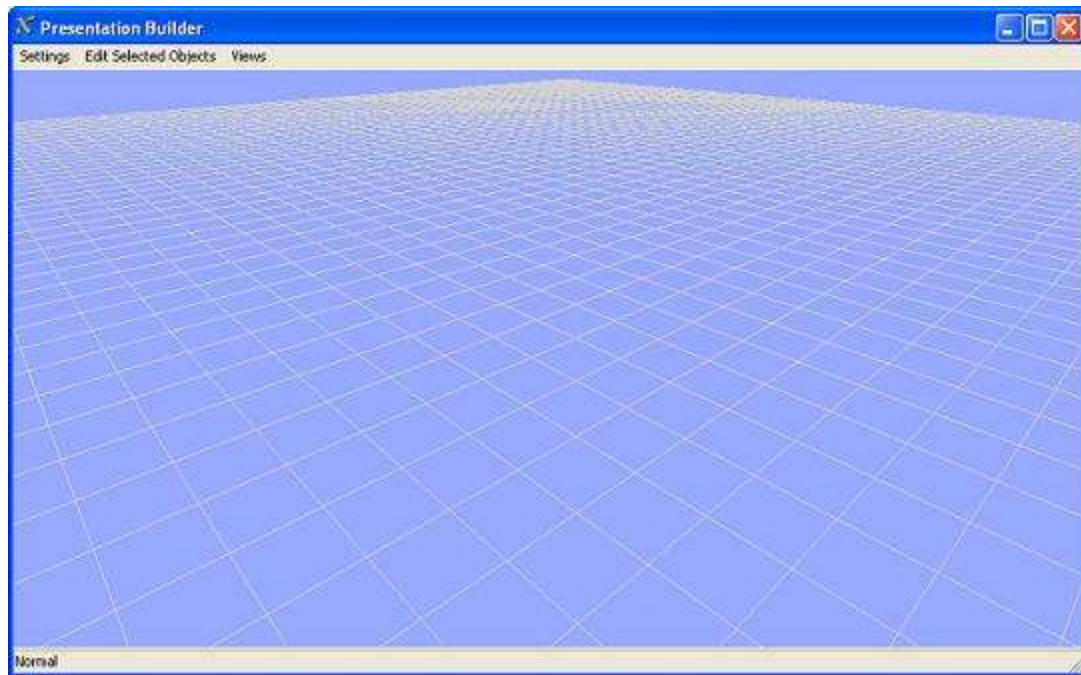
Once you have configured the above parameters, press the Apply button to apply your configuration, then press the Optimize button, and wait until a message appears telling you that the optimization has finished.

**Note:** Make sure your model is compiled before clicking the Optimize button, or else Flexsim may freeze operation.

**Note:** Once you have clicked the Optimize button, don't do anything until the optimization has finished.

## Presentation Builder

The presentation builder is a special perspective view that will assist you in developing a fly-thru presentation of the model. When used with the presentation slide option of the visual tool the presentation builder can develop PowerPoint™ style presentations in 3D. The presentation builder is displayed when you select the Presentation Builder option in the Presentation menu. It is made up of two windows. On the top is a 3D perspective view, which operates just like a regular ortho/perspective view. Below this window is the presentation edit panel. This panel is used to create and edit fly paths.



A Flexsim presentation is simply a series of fly paths. Each fly path is made up of a set of view points, or fly points. Flying through a given fly path means that the perspective view sequentially flies to each fly point in the fly path. A presentation is an ordered series of fly paths.

### Presentation Edit Panel



The presentation edit panel allows you to create, edit and run fly paths. It is made up of several control panels from which you can configure your presentation.

### Paths Panel

The paths panel allows you to edit all fly paths. Click on the New Fly Path button to create a new fly path. The presentation view's current viewpoint will be set as the first point of the fly path. Click on the Delete Fly Path to delete the currently selected fly path. To select a fly path to edit, either click on the Previous or Next buttons, or select the appropriate fly path from the drop-down list.

### Points Panel

This points panel allows you to edit individual view points in a fly path. Click the New Fly Point button to add the presentation view's current viewpoint to the fly path. Click on the Update Point button to change the currently selected point to the presentation view's current viewpoint. Click the Delete Fly Point button to delete the currently selected point. To select a point, either click the Previous or Next buttons, or select the point from the drop-down list.

### Run Control Panel

The run control panel lets you test fly paths that you have created. Click the Run button to repeatedly fly through the currently selected fly path. Click the Step button to step to the next fly point in the fly path. Click the Test button to fly through the currently selected fly path once. Click the Stop button to stop the current fly through. Check the "Run fly paths in simulation time" checkbox to make the time to fly to a fly path dependent on the simulation time instead of real time.

### Sounds Panel

Here you can specify sounds to be played and the beginning and end of a fly path's fly-through. Click the "..." button to browse for a .wav file to play when the fly path is being run. The "Start" file will be played at the very beginning of the fly path run, and the "Finish" file will be played as soon as the fly path finishes.

### Fly Path Data Table

The fly path data table allows you explicitly enter data for the fly points of the current fly path. You will usually only need to edit the Time column, but you can also edit other values

**Time** - This field specifies the amount of time that it will take to fly to the fly point. If the "Run fly paths in simulation time" checkbox is not checked, then the time field is specified in real time seconds. If the box is checked, then you will need to specify the time in thousands of seconds. For example, if you want the view to fly to the position in 2 simulation seconds

**PosX** - This field specifies the x location of focus point of the camera.

**PosY** - This field specifies the y location of focus point of the camera.

**PosZ** - This field specifies the distance the camera is away from its focus point



**RotX** - This field specifies the pitch of the camera.

**RotY** - This field specifies the roll of the camera.

**RotZ** - This field specifies the yaw of the camera.

**Sound** - This is a path to a .wav file that will be played when the camera starts to fly to that point. Only the first and last points in a fly path will play this sound file.

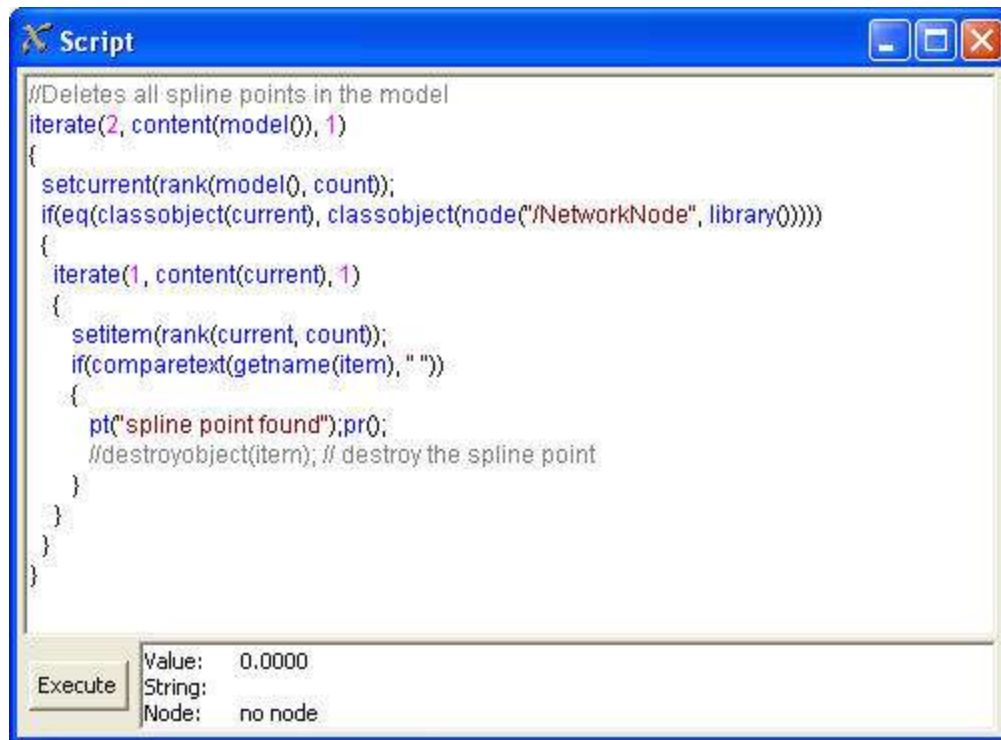
**OnStart** - Here you can write flexscript code to be fired when the fly path starts. The OnStart will only be executed for the first point in the path.

**OnFinish** - Here you can write flexscript code to be fired when the fly path finishes. The OnFinish will only be executed for the last point in the path.

### Navigating Through Fly Paths in the Perspective View

When you are working in the presentation builder perspective view, you can fly through fly paths by pressing keys on the keyboard. Press one of the numbers: 1-9 to run the associated fly path. Press the space bar or the 'N' key, and the view will run the next fly path. Press the 'B' key and the view will go back to the previous fly path.

## Script Editor



The script editor allows you to execute flexscript commands to get information from as well as configure your model. Type the flexscript code in the main text pane at the top. Click the Execute button to execute the code. The return value of the code will be shown in the pane at the bottom.

If your executed script is several lines, you can use the double forward slash '/' to comment certain lines.

**Note:** The return value of the script you execute will only be shown if the last command in the executed script does not have a semi-colon (;) after it.

## Simulation Experiment Control

The experimenter tool can be reached from the Stats menu. This dialog allows you to run experiments by running your model through multiple scenarios, changing certain variables between model runs, and collecting output data from each scenario. Each scenario represents a certain model configuration. The model runs several replications for each scenario.

**Simulation Experiment Control**

Experimenter | Performance Measures | Advanced

**Replications**

☒ Use Experimenter ☐ Save state after each replication (states will be saved in the 'experiment' subdirectory)

Simulation End Time: 3600.000 Replications per Scenario: 5.000 Current Replication: 0.000

Warmup End Time: 0.000 Number of Scenarios: 5.000 Current Scenario: 1.000

**Experiment Variables**

Number of Experiment Variables: 1

	Variable 1	
Path	Not Specified	
Scenario 1	0.000	
Scenario 2	0.000	
Scenario 3	0.000	
Scenario 4	0.000	
Scenario 5	0.000	

Apply OK Cancel

### Replications

These fields determine how many different scenarios to run, how many replications per scenario, and other model run information.

**Use Experimenter** - Checking this box will activate the Experimenter.

**Save state after each replication** - If this box is checked, the model will save its state at the end of every replication. States are saved as .fsp files in the experiment folder. These files can be opened to see the results of each replication using the Load State option from the file menu.

**Simulation End Time** - After the model has run for this amount of time, the model stops, and the next replication or scenario (if there is one) will be run.

**Warmup End Time** - After the model has run for this amount of time, the statistics are reset, but the model is not.

**Replications per Scenario** - This number defines how many replications will be run during each scenario.

**Number of Scenarios** - This number defines how many scenarios will be run. Each scenario may have multiple replications in it. The number of scenarios is defined in the “Number of Scenarios” field. A special event is called at the end of each scenario.

**Current Replication** - This number is the number of the replication that is currently running. The replication numbers start over for each scenario.

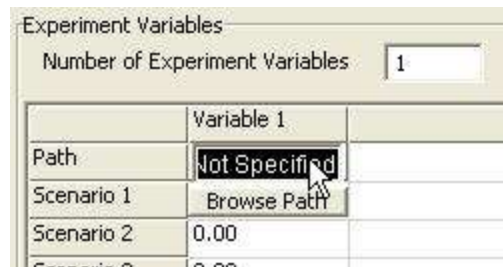
**Current Scenario** - This number is the number of the scenario that is currently running.

## Experiment Variables

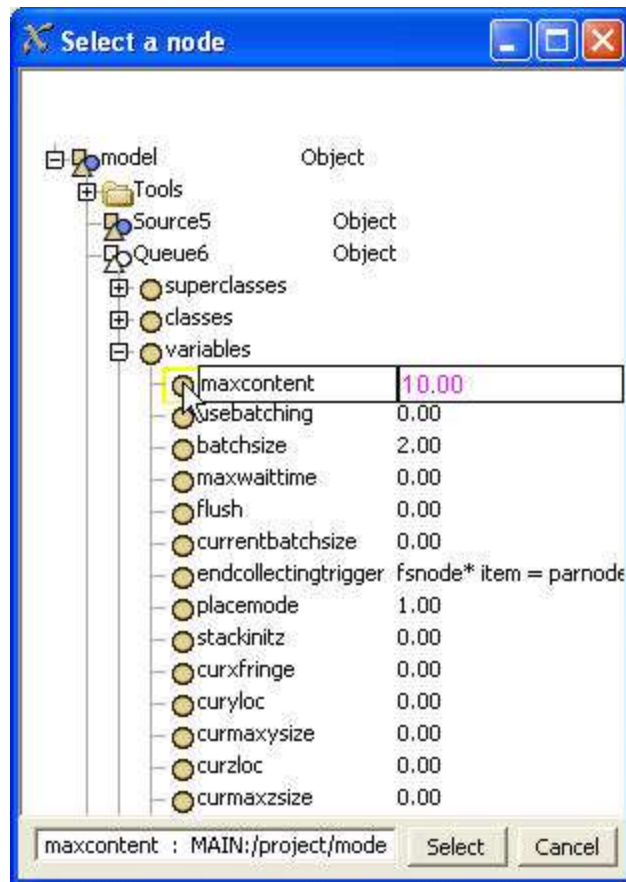
This table allows you to define the configuration for each scenario in the experiment. An experiment variable is some node in the model whose value you want to experiment with. For example it may be the maximum content value of a queue, or it may be a value in a table. You may define several experiment variables for your experiment. Each experiment variable is associated with a column in the table. Enter the number of experiment variables you would like to use, then press Apply, and the appropriate number of columns will be created.

### Specifying the experiment variables

To associate an experiment variable with its appropriate node, click on the Path row of the table.



Click the Browse Path button. This will open a tree browse window, where you can explore the tree for the variable.



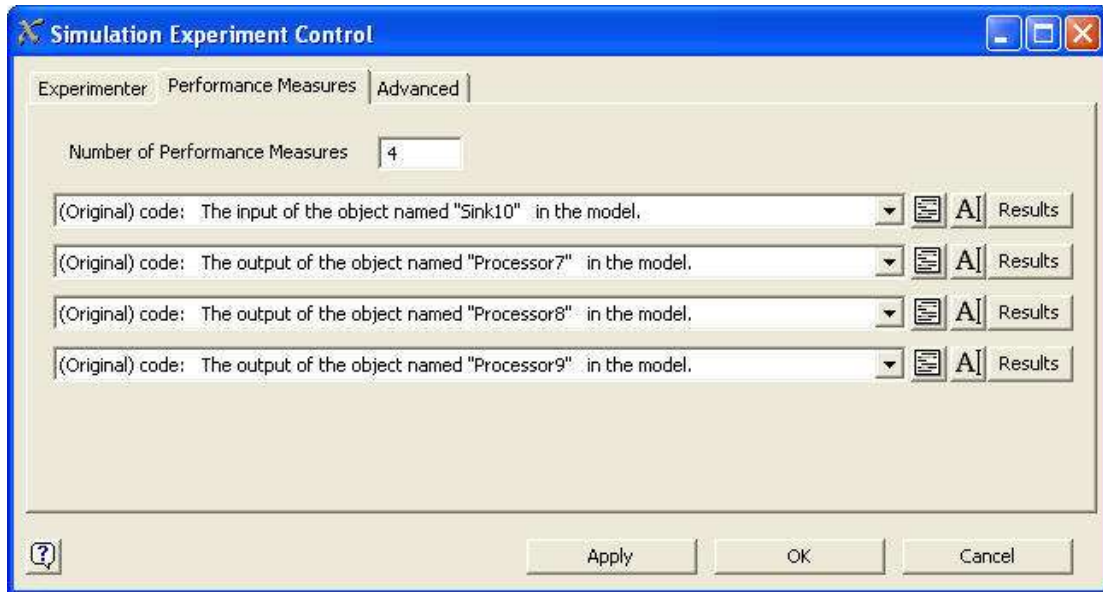
In the above example, the maxcontent variable of a Queue is selected as the experiment variable.

**Note on experiment variable data:** Make sure that the node you select for an experiment variable has number data on it. Otherwise the experiment will not work properly. If you want to experiment with variables that do not have number data on them, then you will need to somehow translate a number value into whatever it is you want to experiment with. For example, if you want to experiment with different types of sendto strategies, you can store a number label on an object or a table, and in the sendto strategy, query the label and execute different strategies dependent on the value of the label.

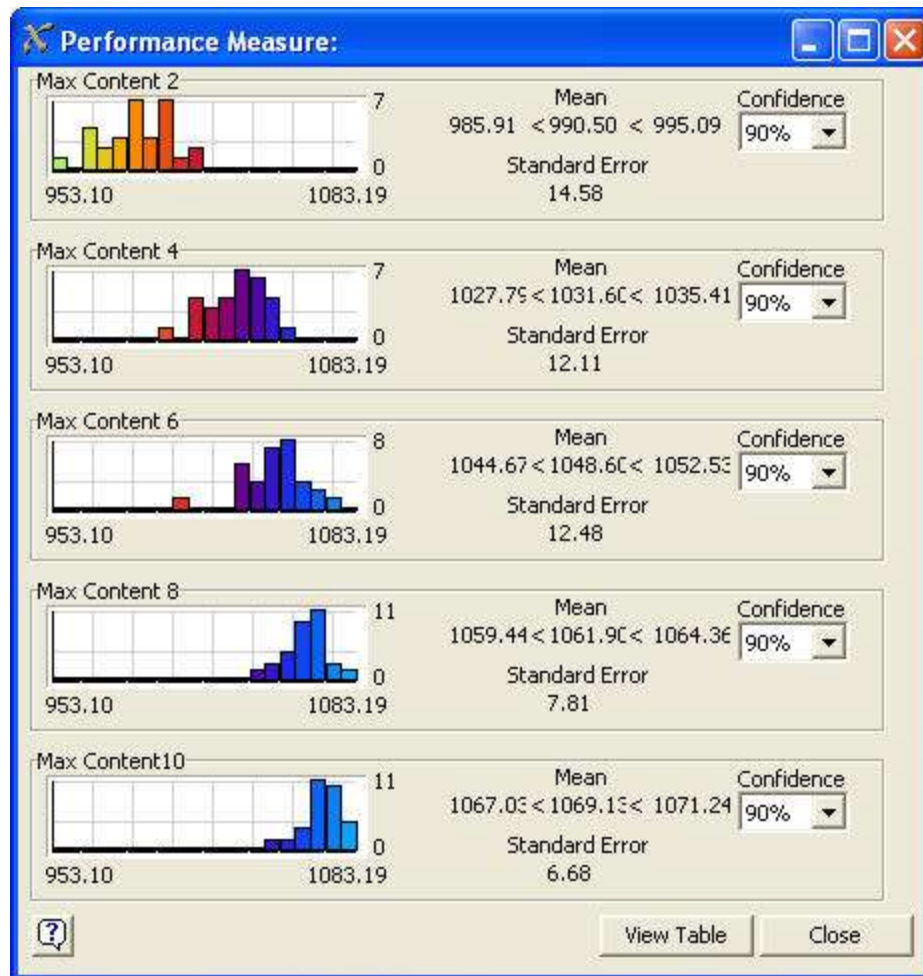
Once you have selected an experiment variable, fill in the value that you want the variable to be set to for each scenario. Each scenario is associated with one row of the table.

### Performance Measures

The performance measures tab page lets you specify the output measures to evaluate for each scenario.



You can define up to 10 performance measures. Enter the number of performance measures and press the Apply button to create the correct number of performance measures. Each performance measure is defined using a pick list. See the performance measure pick list. At the end of each replication the performance measure functions are executed and recorded. Once the whole experiment has finished, you can click on the Results button to bring up a window showing the results of a given performance measure.



The above performance measure results show the input of a Sink for scenarios where the maximum content of a queue was set to 2, 4, 6, 8, and 10. Notice that the maximum content of 10 naturally resulted in the highest throughput. To view the table of all values, press the View Table button.

### Advanced

The fields in the advanced tab page define behavior at various points in the replications.

**Start of experiment** - This pick list is called when the experiment begins running. It is used to set up certain variable values before any scenarios begin to run. This function is only called once, before the first scenario runs. See the start of experiment pick list.

**Start of scenario** - This pick list is called before the first replication in a scenario. It is called once for each scenario. See the start of scenario pick list.

**Start of replication** - This pick list is called when the start time of a replication. It is run once for each replication. See the start of run pick list.

**End of warmup period** - This pick list is called when the warm-up time of the replication has expired. See the end of warm-up pick list.

**End of replication** - This pick list is called when the end time of a replication has expired. It is run once for each replication. See the end of replication pick list.

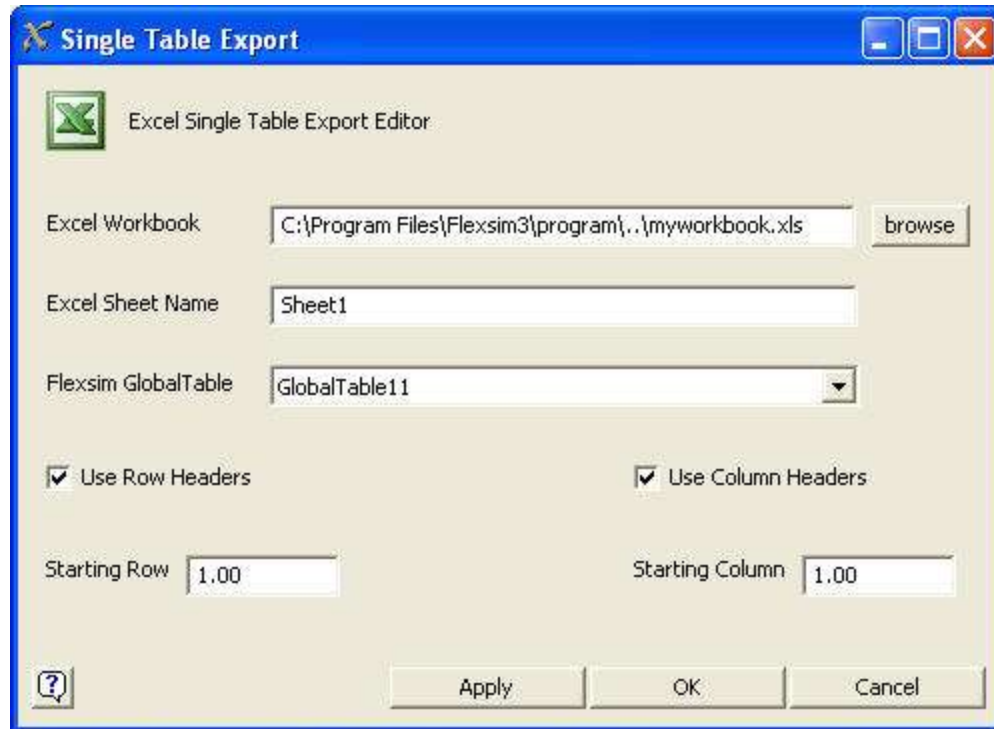
**End of scenario** - This pick list is called at the end of each scenario. It is called once for each scenario. See the end of scenario pick list.

**End of experiment** - This pick list is called when the experiment is over. It is used to write model data at the end of the run. This function is only called once, after the last scenario runs. See the end of experiment pick list.



## Single Table Export

The single table export tool lets you export tables from Flexsim to Microsoft Excel. Once you have configured this editor, click on the Single Table Export button in the Excel Interface window to export the table.



**Excel Workbook** - Here you specify the Excel workbook that you want to export the table to. Click on the browse button to browse for the workbook.

**Excel Sheet** - This field defines the name of the worksheet to export to.

**Flexsim Global Table** - This field defines the global table from which to export the data. Select the table from the drop-down list.

**Use Row Headers** - If this box is checked, then the global table's row headers will be the first row exported.

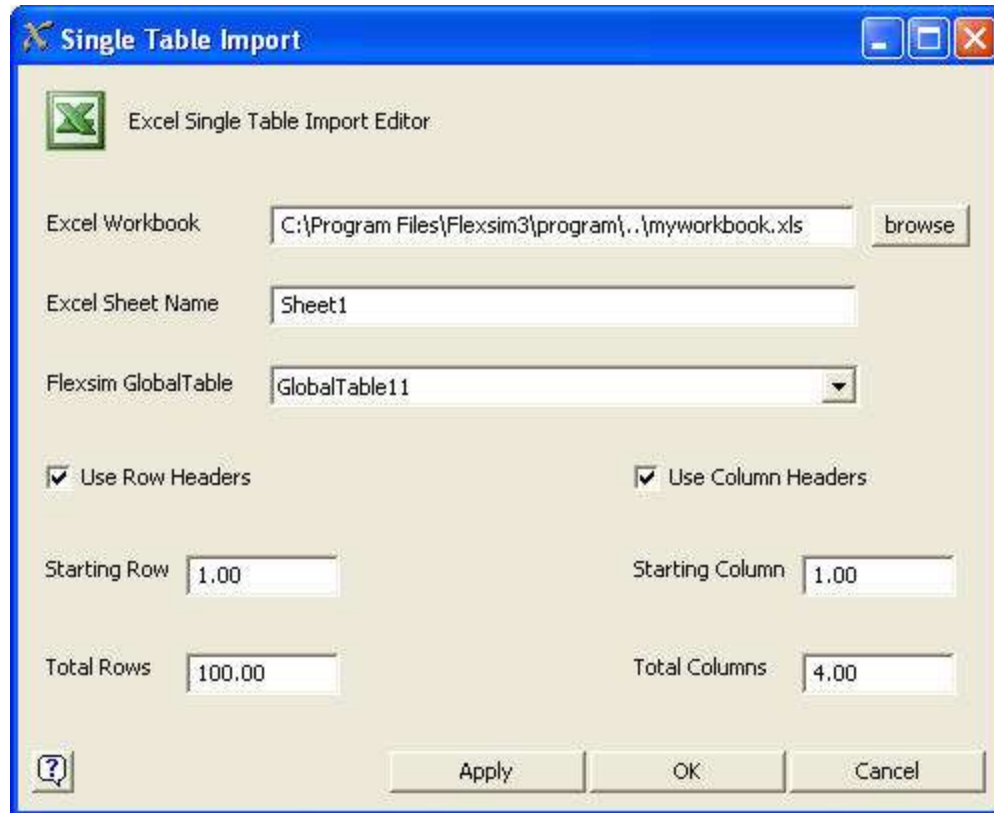
**Starting Row** - This field defines the starting row number of the Excel spreadsheet.

**Use Column Headers** - If this box is checked, then the global table's column headers will be the first column exported.

**Starting Column** - This field defines the starting column number of the Excel spreadsheet.

## Single Table Import

The single table import tool lets you import tables from Microsoft Excel to Flexsim. Once you have configured this editor, click on the Single Table Import button in the Excel Interface window to import the table.



**Excel Workbook** - Here you specify the Excel workbook that you want to import from. Click on the browse button to browse for the workbook.

**Excel Sheet** - This field defines the name of the worksheet to import from.

**Flexsim Global Table** - This field defines the global table to which the data will be imported. Select the table from the drop-down list.

**Use Row Headers** - If this box is checked, then the starting row of the excel spreadsheet will be imported as the global table's row headers.

**Starting Row** - This field defines the first import row number of the Excel spreadsheet.

**Use Column Headers** - If this box is checked, then the starting column of the excel spreadsheet will be imported as the global table's column headers.

**Starting Column** - This field defines the first import column number of the Excel spreadsheet.

## Sky Box Editor

The Sky Box is a background created by Selecting the “Sky Box” option in the “Presentation” menu. It is a box that surrounds the model and has a different picture on each side of it. The size of the box and what pictures are displayed can be defined by the user.

**Note:** The Sky Box will only be correctly shown in a perspective view, not in an orthographic view



**Size** - This is the size of the background box. The box is always centered on the point (0,0,0) in the model. It should be made large enough to surround the entire model.

**Delete Background** - When this button is pressed, the background object is deleted from the model.

**Front** - This file is the image that will be on the front of the background box.

**Back** - This file is the image that will be on the top of the background box.

**Left** - This file is the image that will be on the left of the background box.

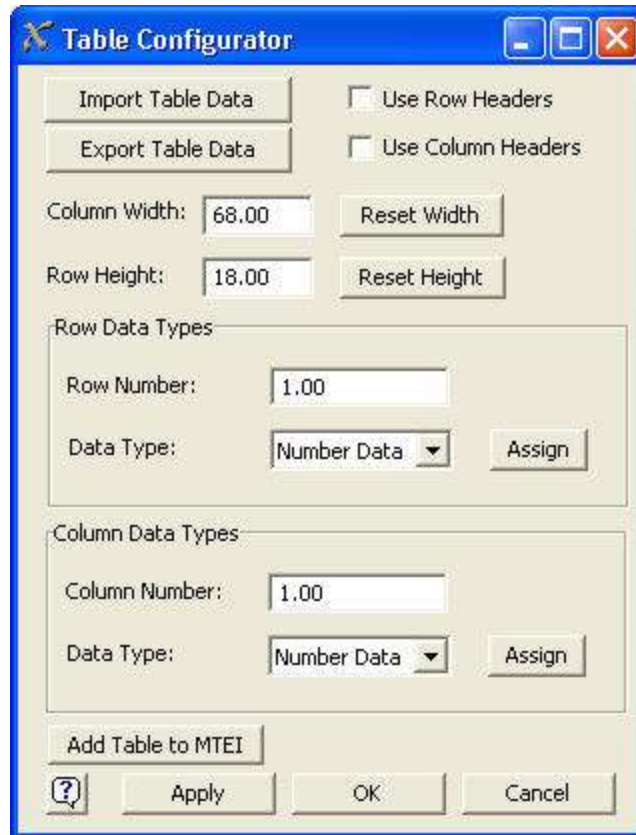
**Right** - This file is the image that will be on the right of the background box.

**Top** - This file is the image that will be on the top of the background box.

**Bottom** - This file is the image that will be on the bottom of the background box.

## Table Configurator

The table configurator lets you configure a table, import data, export data, and assign data types for given rows or columns.



**Import Table Data** - This brings up a file browser to select a .csv file to import into the table.

**Export Table Data** - This brings up a file browser to select a .csv file to export the table to.

**Use Row Headers** - This check box applies only when importing the table data from a .csv file. If it is checked, then the first column of the csv file will be imported as row headers of the table.

**Use Column Headers** - This check box applies only when importing the table data from a .csv file. If it is checked, then the first row of the csv file will be imported as column headers of the table.

**Column Width** - This specifies the default column width of the table. Note that you can specify the width of each column by dragging the column's edge to the right or left. To reset all column widths to the default column width, press the Reset Width button.

**Row Height** - This specifies the default row height of the table. To reset all row heights to the default row height, press the Reset Height button.

**Row Data Types** - Here you can assign a data type to any row in the table. Type in a row number, select a data type from the drop down list, and click on the Assign button.

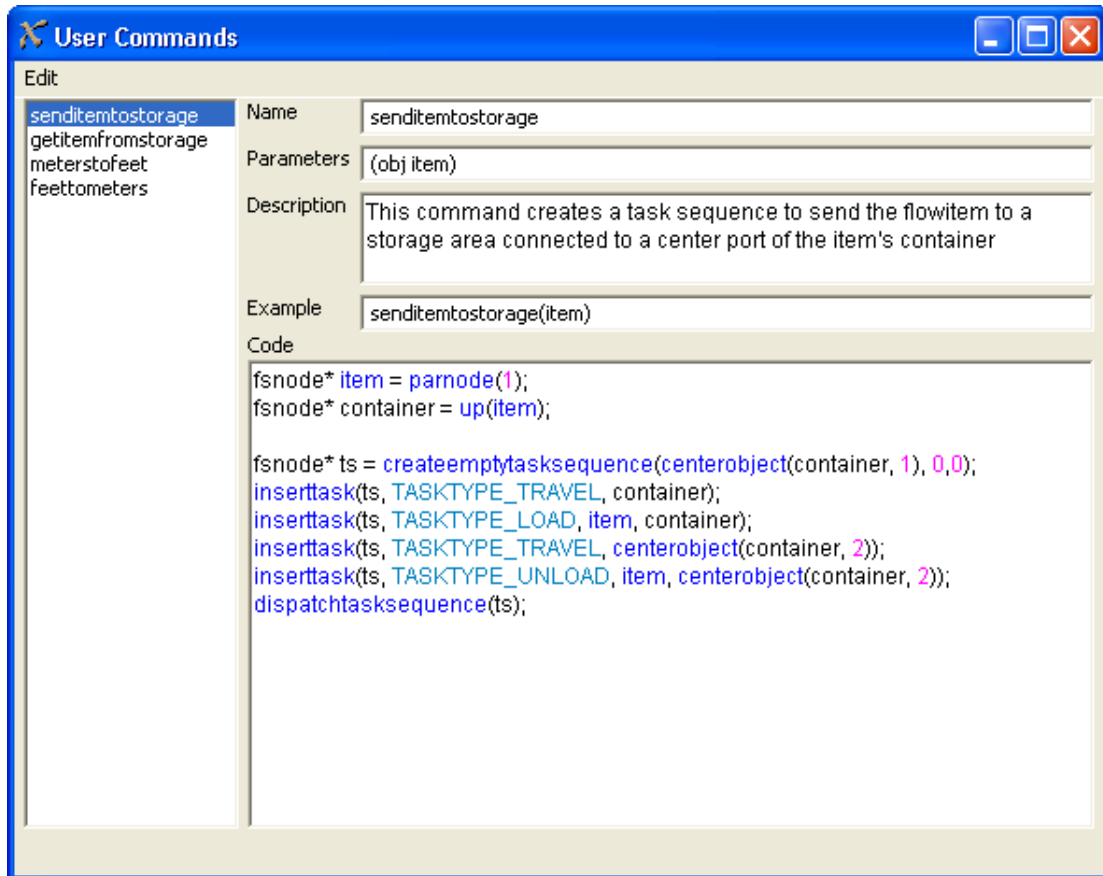
**Column Data Types** - Here you can assign a data type to any column in the table. Type in a column number, select a data type from the drop down list, and click on the Assign button.

**Add Table to MTEI** - This will add the table to the Multiple Table Excel Import.

**Add to User Library** - This group box allows you to add the global table to a user library. You can either add it as a draggable icon or as a component for automatic install. For more information, refer to the user library documentation. Note that this option is not available if you are editing a label table.

## User Commands

The user commands tool lets you add, delete, and edit custom commands in your model.



To add a command, go to the Edit menu, and select New Command. This will create a new command. In the list on the left, this new command should be selected. To edit a command, select it from the list on the left, then enter the name of the command, the parameter list, a description, and an example. Then in the bottom pane, enter the code implementation for the command. To access the parameters passed into the command, use the commands `parval()` and `parnode()`. For example, if the first parameter is supposed to be a reference to an object, execute `parnode(1)` to get a reference to that parameter. If it is a number value, call `parval(1)` to get the value. Once you have created and edited your command, you can call that command like any other Flexsim command. It will appear in blue when typing code and will appear in the command summary.

**Note on calling user commands:** When calling a user command, all parameters passed will need to be converted to a number, or else you may get compiler errors. To do this, use the `tonum()` command. For example, the above figure shows a command called `senditemtostorage()`, which takes one object parameter. To call this command, you will need to cast the object reference to a number parameter: `senditemtostorage(tonum(item));`

**Note on applying changes to a command:** Make sure you've applied any changes to your command before compiling. To apply changes, click on the command again in the list on the left of the window, or just close the window.

## Edit Menu

The edit menu lets you do several operations on your commands.



**New Command** - Select this option to create a new command.

**Delete Command** - Select this option to delete the currently selected command.

**Add Command to User Library** - Select this option to add the user command to the currently active user library. You can either add it as a draggable icon, or as a component for automatic install. For more information, refer to the user library documentation.



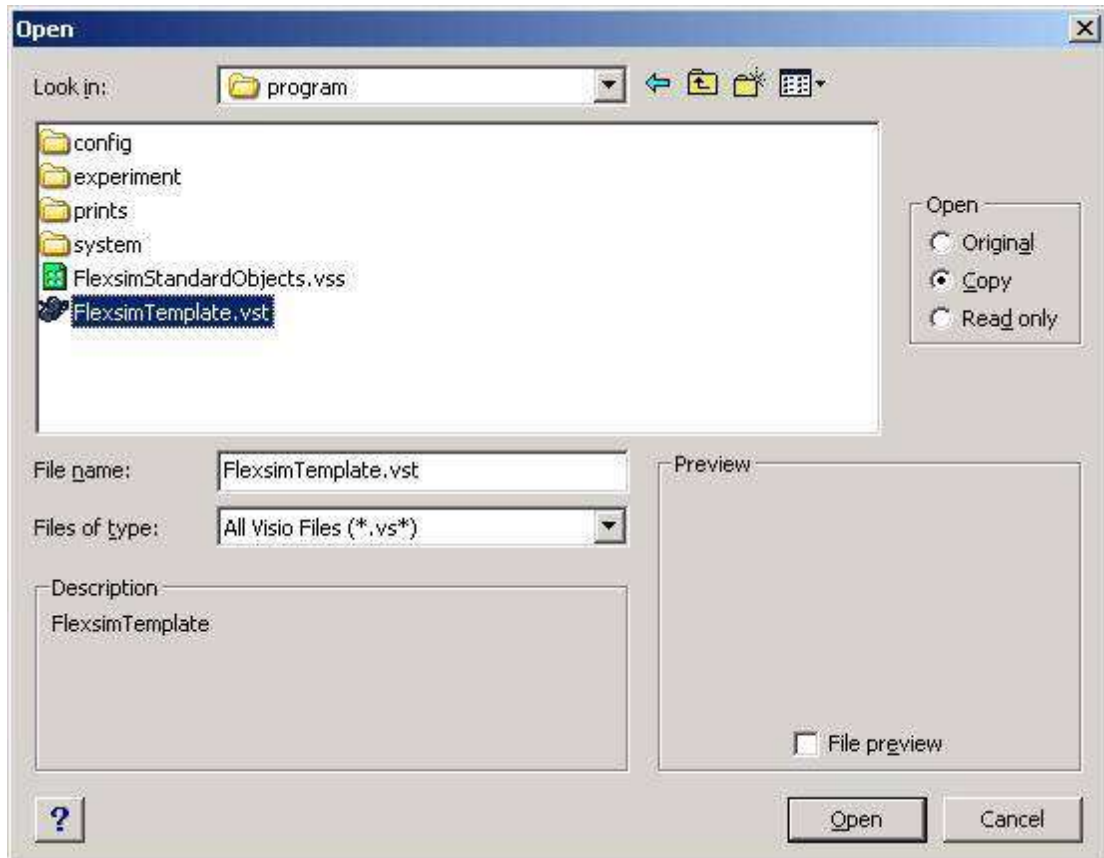
## Microsoft Visio™ Import

The visio import tool allows you to build models in Microsoft Visio™ and import them into Flexsim. Here's how you do it.

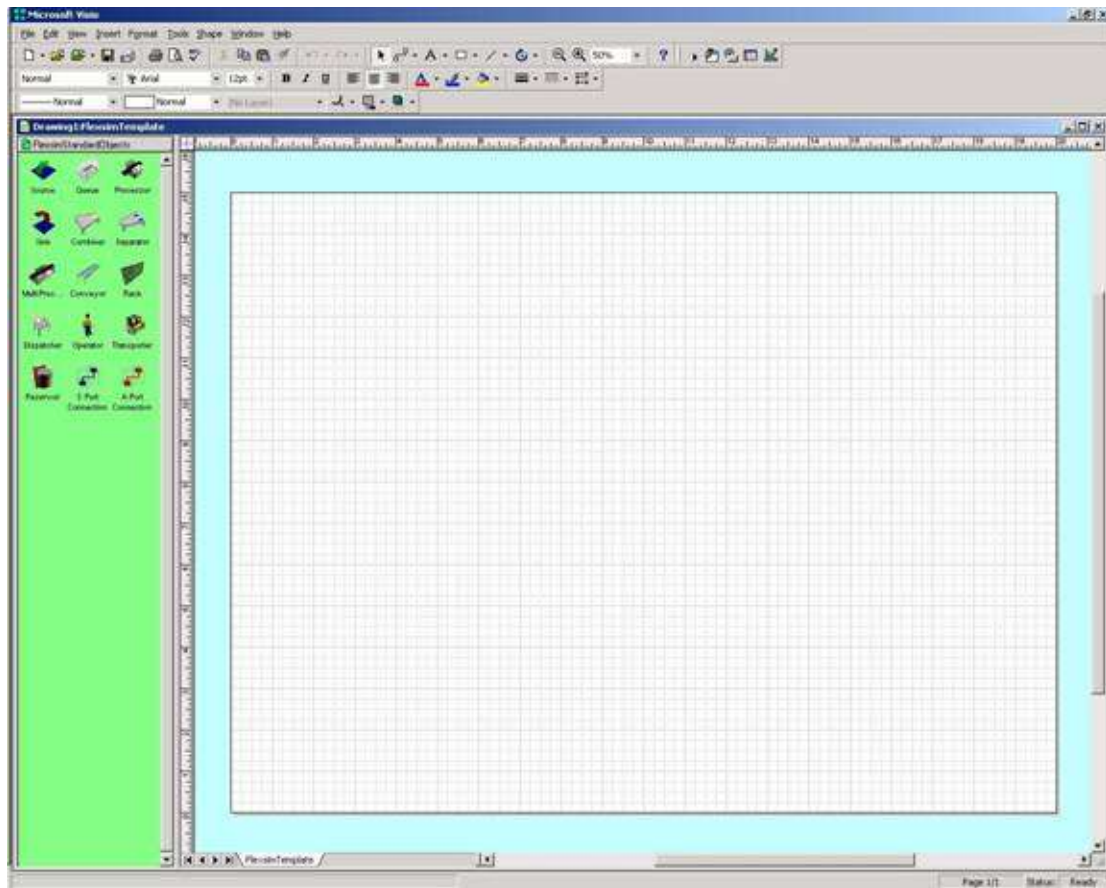
Start Visio



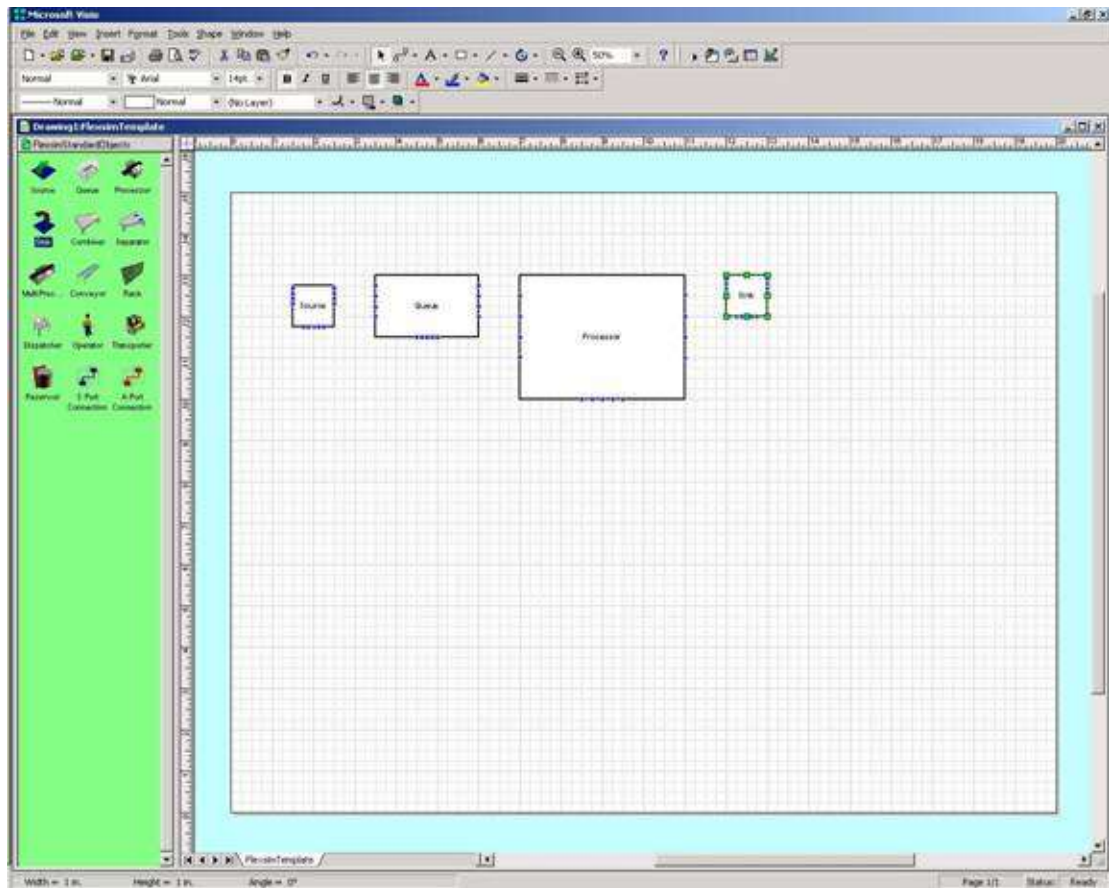
Open the FlexsimTemplate.vst



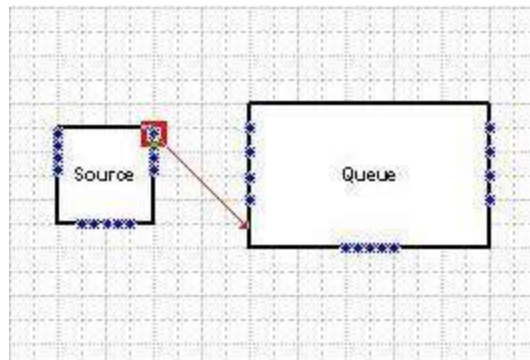
Visio will look like this.



Drag objects from the Stencil into the model space to arrange their layout.

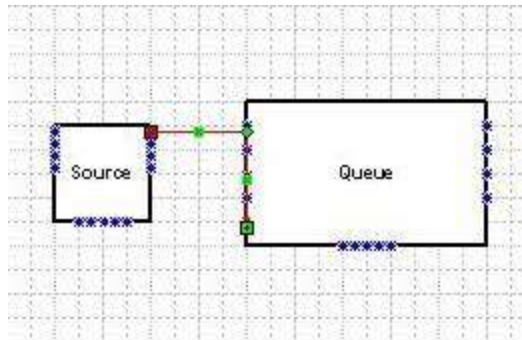


Drag the A Port Connection object into the model to create a connection.

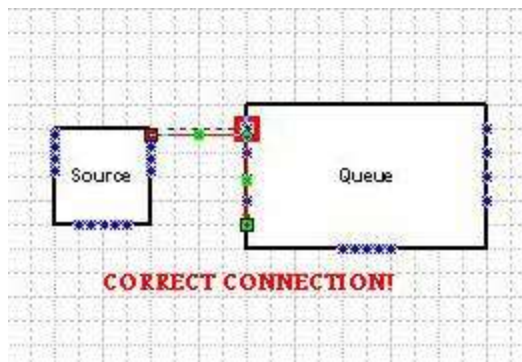
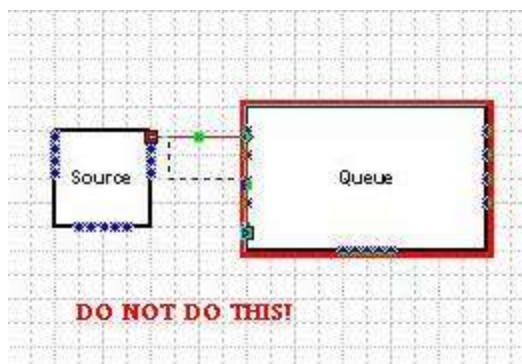


The connection is made from the upper left point to the bottom right point. If you hook the connection backwards, the connection will be backwards in Flexsim.

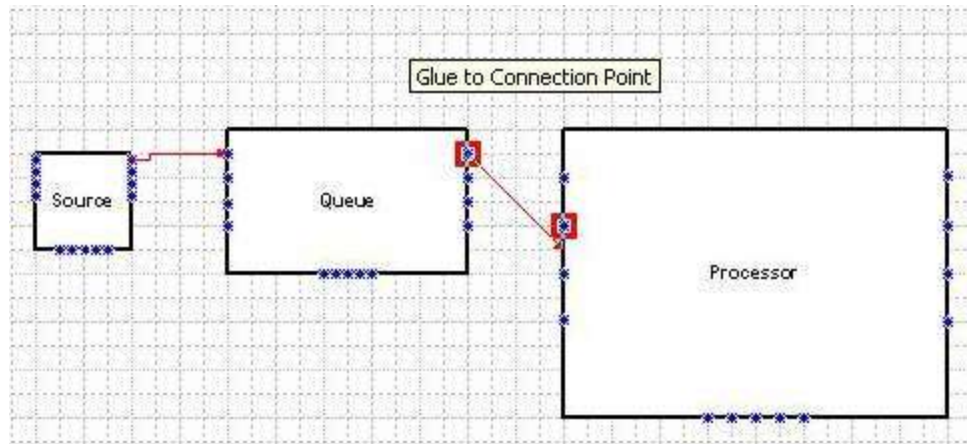
This is a connection that is only connected to the upstream object.



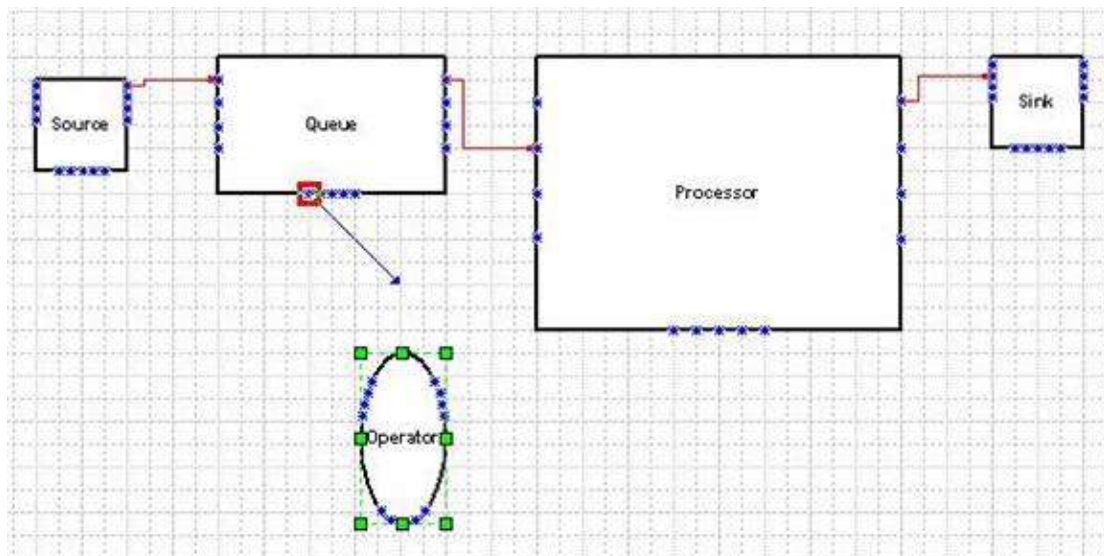
**Note:** Do NOT connect the connection directly to the object. It must be connected to one of the blue points.



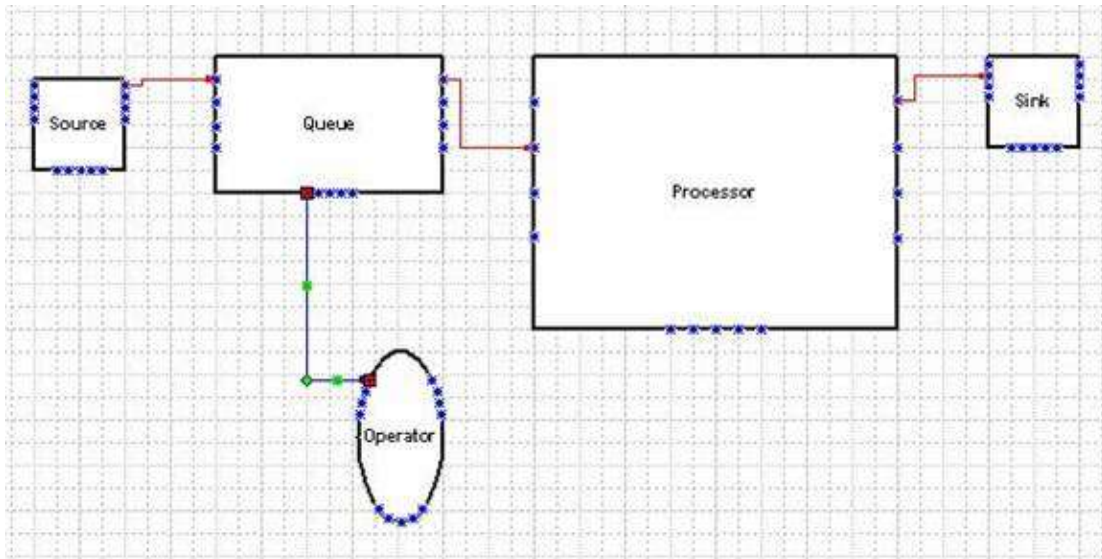
When dragging out a connection, it is acceptable to have both ends touching blue connection points. The connection will be created correctly. Also, the connection point that you connect them to in Visio does not matter. Flexsim connections will be ordered according to the order in which they were dragged out in Visio. The example below will connect Queue to Processor's first input port despite being connected to the second connection point on the object in Visio.



S port connections (center ports) are created in the same way as A port connections. The connection point in Visio that they are connected to does not make a difference in their order. Connecting S port connections to the blue connection points on the left and right is acceptable.



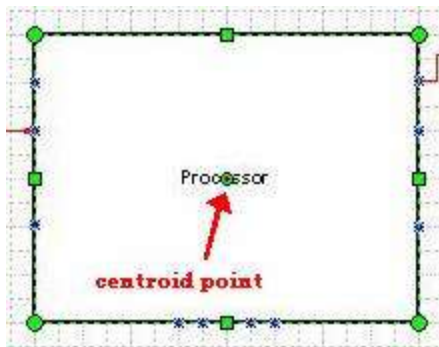





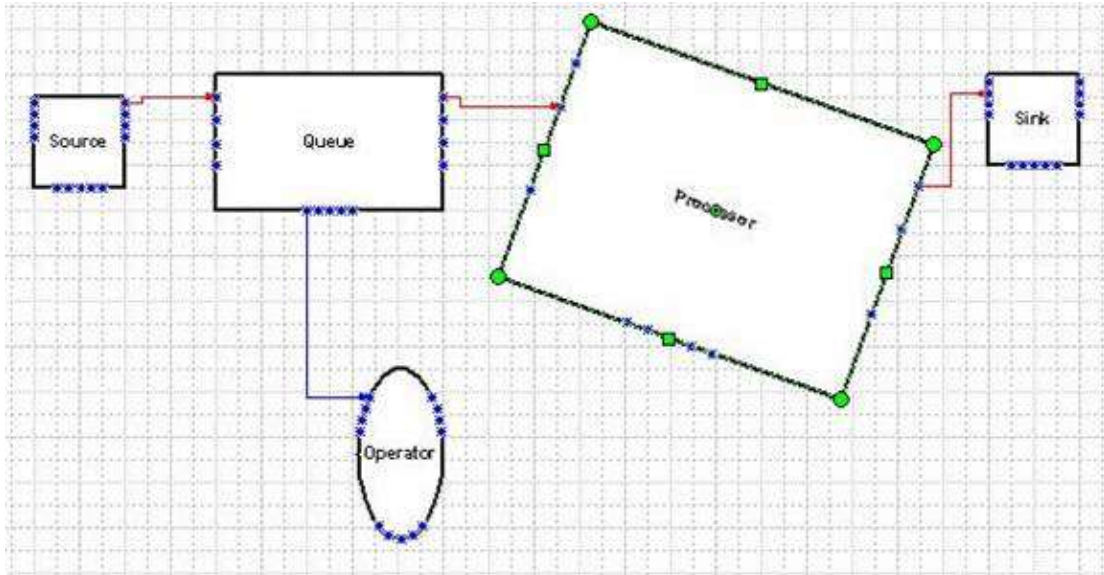
Objects can be rotated and resized from Visio to resize and rotate objects in Flexsim.


The rotation tool in Visio looks like this. 

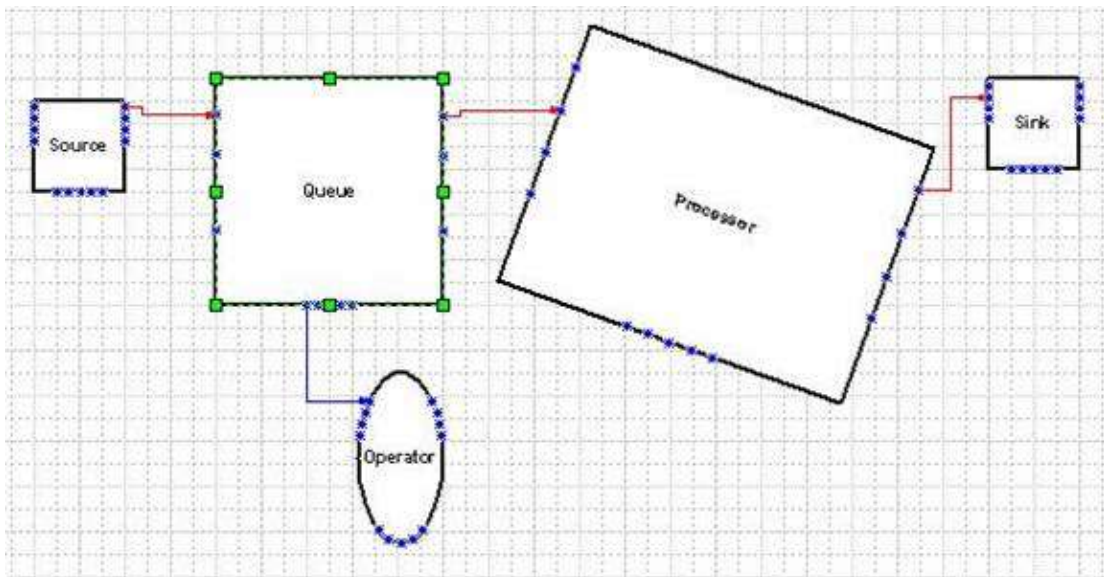
Do not move the centroid point of the object when resizing. The centroid point is the green circle with a black point in its middle. It is used to define where the center of rotation is for that object. If you move it in Visio, the object will not be located correctly in Flexsim.



This processor was rotated by clicking and dragging the rotation tool  around one of its corner points.

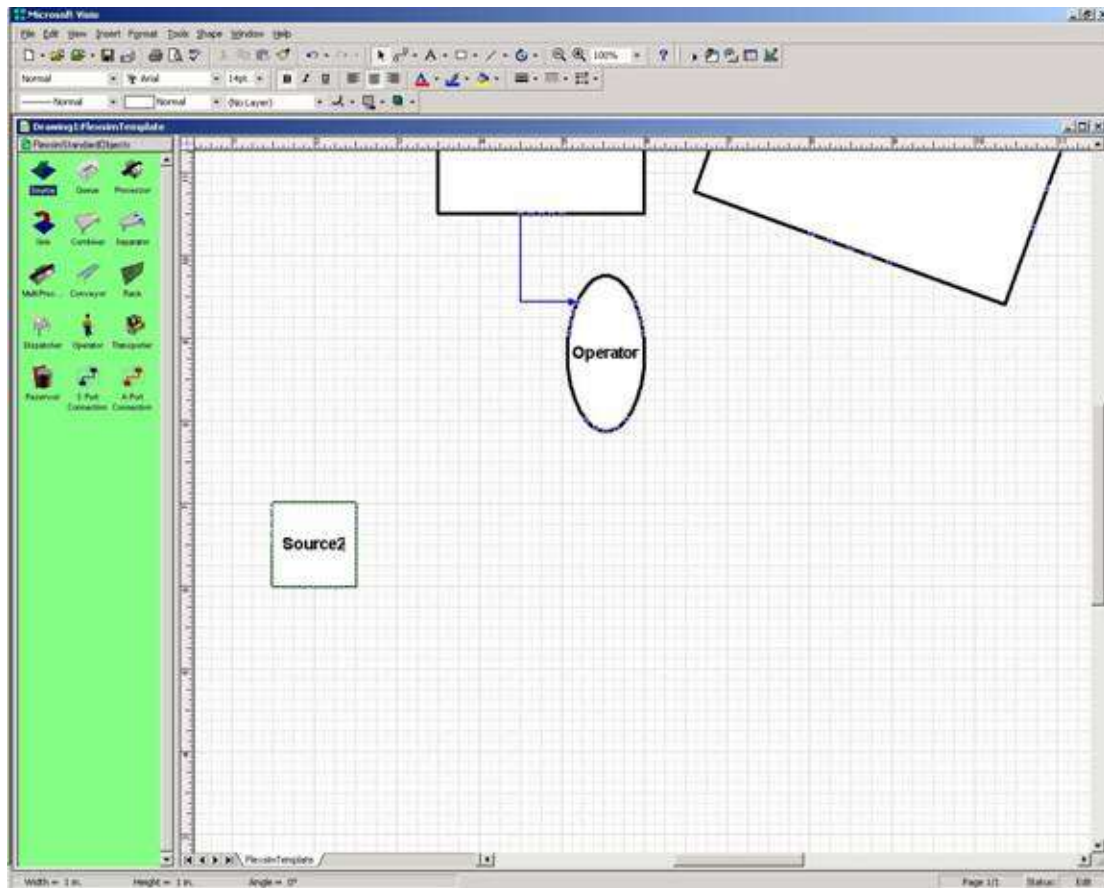


This queue has been resized by dragging its side point down using the selection tool .

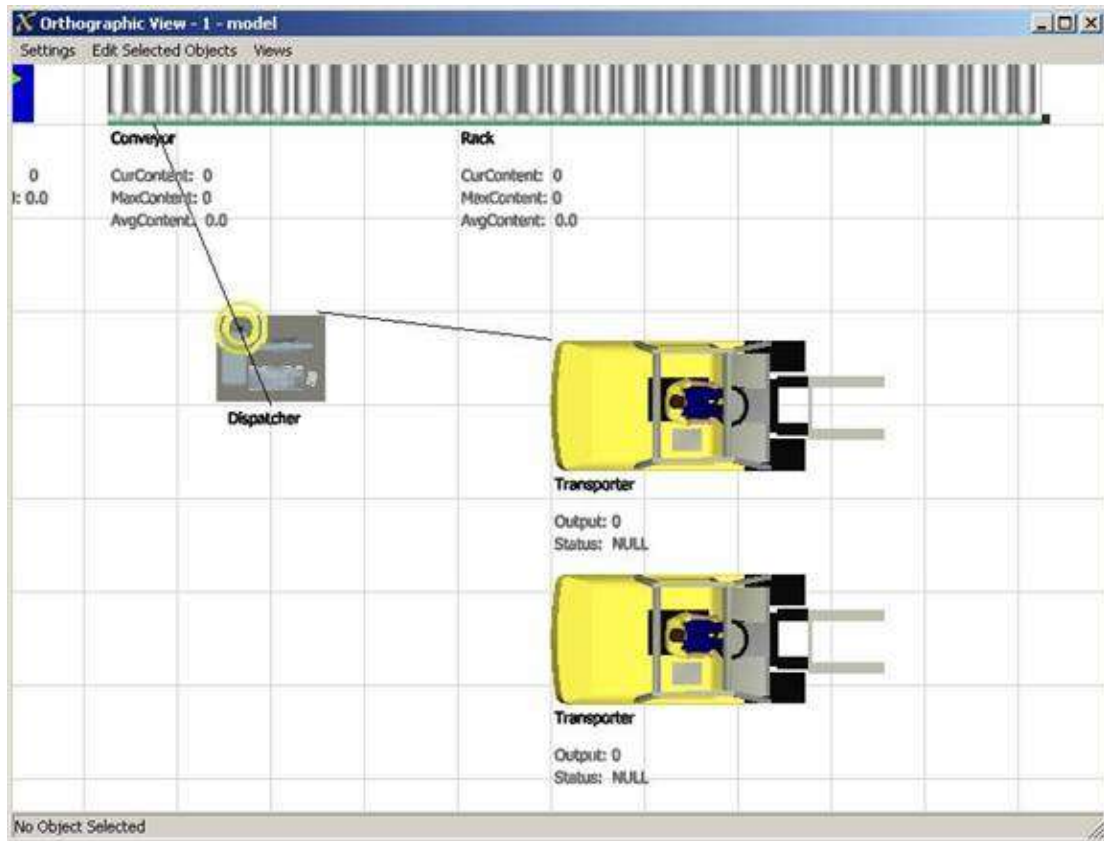


Objects can be renamed by double clicking on the object and typing in a new name.

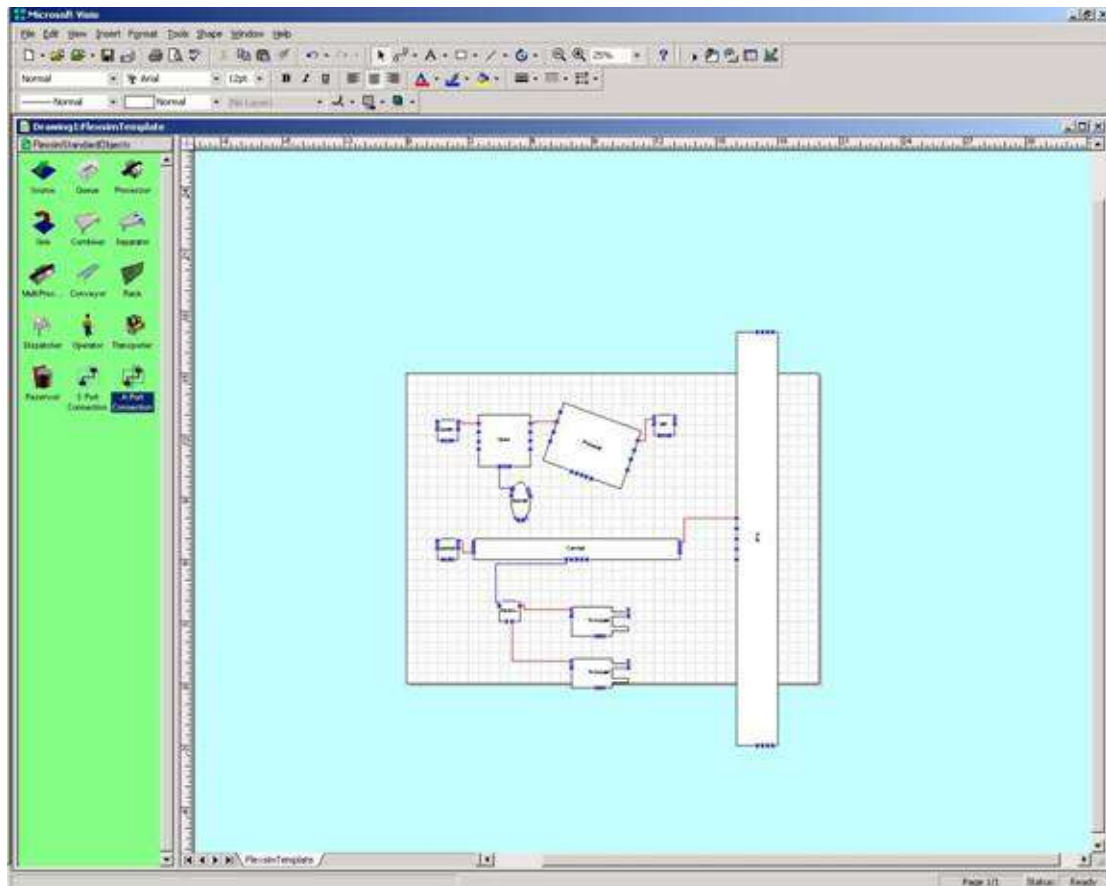




Each object in your model must have a unique name in order for it to be correctly imported into Flexsim. If objects are given the same name in Visio, their connections will not be created correctly in Flexsim. In the example below, the Transporters have the same name, so the Dispatcher was connected to the first one twice.

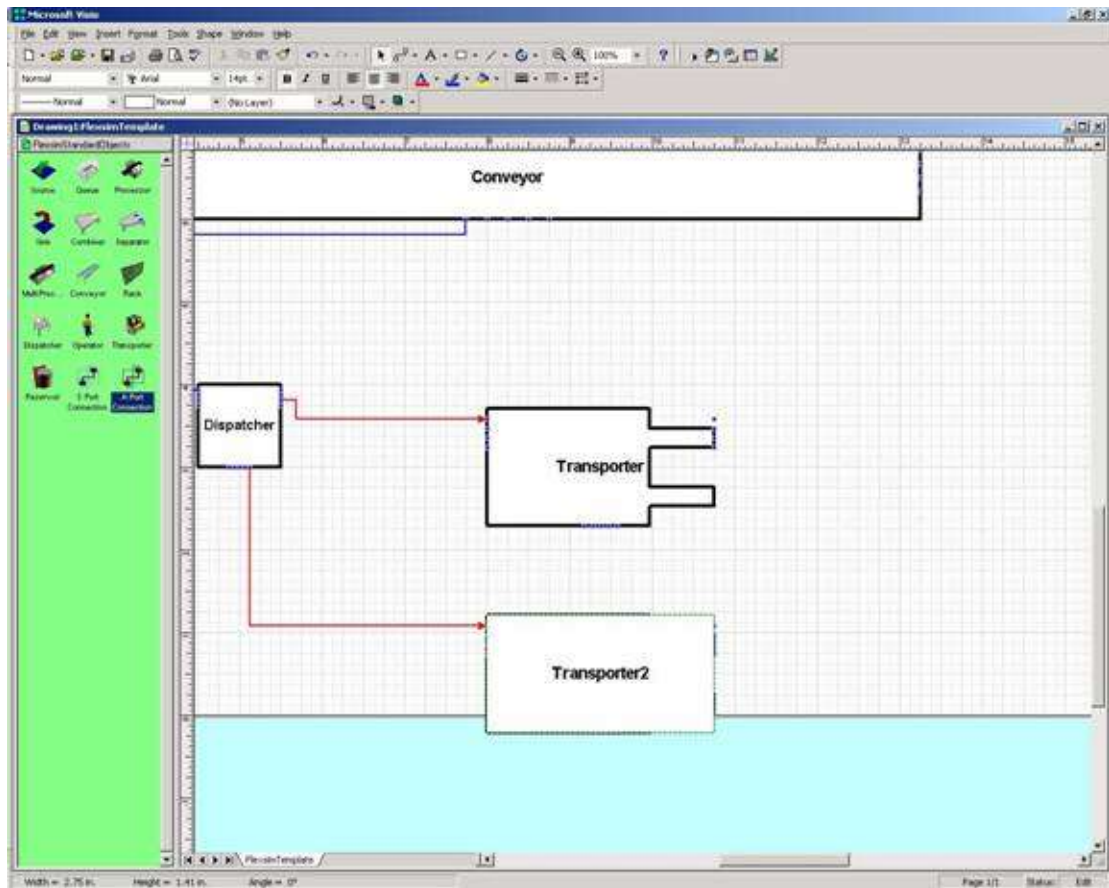


Here is a nearly completed Visio page with a Flexsim layout.

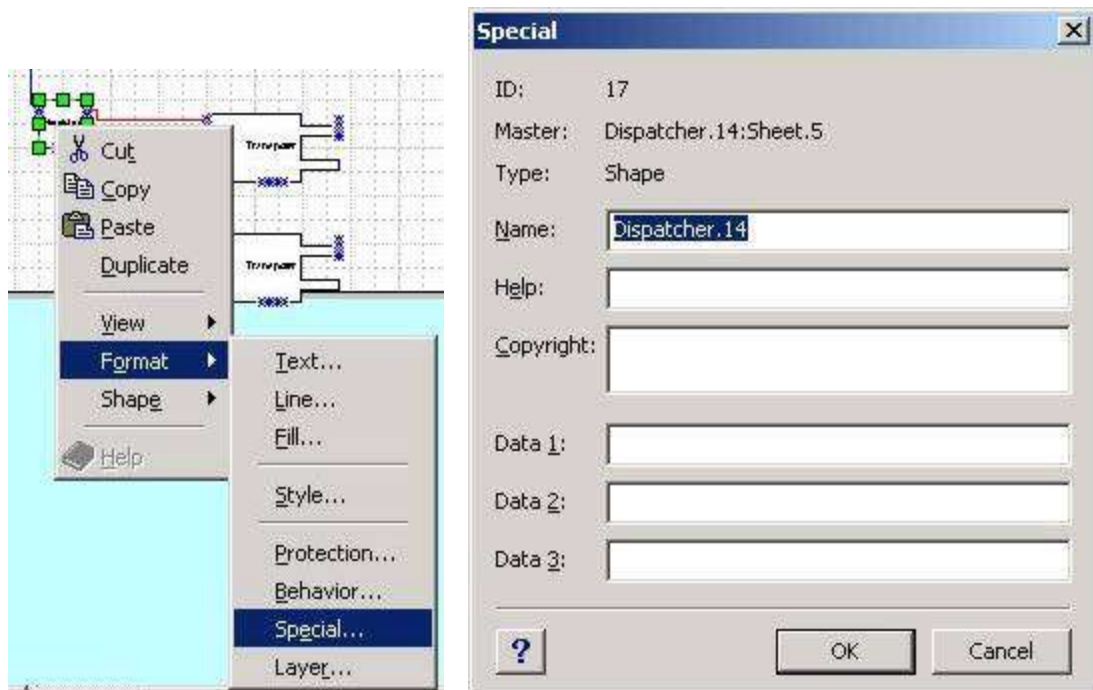


There are a few more things to check to be sure are correct before exporting it to Excel to import into Flexsim.

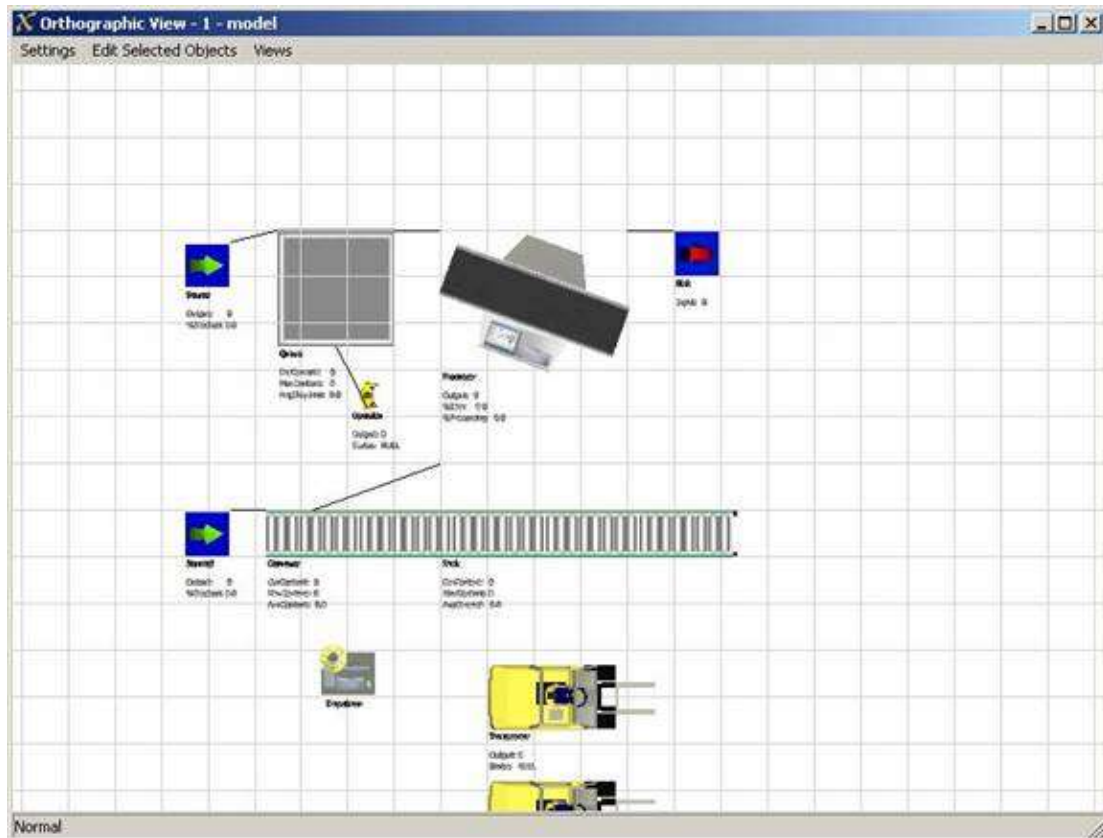
Be sure each object has a unique name.



and each object must be correctly formatted in this menu.

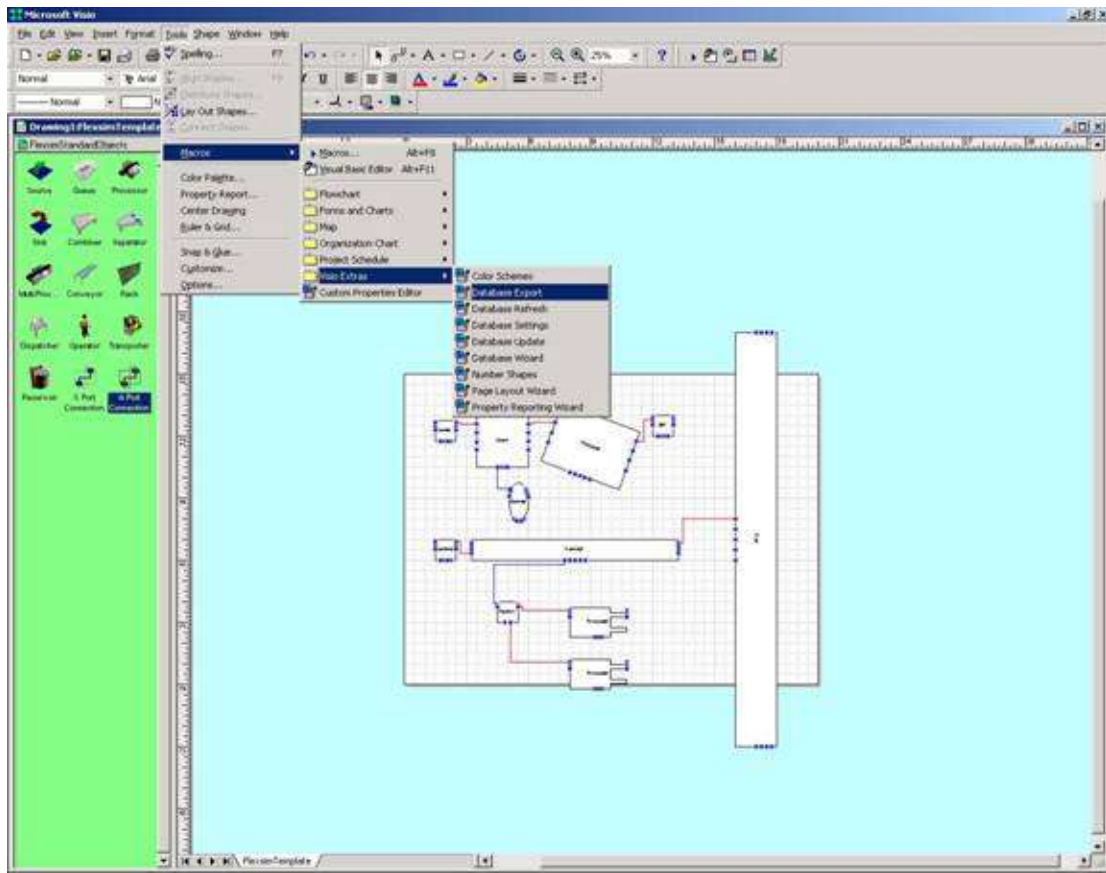


The Name field contains the object's class and a number separated by a period or just the object's class. If there is a number, it must match the ID number or the object will not be correctly exported. Typically the objects do this naturally, but sometimes Visio will incorrectly name an object in this field. If your model didn't correctly import into Flexsim, this was probably the reason why. The example below shows what happens if the name is left as "Dispatcher.14" instead of changed to "Dispatcher.17".

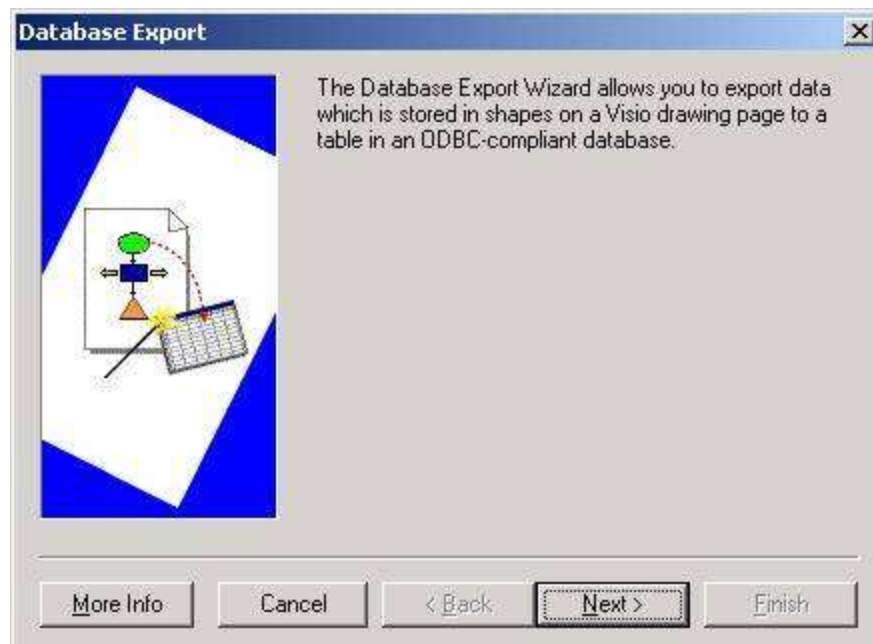


Now that everything has a unique name, and is correctly named in the Special Format, we are ready to export.

Go to the Tools > Macros > Visio Extras > Database Export to export the file.

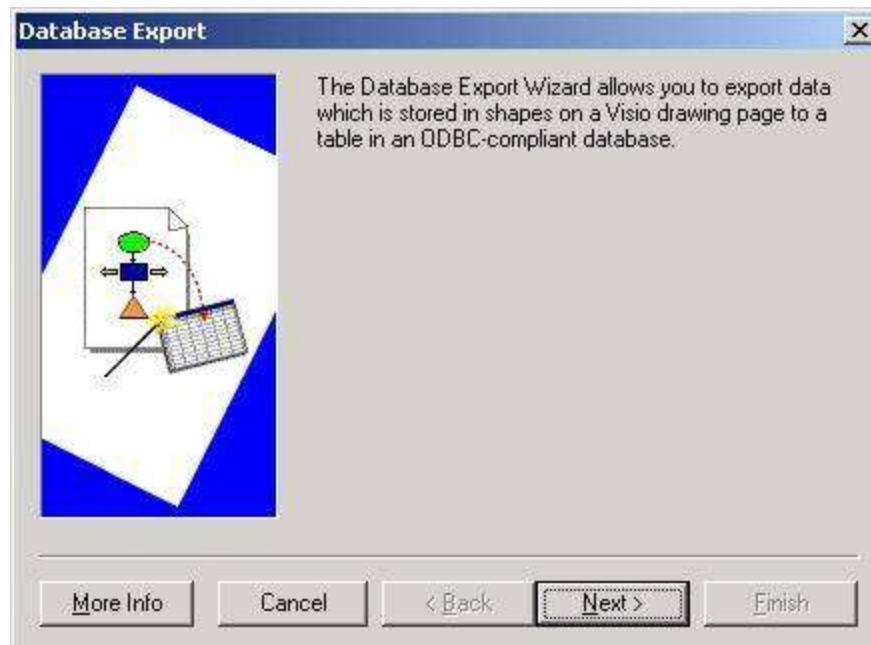


This screen will appear. Click next.

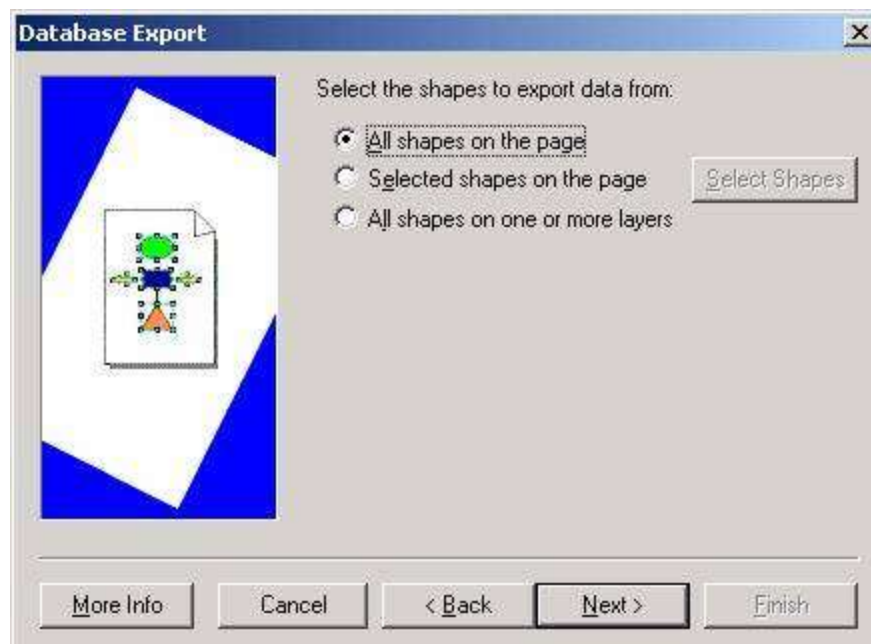


Next.





Select "All shapes on the page" and click Next.



Click the "Add All >>" button to get all the correct information exported.

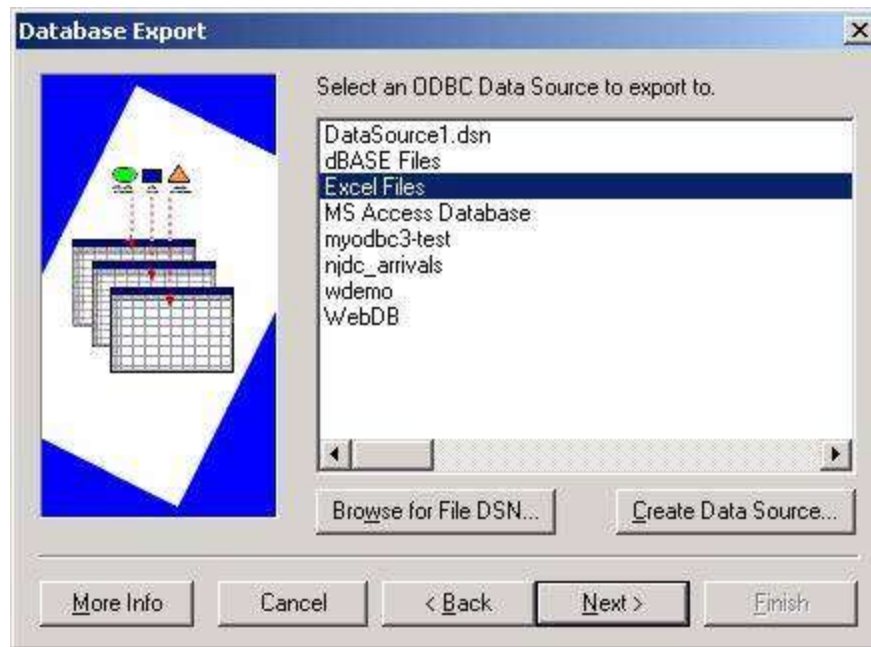


Then click Next.



Select Excel Files and click Next.

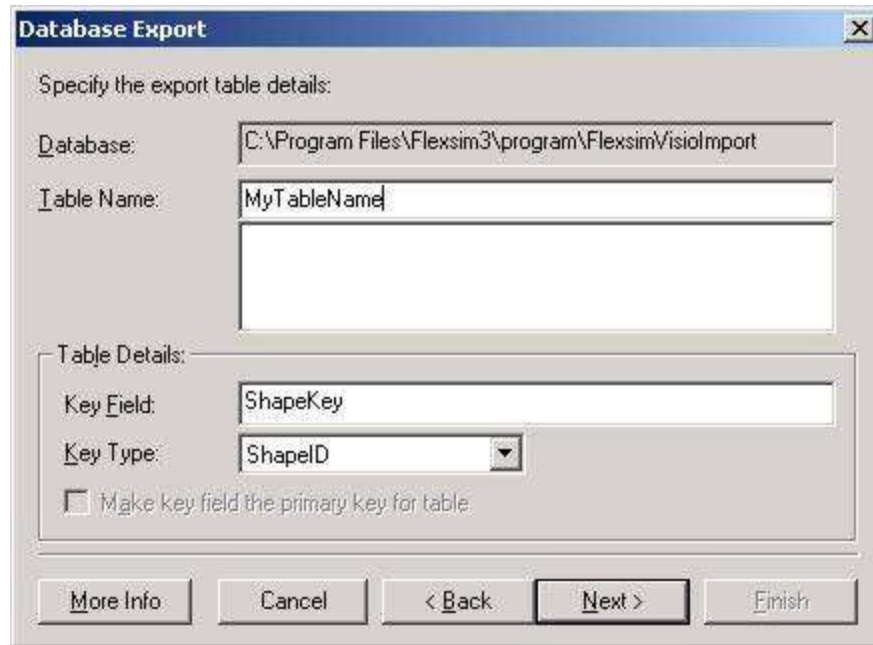




Select the excel file to export to. This should usually be C:/Program Files/Flexsim3/program/FlexsimVisiolmport.xls.



Type a name into Table Name. This name will be typed again into Flexsim so that it knows which sheet to import from. Remember the name you typed and click Next.



**Database Export**

Specify the export table details:

Database: C:\Program Files\Flexsim3\program\Flexsim\visioimport

Table Name: MyTableName

Table Details:

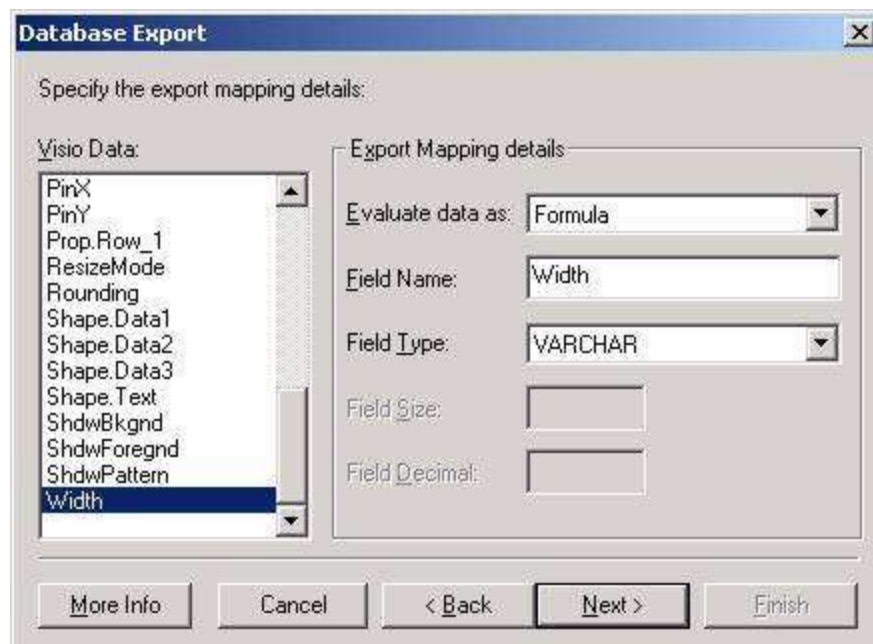
Key Field: ShapeKey

Key Type: ShapeID

☐ Make key field the primary key for table

More Info Cancel < Back Next > Finish

Do not change any values for the data on this screen. Click Next.



**Database Export**

Specify the export mapping details:

Visio Data:

- PinX
- PinY
- Prop.Row\_1
- ResizeMode
- Rounding
- Shape.Data1
- Shape.Data2
- Shape.Data3
- Shape.Text
- ShdwBkgnd
- ShdwForegnd
- ShdwPattern
- Width

Export Mapping details:

Evaluate data as: Formula

Field Name: Width

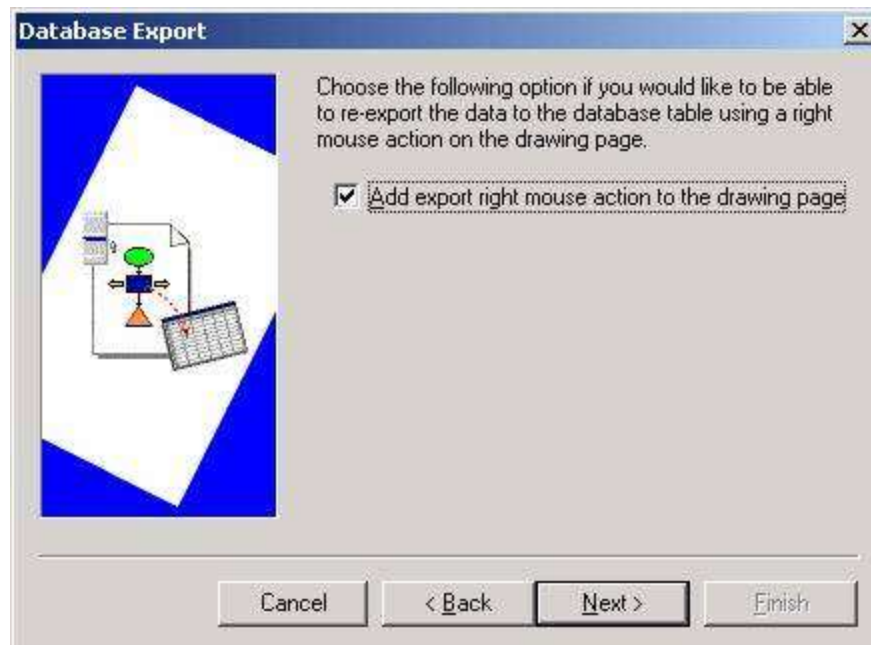
Field Type: VARCHAR

Field Size:

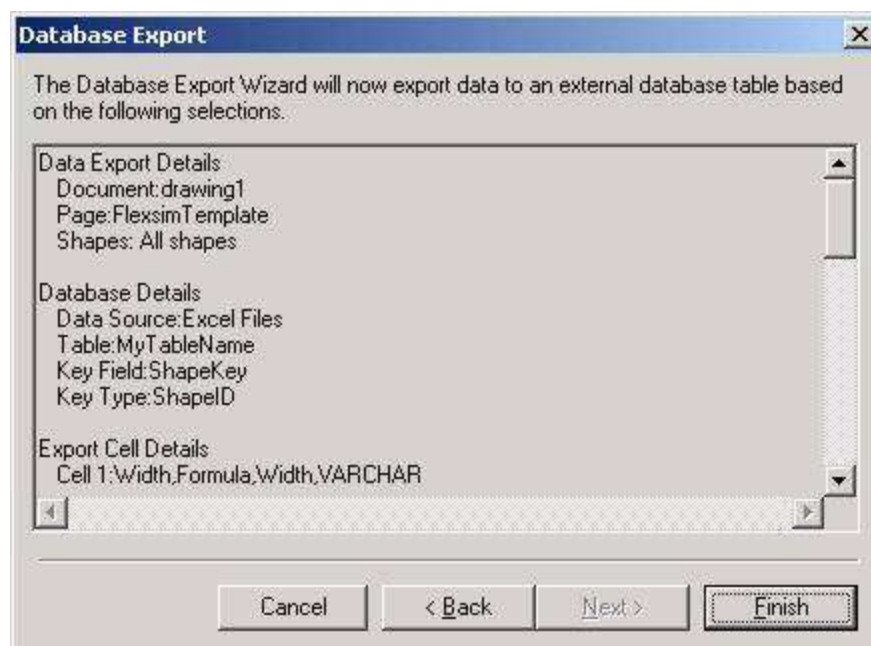
Field Decimal:

More Info Cancel < Back Next > Finish

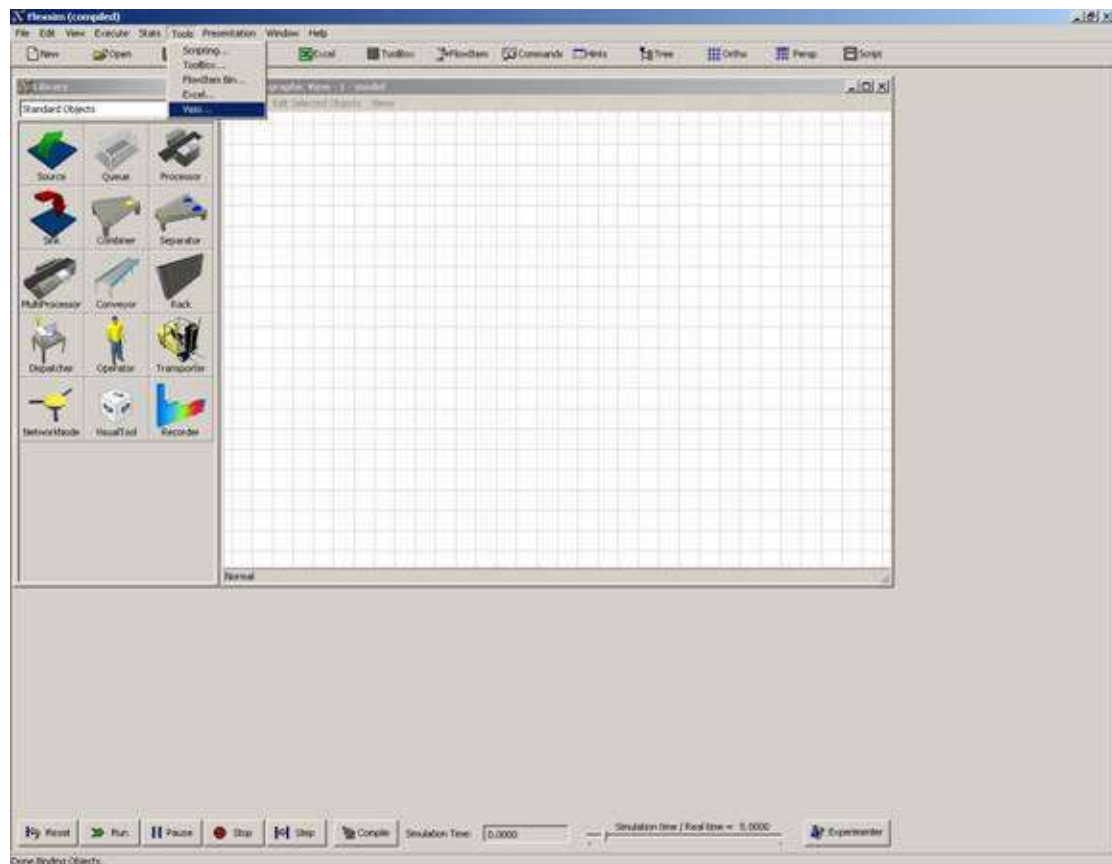
Next.



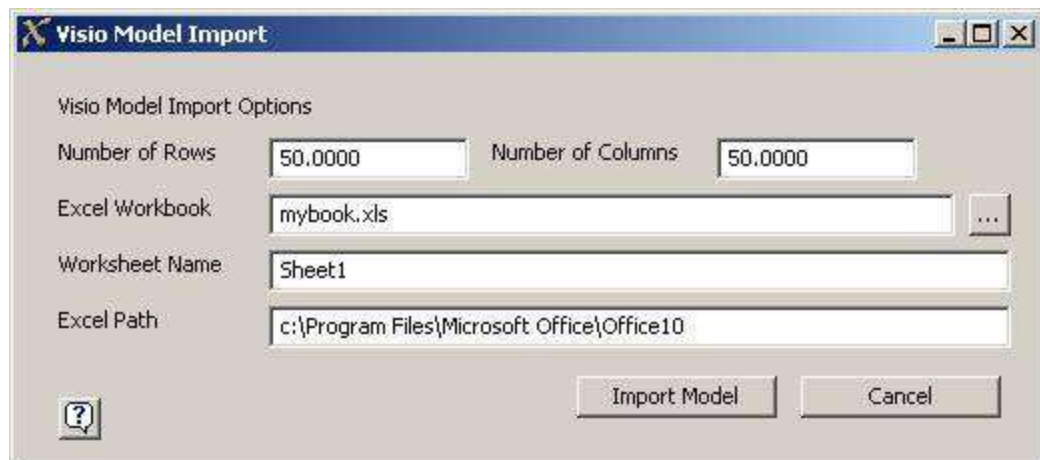
Finish.



Now, in Flexsim go to Tools > Visio...

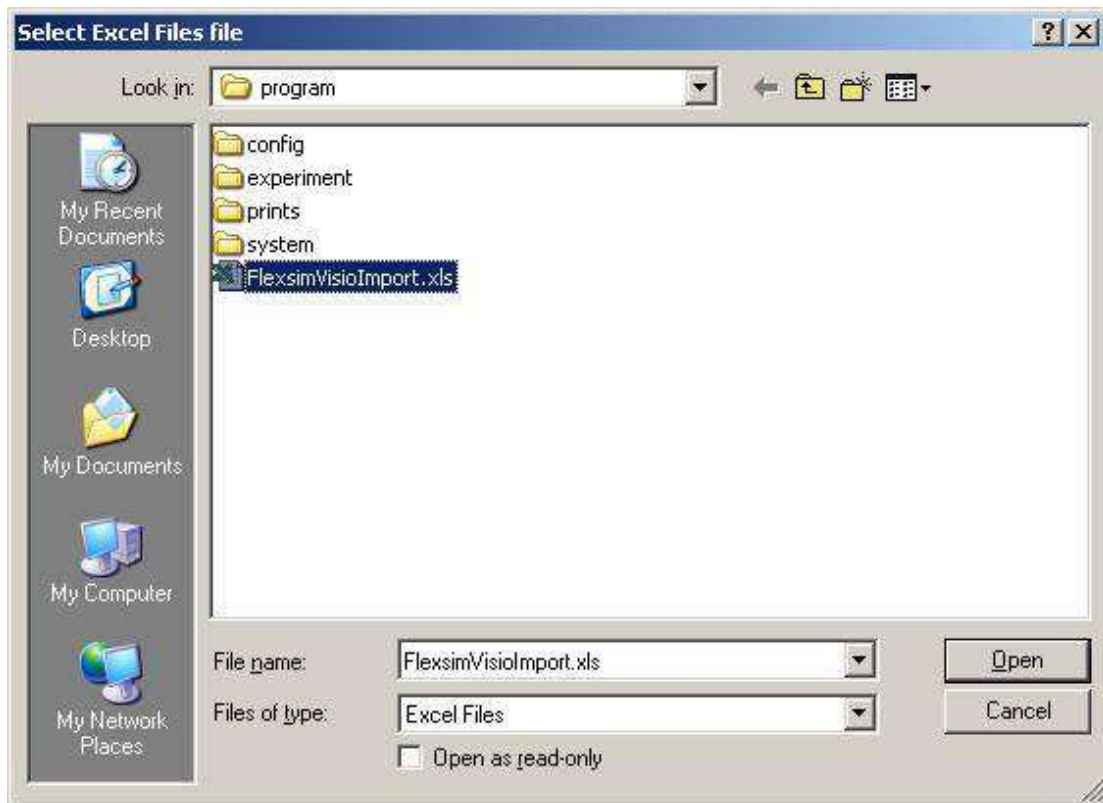


This dialog box will appear.

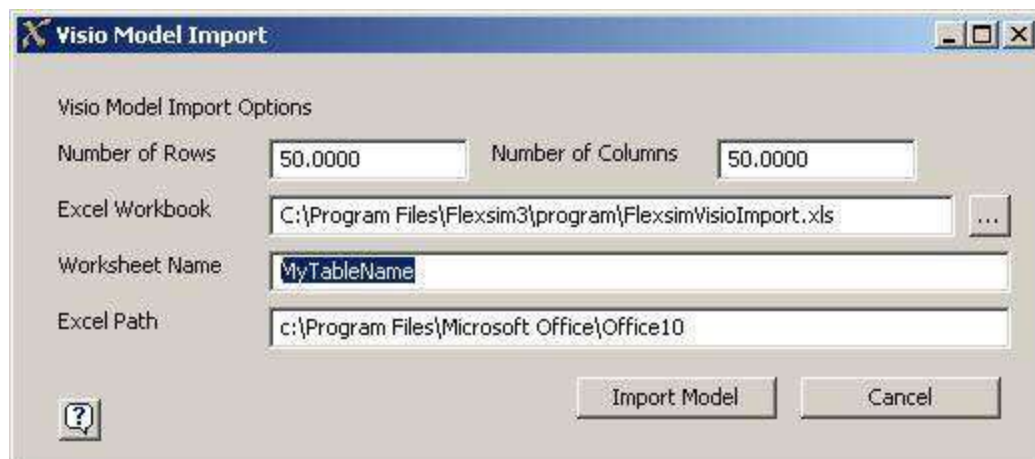


Enter a number into Number of Rows and Number of Columns that is larger than the number of rows and columns in the excel sheet that was just created by Visio Database Export. 50 is usually large enough. If your model does not import correctly, you may need to increase these values after checking your excel sheet to see how large it is.

Click the ... button to browse for the correct excel file to open.



Type the Table Name that was entered earlier into the space for Worksheet Name.



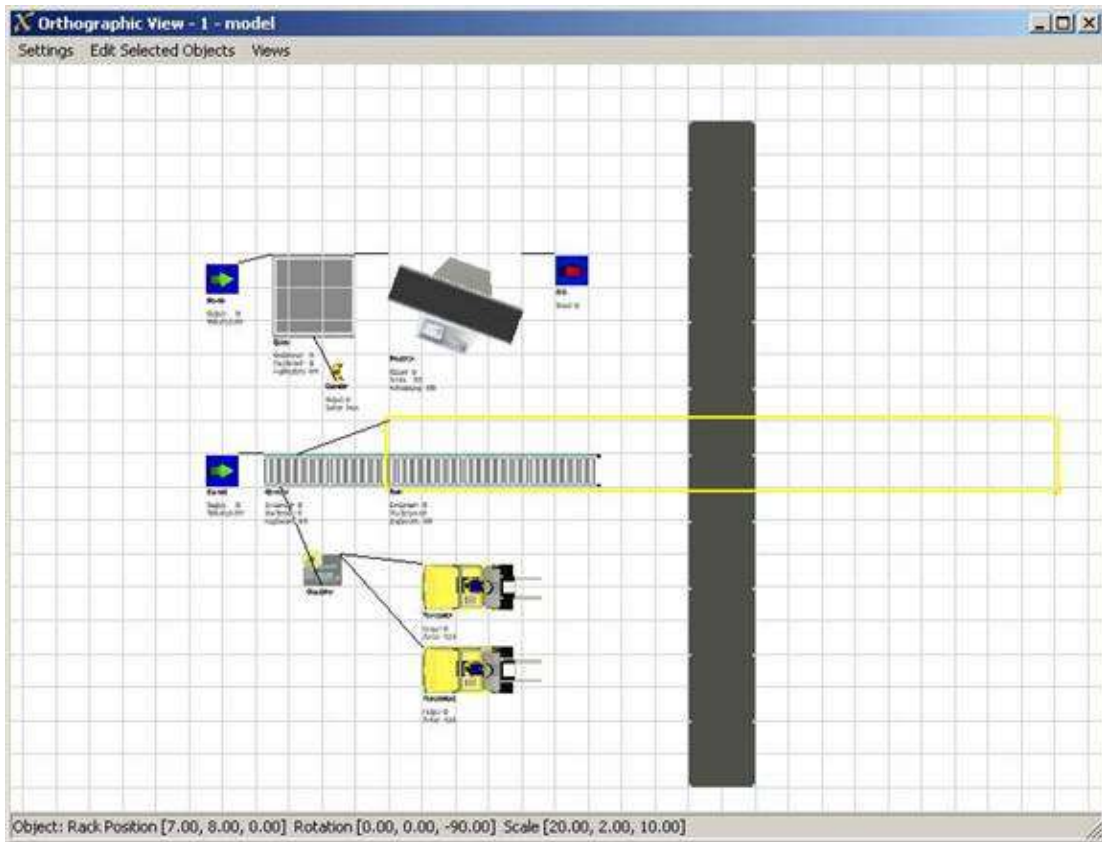
Be sure that the correct Excel Path is specified.

Click the Import button. This message box will appear.



Wait until Excel has completely opened the correct worksheet and then click OK.

The model will then be imported. This may take a few minutes. Wait for the Visio Model Import window to close. Your model will then be imported into Flexsim.





## Watch List

Watch Lists are accessed in the Tools menu. A Watch List is a list of variables that the modeler would like to keep an eye on. Variables are added to the table and are watched and noted when their value changes.

	ObjectName	WatchVariable	OldValue	ObjectAddr
Item1			0.00	0.00
Item2			0.00	0.00
Item3			0.00	0.00

**Name** - The name of the Watch List object. This should be something descriptive and easy to remember. For example, "Watching Servers", or "Watching Label WIP".

**Number of variables to watch** - The number of variables you would like to watch. Each variable must be added to the table.

### Watch List Table

Edit the watch list table to specify the object and variable that you want to watch.

**ObjectName** - Enter here the name of the object in the model.

**WatchVariable** - Enter here the name of the variable or attribute to watch on the object.

**OldValue** - This is the last recorded value for the object. Do not change this value.

**ObjectAddr** - This is a pointer to the object. Do not change this value.

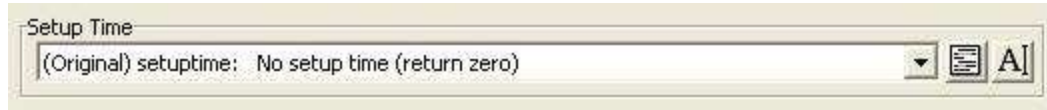
**OnChange Trigger** - This pick list specifies an action to be taken when a watched variable changes. See the OnChange trigger pick list.



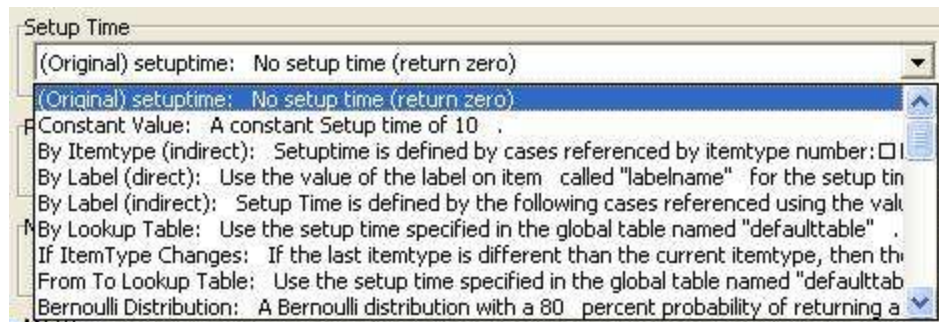


# Pick Lists


## Pick Lists



You will find pick list windows throughout Flexsim. These windows give you an easy interface for implementing functionality in Flexsim. Behind the scenes, each of these windows refers to a piece of code. The nice thing about these pick list interfaces is that they allow you to write functionality without writing code. They give you a list of commonly used functionality when you click on the drop-down box.



### Code Template

Once you have chosen a selection, you can customize that selection by clicking on the code template button . This shows a pop-up window explaining what the selection does. It also allows you to enter your own information for specific parameters highlighted in blue.




In the above example, the "By Global Table Lookup" option was selected in the drop down box. The code template window tells you that the value in the table named "labelname", row getitemtype(item), column 1, will be used as the setup time for Processor1.

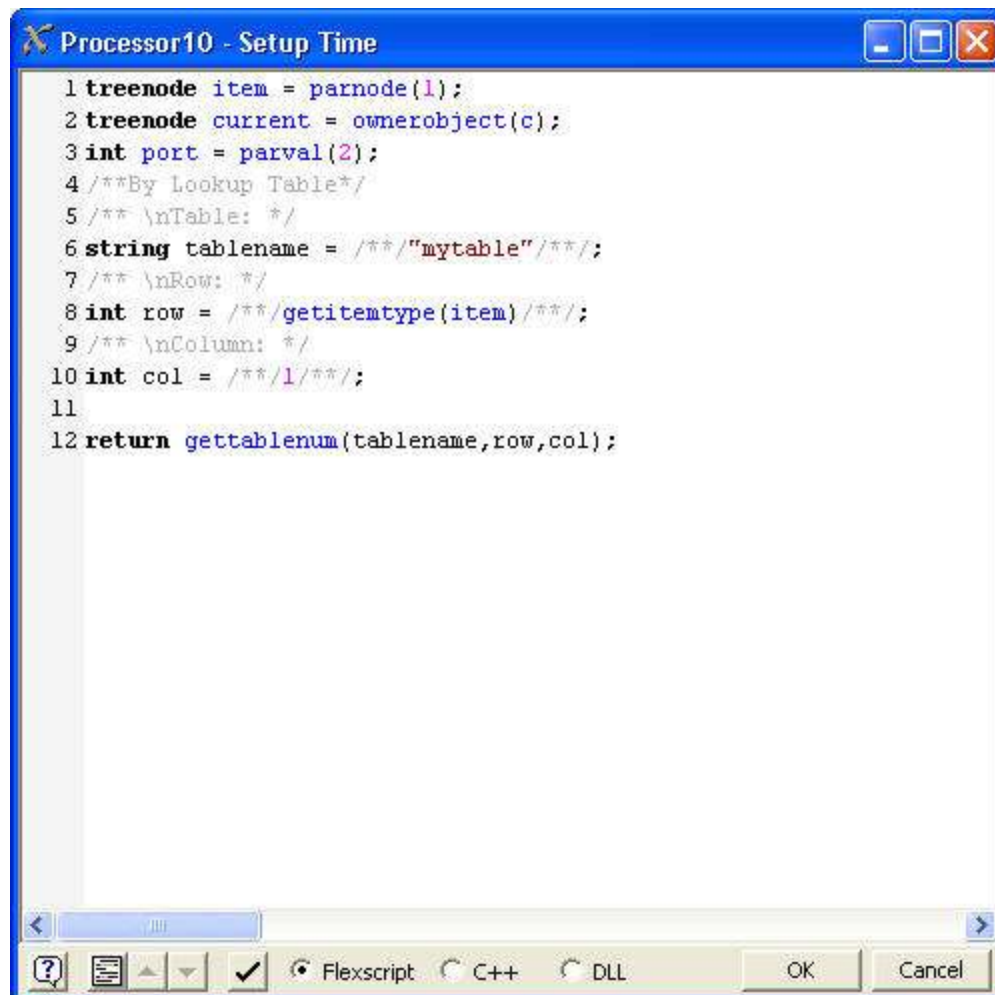
Again, when you select a picklist option and edit the code template window, behind the scenes you are really creating your own segment of code. The nice thing about it,



though, is that you only need to specify a minimal amount of information. You don't really have to write code.

### Code Edit

Experienced modelers also have the ability to write the code explicitly when needed.

By clicking on the code edit button , you can bring up the code edit window, in which you can see all of the code that implements this field. Note that much of the code you will see is actually used to format the code template window. You can decipher the real code from the code template formatting code by the color. Code template formatting code is commented out in gray.



Within the code window, you can specify whether you want your code to be interpreted as Flexscript or compiled as C++ (in which case you will need to compile your model). You can also check the Flexscript syntax by pressing the  button. You can also view the resulting template code from within this window by pressing the  button. Then press the up or down arrows to apply your changes to the template or code sections respectively.

For more information on how to write code in Flexsim, refer to writing logic in Flexsim.

### DLL Functionality

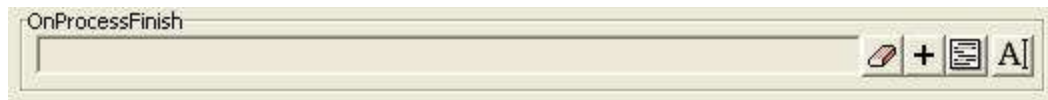
You can also specify the given field as accessing a function from a dll. In this case you would not provide the code as the text, but would provide the path to the dll as well as the name of the function to call. To create such dll you would need to use a special Visual C++ project. This project is available on the user community. The code field itself will need to specify two strings, each enclosed in quotes. The first string is the path to the dll. The second string is the name of the function. When you press the DLL radio button a message will appear that will let you create a template specifying the two strings.



### Locking Code

There is also a "Locked" checkbox at the bottom of the view. This checkbox should only apply to Flexscript or C++ code. It lets you lock the code state of the field to either Flexscript or C++. In the main Build menu, there are two options to make all code either C++ or Flexscript. We provide this option so that modelers can have both the ease of use of Flexscript (code works immediately when editing in Flexscript, without having to compile) as well as the run-speed of C++ (since it is compiled, it runs much faster than Flexscript). While in the model building phase you can use Flexscript, so that your code is interpreted immediately after you write it. Then, once your model is ready to run, you can choose the Build|Make all code C++ option, compile, and run to get the speed of C++. However, there may be some code that you write that cannot be converted from Flexscript to C++ or vice versa because it uses features specific to that language. In this case you would check the Lock checkbox to lock the code state of the given field, so that when you choose the Make all code C++ or Make all code Flexscript menu option, that field will not be switched over.

### Triggers

The interface for object triggers is slightly different than for standard pick lists. Here there are two buttons instead of a drop down box.



The  button erases all code for the trigger. The  button opens a drop-down menu where you can select a trigger option to add to the trigger. The other buttons work the same as with regular pick lists.

## Triggers

### Load/Unload Trigger (On Load/On Unload)

#### Overview:

Load Trigger: This trigger is fired once the TaskExecuter has finished its loadtime and just before it moves the flowitem into the TaskExecuter.

Unload Trigger: This trigger is fired once the TaskExecuter has finished its unloadtime and just before it moves the flowitem into its destination.

#### Access variables:

item : the item that is about to be loaded/unloaded

current : the current object

## Entry/Exit Trigger (On Entry/On Exit)

### Overview:

Entry Trigger: This function is executed each time a flowitem enters this object.

Exit Trigger: This function is executed each time a flowitem exits this object.

**Note on entry/exit trigger context:** Generally, the entry and exit triggers are executed at the very beginning of the OnReceive/OnSend event of an object, before any other logic is executed. This means that you can change the object's variables, labels, etc., and have those changes be applied correctly within the event logic. However, executing commands that may affect further events of the object should not be executed in the entry/exit trigger, because some events have yet to be created in the event logic of the object, and functions which affect the object's events should wait until those events have been created. In such a case you should send the object a delayed message in 0 time (using the `senddelayedmessage()` command), and then execute the functionality from the message trigger. This allows the object to finish the rest of its event logic before your commands are executed. Commands which apply in this setting are `stopobject()`, `requestoperators()`, `openoutput()`, `openinput()`, `resumeinput()`, `resumeoutput()`, and in some cases the creating and dispatching of task sequences, depending on the types of tasks that are in them.

### Access variables:

`current` : the current object

`item` : the involved object that just entered/exited

`port` : the number of the port that the object came/left through

## Message Trigger (On Message)

### Overview:

This function is executed when a message is sent to this object by the `sendmessage` or `senddelayedmessage` commands. Each command may pass up to three user-defined parameters.

### Access variables:

<code>current</code>	:	the current object
<code>msgsendingobject</code>	:	the object that sent the message
<code>msgparam(1)</code>	:	the message's first value parameter
<code>msgparam(2)</code>	:	the message's second value parameter
<code>msgparam(3)</code>	:	the message's third value parameter

## Process Finish Trigger (On Process Finish)

### Overview:

This code is executed each time a flowitem finishes its process time.

### Access variables:

current : the current object

item : the object that has finished its process time

## Creation Trigger (On Creation)

### Overview:

This code is executed when a flowitem is first created.

### Access variables:

current : the current object

item : the object that was just created

rownumber : the row number of the arrival (if one applies)



## Collision Trigger (Handle Collision)

### Overview:

The collision trigger is fired whenever an object executes its collision check and finds that it has collided with one of its collision members. For more information on collision detection, refer to TaskExecuter collision detection.

### Access variables:

thisobject : the current object

otherobject : the object that the current object collided with

thissphere : the involved sphere of the current object

othersphere : the involved sphere of the object that the current object collided with

## Node Entry Trigger (On Continue/On Arrival)

### Overview:

This function is executed when a traveler enters this node from any direction. OnArrival happens when a traveler arrives at the node. OnContinue happens when the traveler continues down the next path.

### Access variables:

current : the current object

traveler : the involved traveler that just entered

edgenum : the number of the edge that the traveler came through

## OnCover OnUncover Trigger Picklist

### Overview:

This function gets executed when the state of a photo eye changes.

### Access variables:

current : current object

item : the flowitem that is currently in front of the photo eye.

photoeye : photoeye number (row in the table)

covermode : For a cover trigger, 1 means a green to yellow state transfer, 2 means yellow to red. For an uncover trigger, 1 means a yellow to green transfer, 2 means a red to green transfer.

## OnChange Trigger

### Overview:

This function gets executed whenever any of the variables in the watchlist change. The station checks the variables on EVERY event in the model, so it is sure to catch a change when it happens.

### Access variables:

current	:	the current object
changedobject	:	the object whose variable changed
changeditem	:	the variable (node) that was changed
changedvalue	:	the new value of the variable
oldval	:	the old value of the variable

## OnDraw Trigger (Custom Draw Code)

### Overview:

This function is executed prior to the object's OnDraw Event. It is used to perform user-defined drawing commands and animation. If this function returns 1, the object's standard OnDraw function is not called. If it returns 0, OnDraw occurs normally. FlexScript, C++, and/or OpenGL code can be used to define what is drawn.

Common Commands:

```
drawcolumn(xloc,yloc,zloc,nrsides,baseradius,topradius,height,xrot,yrot,zrot,red,green,blue[,opacity,texture,xrep,yrep])
```

```
drawcube(xloc,yloc,zloc,xsize,ysize,zsize,xrot,yrot,zrot,red,green,blue[,opacity,texture,xrep,yrep])
```

```
drawcylinder(xloc,yloc,zloc,baseradius,topradius,height,xrot,yrot,zrot,red,green,blue[,opacity,texture])
```

```
drawdisk(xloc,yloc,zloc,innerradius,outerradius,startangle,sweepangle,xrot,yrot,zrot,red,green,blue[,opacity,texture])
```

```
drawline(view,x1,y1,z1,x2,y2,z2,red,green,blue)
```

```
drawobject(view,shape,texture)
```

```
drawrectangle(xloc,yloc,zloc,length,width,xrot,yrot,zrot,red,green,blue[,opacity,texture,xrep,yrep])
```

```
drawsphere(xloc,yloc,zloc,radius,red,green,blue[,opacity,texture])
```

```
drawtext(view,text,xloc,yloc,zloc,xsize,ysize,zsize,xrot,yrot,zrot,red,green,blue)
```

```
drawtomodelscale(object)
```

```
drawtoobjectscale(object)
```

```
drawtriangle(view,x1,y1,z1,x2,y2,z2,x3,y3,z3,red,green,blue)
```

```
spacerotate(x,y,z)
```

```
spacescale(x,y,z)
```

```
spacetranslate(x,y,z)
```

### Access variables:

current : the current object

view : the view that the object is being drawn in



## OnEmpty or OnFull Trigger

### Overview:

These triggers are called on fluid objects when their content reaches 0 or when it reaches their maximum content. These triggers are often used to open or close ports or send messages to other objects in the model.

### Access variables:

current : the current object

**There are no pick-options currently written for these triggers.**

## Reset Trigger (On Reset)

### Overview:

This function is executed when the model is reset.

### Access variables:

current : the current object



## Down/Up Trigger (On Break Down/On Repair)

### Overview:

Down Trigger: This function is executed when the MTBFMTTR object tells its members to break down.

Up Trigger: This function is executed when the MTBFMTTR object tells its members to come back up.

### Access variables:

current : the MTBFMTTR object

members : the members list in the MTBFMTTR object

involved : either 1) the MTBFMTTR object or 2) the involved member

curmember : the current member (will cycle through all members) - not available in all options

index : the rank in the members list of curmember

## **Breakdown/Repair Trigger (On Break Down/On Repair)**

### **Overview:**

Breakdown Trigger: This code is executed each time this object goes down.

Repair Trigger: This code is executed each time this object finishes its repair time.

### **Access variables:**

current : the current object

## OnResourceAvailable

### Overview:

This trigger is fired when a resource downstream from the Dispatcher becomes available. If the function returns a 0, the Dispatcher will do its own dispatching logic. If the function returns a 1, the Dispatcher will not do anything, and assumes all dispatching logic is done with this trigger using the `movetasksequence()` and `dispatchtasksequence()` commands.

### Access variables:

`current` : the current object

`port` : the output port of the dispatcher

`resource` : the downstream resource that has become available



## Time Pick Lists

### Time Picklists

**Note:** The following picklist options are used in several pick lists, including Cycle Time, Setup Time, MTBF, MTTR, Load/Unload, Minimum Stay Time, and Time picklists. Some of the options below are not included in all of the above mentioned pick lists because the context was not appropriate.

**Note on distribution pick list options:** Many of the pick options in these pick lists are distributions, and are not documented here. For more information on the different Flexsim distributions, refer to the ExpertFit documentation.

## Load/Unload Time

### Overview:

Load Time: Returns the value of the load time. The flowitem is moved into the TaskExecuter at the end of the load time.

Unload Time: Returns the value of the unload time. The flowitem is moved into the station at the end of the unload time.

### Access variables:

current : the current object

item : the involved flowitem

station : the object where the flowitem is being loaded or unloaded

For a list of pick options, see Time Picklists

## Minimum Staytime (Minimum Dwell Time)

### Overview:

This function returns the minimum time that each flowitem must reside in the Rack. When the flowitem enters the Rack, it executes this function, and creates an event in the returned. After the time has expired, the Rack releases the item. If this function returns a value of -1, the Rack will not create an event to release the item at all, and you will need to release the item explicitly using the `releaseitem()` command. This can be used if you want to implement your own releasing strategy for the Rack.

### Access variables:

`current` : the current object

`item` : the involved flowitem

`port` : the port the product entered through

For a list of pick options, see Time Picklists

## Setup Time

### Overview:

This function returns the setup time for the processing object.

### Access variables:

current : the current object

item : the involved flowitem

port : the port the product came in through

For a list of pick options, see Time Picklists

## **Time Picklist (Inter-Arrivaltime Usage)**

### **Overview:**

Returns the number of seconds between creation of flowitems.

### **Access variables:**

current : the current object

For a list of pick options, see Time Picklists



## Cycle Time (Process Time)

### Overview:

This function returns the process time for the processing object.

### Access variables:

current : the current object

item : the involved flowitem

For a list of pick options, see Time Picklists

---

## Fixed Resources

### Down Dispatcher (Pick Operator)

**Overview:**

This function returns a reference to a dispatcher (or operator) that the down operator request will be sent to. The return value must be cast into a double, because the function is hard-coded to return a double.

**Access variables:**

current : the current object

## Flow Rate

### Overview:

**Incoming Flow Rate:** Returns the number of seconds between flowitems entering a reservoir. This function is executed whenever a flowitem enters. Here the volume of the entering flowitem should be determined. If the reservoir has been full, and a flowitem exits make space available in the Reservoir, the in flow rate function is executed again from the OnSend event to determine the next time the flowitem.

**Outgoing Flow Rate:** Returns the number of seconds between flowitems leaving a reservoir. Usually this function is executed when a flowitem exits, to find the time to release the next flowitem. However, the function is also called for the first flowitem that enters, to get the time to release that flowitem.

### Access variables:

current : the current object

isentry : 1 means this function was executed from a flowitem's entry, 0 means it was executed from a flowitem's exit.

item : If isentry is 1, then this is a reference to the flowitem that just entered. Otherwise, it refers to the flowitem that just exited.

unitsnode : This is a reference to a node whose number value can be set to represent a volume for the item. Setting this node's value only applies if isentry is 1 and it is an incoming flow rate trigger, not an outgoing flow rate trigger.

## Place in Bay

### Overview:

This function returns the bay number in which to place the entering flowitem. Bays are numbered starting at 1 for the bay closest to the Rack's origin and increasing numerically going away from the origin.

### Access variables:

current : the current object

item : the involved flowitem

## Place in Level

### Overview:

This function returns the level number in which to place the entering flowitem. Levels are numbered starting at 1 for the level closest to the Rack's origin and increasing numerically going away from the origin

### Access variables:

current : the current object

item : the involved flowitem

## Process Dispatcher (Pick Operator)

### Overview:

This function returns a reference to a dispatcher (or operator) that the operator request will be sent to. The return value must be cast into a double, because the function is hard-coded to return a double.

### Access variables:

current : the current object

item : the involved flowitem

## Receive From Port (Pull From Port)

### Overview:

When this object is in pull mode and it is ready to receive its next flowitem, it will first evaluate this Pull From Port field to determine which input port to open. If the Pull From Port field returns a zero (0) for the input port number, then it will open all of its input ports. When an input port is opened, the OnInOpen event of this object will execute immediately if the connected upstream output port is already open, or the OnInOpen event will fire in the future when the upstream output port is eventually opened. When the OnInOpen event fires, the PullRequirement will be evaluated, determining which flowitem will actually get pulled in.

### Access variables:

current : the current object

## Pull Requirement

### Overview:

This field is a boolean expression returning either true (1) or false (0). It is evaluated in the OnInOpen event of this object whenever an input port becomes ready. An input port is ready when it is open AND the upstream output port that it is connected to is open. The connection becomes ready when either the input port is opened, OR the upstream output port is opened (assuming the paired port is already open). This Pull Requirement expression will be evaluated for each ready flowitem within the object connected to the input port which just became ready causing the OnInOpen event to fire. If the expression returns a true (1), then the ready flowitem will be pulled in through the ready input port.

### Access variables:

current : the current object

item : the ready flowitem for which the Pull Requirement is currently being evaluated

port : the number of the input port that became ready



## Rise/Fall Through Mark Triggers

**Overview:**

This code is executed when the content rises up or falls down through the user defined high, low, or middle mark.

**Access variables:**

current : the current object

item : the involved flowitem

## Send Requirement

### Overview:

This field is evaluated on a MergeSort when a flowitem reaches an output position on the conveyor. It should return a true or false (1 or 0) as to whether the flowitem should be released out the respective port.

### Access variables:

current : the current object

item : the object that is ready to be sent

port : the output port that the flowitem has arrived at.

## Send To Port

### Overview:

This field is evaluated once for each flowitem at the time the flowitem is ready to be sent to the next object. In a Processor object, the flowitem is ready to be sent at the end of its processing time. In a Queue, the flowitem is ready to be sent after the batch has accumulated and has been released.

The SendToPort expression must return a valid output port number. The output port will be opened, and the port number will be assigned to the flowitem. The flow item will then be pushed (or pulled) when the port connection becomes ready (output port and connected downstream input port are open). In the special case where the SendToPort expression returns a 0, all output ports will be opened, and the flowitem may leave through the first ready port. If the object is configured to continuously evaluate sendto, then this field is also evaluated every time a downstream object becomes ready to receive a flowitem. If the function returns a -1, then the flowitem will not be released at all, and should be released later on using the releaseitem() command, or should be moved out using the moveobject command.

### Access variables:

current : the current object

item : the object that is ready to be sent

## **Split Quantity (Split/Unpack Quantity)**

### **Overview:**

This function returns the number of items to split/unpack from the current item.

### **Access variables:**

current : the current object

item : the involved object. This is the item that will be split/unpacked

## Transport Dispatcher (Request Transport From)

### Overview:

This function returns a reference to the dispatcher (or transport) that the transport request will be sent to. The return value needs to be cast into a number, since this function is hard-coded to return a double.

### Access variables:

current : the current object

item : the item to be transported

## Item Speed (Speed)

### Overview:

This field is evaluated once for each flowitem when the flowitem enters. It returns the speed of that flowitem

### Access variables:

current : the current object

item : the item that is ready to be sent

The item speed pick list uses the same options as the Time pick list, except the return value is interpreted as a speed instead of a time.

---

## Mobile Resources

### Break To (“Break To” Requirement)

#### Overview:

Define what type of tasksequences this object will accept during a "break" task in its active tasksequence. A standard tasksequence is made up of a travel - load - break - travel - unload sequence of tasks. This field will only be evaluated when the TaskExecuter performs a "break" task.

When the TaskExecuter receives a "Break" task, this is a notification that it may now put the currently active task sequence back in its queue and see if there are any other task sequences that it might want to do before it finishes the current one. This allows for capabilities such as loading several items before dropping them off. In such a case, the TaskExecuter will first make a check to make sure it is currently capable of "multitasking." This is done by asking "is my content less than my capacity?" If the check is true, then it will execute this code to find a tasksequence to break to. If this code returns NULL, then it will not break at all. If a valid tasksequence numeric pointer is returned, then it will break to the new tasksequence and begin executing it. When finished with new tasks, it will return to the original tasksequence.

#### Access variables:

tasksequence : a reference to the tasksequence that I'm checking

current : the current object

## Pass To

### Overview:

This function is fired when the Dispatcher receives a task sequence, and should return the output port that the Dispatcher will pass the tasksequence to. If 0 is returned, the tasksequence will automatically queue up according to the defined Queue Strategy until the tasksequence can be passed to an available Dispatcher or TaskExecutor. If a value greater than 0 is returned, the tasksequence will be sent immediately to the returned port number. If a value of -1 is returned, then the Dispatcher does nothing, but rather assumes all dispatch logic is done within the passto function using the movetasksequence() and dispatchtasksequence() commands.

### Access variables:

tasksequence : a reference to the tasksequence node

current : the current object



## Queue Strategy

### Overview:

When the Dispatcher receives a tasksequence, and the "Pass to" function returned a 0, then when a new tasksequence enters the Dispatcher's queue, the Dispatcher goes through his queue of tasksequences, and executes this same function on each tasksequence, and compares it with the value returned for the tasksequence just received. The new tasksequence will then be sorted from highest to lowest (highest value returned will be sorted to the front of the tasksequence queue) according to the value it returned in this function.

### Access variables:

tasksequence : a reference to the tasksequence node

current : the current object

## Experimentation

### End of Experiment

**Overview:**

This function is executed once at the end of the last run of the last scenario.

**Access variables:**

replication : current replication number

scenario : current scenario number

## End of Run (End of Replication)

### Overview:

This function is executed at the end of the user-defined simulation time for each replication run.

### Access variables:

replication : current replication number

scenario : current scenario number

## End of Scenario

### Overview:

This function is executed after the last replication of each scenario.

### Access variables:

replication : current replication number

scenario : current scenario number

## End of Warmup (End of Warmup Period)

### Overview:

This function is executed at the end of the user-defined warmup time for each replication run. The time and system variables are reset, but the flowitems remain where they are.

### Access variables:

replication : current replication number

scenario : current scenario number

## Pass To

### Overview:

This function is called at the end of each replication of a scenario, and returns a performance measure value to be recorded for the current replication.

### Access variables:

There are no access variables for the performance measure function. Use `model()` as a starting point.

## Start of Experiment

### Overview:

This function is executed once at the start of the experiment (before model reset).

### Access variables:

replication : current replication number

scenario : current scenario number

## Start of Run (Start of Replication)

### Overview:

This function is executed at the beginning of each replication run (after model reset).

### Access variables:

replication : current replication number

scenario : current scenario number



## Start of Scenario

**Overview:**

This function is executed before the first replication of each new scenario (before model reset).

**Access variables:**

replication : current replication number

scenario : current scenario number



## Visualization

### Table X Val (X Value)

**Overview:**

This function is executed for every data point drawn on a Recorder object. Each data point of a Recorder graph is made up of an x value and a y value. This function returns the x value.

**Access variables:**

current : a reference to the recorder object.

curcount : the current data point number

## Table Y Val (Y Value)

### Overview:

This function is executed for every data point drawn on a Recorder object. Each data point of a Recorder graph is made up of an x value and a y value. This function returns the y value.

### Access variables:

current : a reference to the recorder object.

curcount : the current data point number

## **Text (Text Display)**

### **Overview:**

This function was used with the VisualText object, but is now deprecated. The VisualTool object is now used to display text.

## Text Code (Text Display)

### Overview:

This function is executed every time the VisualTool is redrawn to display 3d text in the model view window. Changes made in this field will not be seen until after a successful compile.

### Access variables:

current : the current object

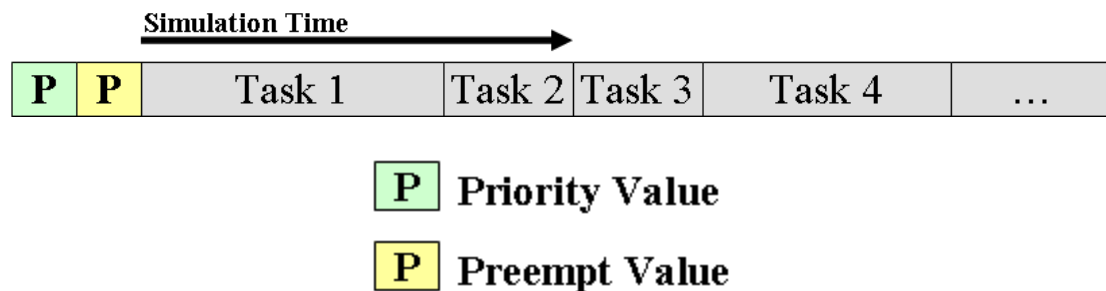


# Task Sequences

## Flexsim Task Sequences

### What is a Task Sequence?

A task sequence is a series of tasks to be executed in sequential order by a TaskExecutor, as shown in the figure below. The term TaskExecutor implies any object that inherits from the TaskExecutor class. This includes Operators, Transporters, Cranes, ASRSvehicles, Robots, Elevators, and other mobile resource objects. An object is a TaskExecutor if the TaskExecutor tab page is one of the pages in its Parameters window.



In addition to being a series of tasks, each task sequence has a priority value, which defines the importance of executing that task sequence with respect to other task sequences. Each task sequence also has a preempt value, which defines whether that task sequence should cause other task sequences' execution to be halted in order to execute this task sequence.

### Automatically Created Task Sequences

FixedResources have a default mechanism for creating task sequences to move flow items to the next station. You can use this default functionality by checking the "Use Transport" box in the "Flow" tab page of the FixedResource's Parameters. Processors also have a default mechanism for creating task sequences to call operators for setup times, process times and repair operations. This is done by modifying the "Operators" tab page on a Processor, Combiner, or Separator. Each of these default mechanisms triggers a task sequence to be automatically created.

### How Task Sequences Work

When you check the "Use Transport" field on a Flow tab page, the following task sequence is created.

1. Travel to the object currently holding the item
2. Load the item from that object
3. Break
4. Travel to the destination object

5. Unload the item to the destination object

P	P	Travel	Load	Break	Travel	Unload
---	---	--------	------	-------	--------	--------

When a TaskExecutor executes this task sequence, it will execute each task in order. Each task mentioned above corresponds to a specific task type. Notice in the above example that there are two “Travel” task types in the task sequence, one “Load” task type, one “Unload” task type, and one “Break” task type.

### Travel Task

The “Travel” task type tells the TaskExecutor to travel to some object in the model. This may be done in several different ways, depending on the model’s setup. If the TaskExecutor is connected to a network, then the travel task will cause it to travel along the network, arriving at a network node that is connected to the desired destination object. If the TaskExecutor is a Crane object, then it will lift up to a modeler-defined height, then travel to the X/Y location of the destination object. Hence, a travel task can imply several things, depending on the setup of the model, as well as the type of object that is being used. The one thing, however, that all travel type tasks have in common is that they all have some destination object in the model that they are trying to get to.

### Load and Unload Tasks

The “Load” and “Unload” task types tell the TaskExecutor to load or unload a flow item into or out of a station. This usually involves traveling an offset distance in order to pick or place the item at the right spot, as well as going through a modeler-defined load or unload time before transferring the item. While the load/unload time is handled in the same way for all TaskExecutors, the offset travel may differ depending on the type of TaskExecutor. A Transporter, for example, will travel to the pick/place location while lifting its fork up to the pick/place height. A robot, on the other hand, will rotate to the spot where the item should be picked/placed. For more detail, see offset travel.

### Break Task

The “Break” task type tells the task executor to check if there are any other task sequences that it can “break” to. For example, if a transporter has two items that are waiting to be picked up from the same location, and it has the ability to load two or more items, then the transporter would have two task sequences to do. Both of them would be like the above mentioned task sequence. One is the active task sequence to pick up the first item, and the other task sequence is placed in his task sequence queue to be executed once he finishes the active task. The break task allows the transporter to stop the first task sequence after he has loaded the first item, and begin the second task sequence, which is to travel to the second item’s station and load the second item. If the task sequence did not contain the break task, the TaskExecutor would have to finish the first task sequence completely, unloading the first item before being able to load the second item.

## Operator Task Sequences



Here's another example of an automatically created task sequence. The Processor object creates this task sequence to request an operator to work at the processing station. The task sequence is described as follows:

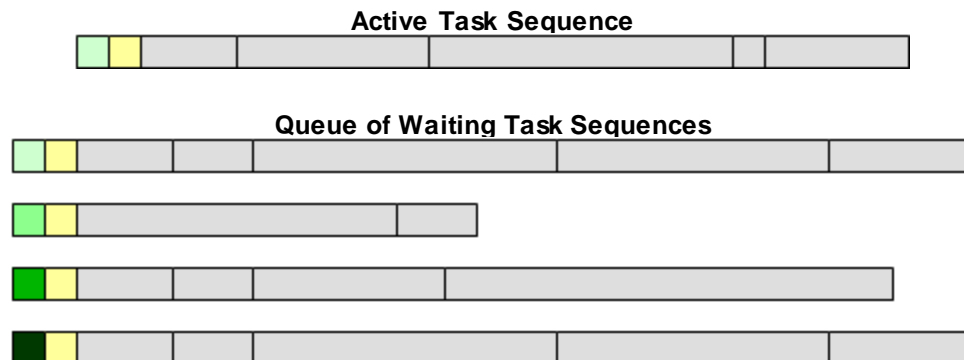
1. Travel to the processing station
2. Be utilized until freed from the processing station.



As in the previous example, the first task tells the TaskExecutor to travel to the station. The second task is a new task type not mentioned in the previous example. This is a “Utilize” task type. It tells the TaskExecutor to go into a given state, like “Utilized” or “Processing”, and then wait until it is freed from that state. The operator is freed when the freeoperators() command is called. Since the Processor automatically creates this task sequence, it automatically handles freeing the operator as well.

**Note:** an operator task sequence is not created exactly as described above. In reality there are more tasks added. For simplification purposes, though, we present the above example. For more information on the operator task sequence, refer to the documentation on the requestoperators command in the command summary.

At any given time in a simulation, a TaskExecutor can have one active task sequence as well as a queue of waiting task sequences. A Dispatcher object, on the other hand, can have a queue of waiting task sequences, but cannot actively execute any of those task sequences. Rather, it dispatches its queue of task sequences to TaskExecutors connected to its output ports. This is what distinguishes the Dispatcher object from the TaskExecutor and its sub-classes.



If no task sequences are preempting, then a TaskExecutor will execute its active task sequence until that task sequence is finished. Then it will make the first task sequence in its queue the active task sequence and start executing that one. This repeats until all task sequences in the queue have been executed.

## More Task Types

At this point you have been introduced to 5 task types, namely travel, load, unload, break, and utilize. There are several more task types that can be used when creating task sequences. For a more detailed explanation of each task type and its usage, see Task Types. The next section will address how to create and customize your own task sequences.

## Custom Built Task Sequences

You can also create custom task sequences using 3 simple commands:

```
createemptytasksequence()
inserttask()
dispatchtasksequence()
```

First, create a task sequence by using `createemptytasksequence()`. Then insert tasks into the task sequence by successive `inserttask()` commands. Finally dispatch the task sequence with `dispatchtasksequence()`.

The following example tells a forklift to travel to an object, referenced as “station”, then load a flow item, referenced as “item.”

```
treenode newtasksequence = createemptytasksequence(forklift, 0,0);
inserttask(newtasksequence, TASKTYPE_TRAVEL, station);
inserttask(newtasksequence, TASKTYPE_LOAD, item, station, 2);
dispatchtasksequence(newtasksequence);
```

If you are confused by the “treenode” syntax, refer the help on the Flexsim tree structure. In brief terms, “treenode newtasksequence” creates a reference, or pointer, to the task sequence as a Flexsim node so that it can be used later when tasks are added to the task sequence.

The `createemptytasksequence` command takes three parameters. The first parameter is the object that will handle the task sequence. This should be a Dispatcher or TaskExecutor object. The second and third parameters are numbers, and specify the task sequence’s priority and preempting values, respectively. The command returns a reference to the task sequence that was created.

The `inserttask` command inserts a task onto the end of the task sequence. Each task that you insert has several values associated with it. First, it has a type value, which defines what type of task it is. It also has a reference to two involved objects for the task, referred to as `involved1` and `involved2`. These involved objects and what they mean depend upon the task type. For some task types both involved parameters are needed and have meaning, whereas for others, the involved objects are not used. Some task types may use one involved object, and some have involved objects which are optional. Refer to the documentation on task types for information on what a specific task type’s involved objects represent. The task can also have up to four number values. These are task variables, referred to as `var1`, `var2`, `var3`, and `var4`. Again, their meaning depends on the task type. For the load task below, notice that `var1` was specified as 1. For a load task, this specifies the output port through which the item will leave the station.

<b>P</b>	<b>P</b>	<b>Travel</b>	<b>Load</b>
<b>Task Type: Load</b>			
<b>involved1 : object to load: item</b>			
<b>involved2 : object to load from: current</b>			
<b>var1 : output port: 1</b>			
<b>var2 : N/A : 0</b>			
<b>var3 : N/A : 0</b>			
<b>var4 : N/A : 0</b>			

The inserttask command takes two or more parameters, which specify the task's values. The first parameter is a reference to the task sequence into which the task is inserted. The second is the type of task. This can be chosen from an enumerated list of task types. The third and fourth parameters reference the two involved objects. If a specific involved object is not used or is optional for a task type, then you can simply pass NULL into the inserttask command, or even leave that parameter out if there are no number variables that you need to specify either. The fifth through ninth parameters are optional, and define var1-var4. By default, these values are zero.

**Note on optional parameters:** Even though many of the parameters of the inserttask command are technically optional, depending on the task type, you will still need to specify them. Also, parameters need to still be specified in their correct order. If, for example, you want to specify var1 of the task, but don't care what involved1 or involved2 are, you will still need to pass the NULL value into parameters 3 and 4, even though they are optional, in order to correctly pass var1 in as parameter 5.

## Querying Information on Task Sequences

Once you have dispatched task sequences, you can also query and change certain values on those task sequences. The following commands allow you to make such queries and changes.

`treenode gettasksequencequeue(treenode dispatcher)`

This command returns a reference to the task sequence queue of a dispatcher/taskexecutor object. It can be treated like a regular treenode. Say, for example, I am a dispatcher, and I want to query things on the first task sequence in my queue. I could access the task sequence by:

`first(gettasksequencequeue(current))`

`treenode gettasksequence(treenode dispatcher, int rank)`

This command is another way that you can get references to task sequences. Rank is the rank of the task sequence in the task sequence queue. Also, if rank = 0, then it will return a reference to the currently active task sequence for the task executor, or the task sequence that he is doing right now.

`treenode gettaskinvolved(treenode tasksequence, int rank, int involvednum)`

This command returns a reference to an involved object for a given task in a task sequence. Rank is the rank of the task in the task sequence. Involvednum is a 1 or a 2, and references which involved object. Say, for example, that a TaskExecutor is about to do a load task, but he wants to know the object from whom he is loading the item. In a load task, involved1 is the item, and involved2 is the station from which to load. Let's say also that I know that the load task is the 3rd task in the sequence, and the task sequence is currently the active task sequence to get a reference to the station, I would code:

`gettaskinvolved(gettasksequence(current,0), 3, 2)`

You'll need to know task types and which involved means what for a given task type, but once you know that, it's easy. Most of it is documented in the begintask() method of the TaskExecutor in the library.

`int gettasktype(treenode tasksequence, int rank)`

This command returns the task type of a given task. Rank is the rank in the task sequence. You can compare this with macros like TASKTYPE\_LOAD, TASKTYPE\_TRAVEL...

`int getnroftasks(treenode tasksequence)`

Returns the number of tasks in the task sequence that have not yet been finished.

`int gettotalnroftasks(treenode tasksequence)`

Returns the total number of tasks in the task sequence.

`int gettaskvariable(treenode tasksequence, int rank, int varnum)`

Returns the value of a variable in the task of a tasksequence. Again, rank is the rank of the task in the task sequence. Varnum is a number between 1 and 4, and is the variable number. As in the involved objects, variable numbers and what they mean depend on the task type.

`int getpriority(treenode tasksequence)`

Returns the priority of a given task sequence.

`void setpriority(treenode tasksequence, double newpriority)`

Sets the priority of the tasksequence

`int getpreempt(treenode tasksequence)`

Returns the preempt value for the task sequence. You can compare this with PREEMPT\_NOT , PREEMPT\_ONLY, PREEMPT\_AND\_ABORT\_ACTIVE, PREEMPT\_AND\_ABORT\_ALL. For information on preempting, go to Task Sequence Preempting.

`void setpreempt(treenode tasksequence, int newpreempt)`

Sets the preempt value of a task sequence. You would pass into newpreempt one of the previously mentioned macros. For information on preempting, go to Task Sequence Preempting.

## Task Sequence Preempting

Every task sequence has a preempting value. Preempting is used to break a TaskExecuter away from its current operation to execute a more important operation. For example, operator A's most important responsibility is to repair machines. When there are no machines to repair, however, it should also transport material throughout the model. If a machine breaks down while operator A is in the middle of transporting a flowitem somewhere, then the operator should stop whatever he is doing and repair the machine, instead of finishing the transport operation. To do this, you will need to use a preempting task sequence to break the operator away from his current operation.

To create a preempting task sequence, specify a non-zero value in the preempt parameter of the `createemptytasksequence()` command.

```
createemptytasksequence(operator, 0, PREEMPT_ONLY);
```

There are four possible preempt values. These values tell the TaskExecuter what to do with the original task sequence(s) that have been preempted.

- **0 - PREEMPT\_NOT** - This value is non-preempting.
- **1 - PREEMPT\_ONLY** - If a task sequence has this value, then the TaskExecuter will preempt the currently active task sequence and put it back in its task sequence queue to be finished later. When a task sequence is preempted, it is automatically placed at the front of the task sequence queue. When the TaskExecuter eventually comes back to the original task sequence, the current task in that task sequence will be done over again, since it was not finished. Also, you can specify a series of tasks to do over again when it comes back to the task sequence using the `TASKTYPE_MILESTONE` task. This preempt value is the most common used.
- **2 - PREEMPT\_AND\_ABORT\_ACTIVE** - If a task sequence has the `PREEMPT_AND_ABORT_ACTIVE` value, then the TaskExecuter will stop the currently active task sequence and destroy it, so that it will never come back to that original task sequence.
- **3 - PREEMPT\_AND\_ABORT\_ALL** - If a task sequence has the `PREEMPT_AND_ABORT_ALL` value, then the TaskExecuter will stop the currently active task sequence, destroy it, and destroy all task sequences in its task sequence queue.

### Interaction Between Multiple Preempting Task Sequences

If a TaskExecuter is currently working on a preempting task sequence, and it receives a new task sequence that is also preempting, it will use the priority value of the task sequence to determine which task sequence to do. If the priority value of the new task sequence is higher than the priority value of the one it is currently working on, the TaskExecuter will preempt its current task sequence and execute the new one. If the priority value of the new task sequence is less than or equal to the priority of the task sequence it is currently working on, then the TaskExecuter will not preempt the active task sequence, but will queue up the new task sequence just like any other task sequence it receives. If it must queue up the task sequence, it will not

take the preempt value into account for its queueing logic unless you explicitly tell it to in the queue strategy.

**Note on queueing a preempting task sequence:** If a preempting task sequence does not actually preempt a TaskExecutor, then it will be queued up like any other task sequence. If you want to have preempting task sequence be brought to the front of the queue, then either make your preempting task sequences have higher priority than all other task sequences, or take preempting into account in the queue strategy.

**Note on dispatching a preempting task sequence to a Dispatcher:** If a preempting task sequence is given to a Dispatcher, the Dispatcher will not consider the preempt value of the task sequence unless you explicitly tell it to. If the Dispatcher is set to dispatch to the first available TaskExecutor, then it will do just that, and not send the preempting task sequence immediately to a TaskExecutor. If you want the Dispatcher to dispatch preempting task sequences immediately, then you will need to specify such logic in its Pass To function.

To query or change the preempting and/or priority values of a task sequence, you can use the `getpreempt()`, `setpreempt()`, `getpriority()`, and `setpriority()` commands. For more information on these commands, refer to the command summary.

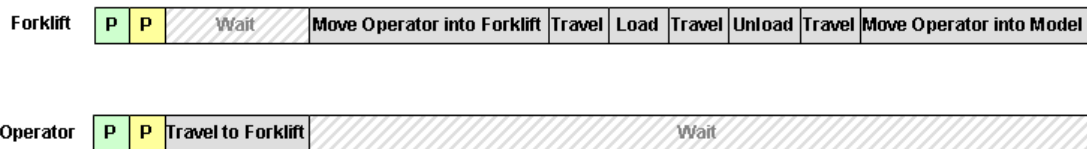


## Coordinated Task Sequences

Coordinated task sequences are used for operations which require sophisticated coordination between two or more TaskExecuters. These task sequences implement concepts like allocation and de-allocation of TaskExecuters, as well as synchronizing several operations being done in parallel.

### Example

A team of three operators share two forklifts. An operation needs one operator and one forklift. The operator should travel to the forklift, and the forklift should then move the operator into itself. Then the forklift should travel to the load location, pick an item, then travel to an unload location and drop off the item. Then the forklift should travel to its parking location, and unload the operator. Doing this using simple task sequences would be very difficult, because it deals with two different resources that work in a very coordinated fashion. Coordinated task sequences make this example much easier to simulate. The diagram below illustrates the two task sequences that need to be done for the forklift and operator. Notice that there are some parts where one resource needs to wait and do nothing while the other operates.



### Commands

Coordinated task sequences are built and dispatched using a set of commands which are mutually exclusive from the default task sequence commands. The commands for coordinated task sequences are as follows.

```
createcoordinatedtasksequence()
insertallocatetask()
insertdeallocetask()
insertsynctask()
insertproxytask()
dispatchcoordinatedtasksequence()
```

For the previous example, the code to build the task sequence would be written as follows. It is assumed that references called `operatorteam` and `forkliftteam` have been established. These reference dispatchers to three Operator objects, and two Transporter objects, respectively. References have also been established for a loadstation from which to load, an unloadstation to unload to, and the item.

```
treenode ts = createcoordinatedtasksequence(operatorteam);
int opkey = insertallocatetask(ts, operatorteam, 0, 0);
int forkliftkey = insertallocatetask(ts, forkliftteam, 0, 0);
int traveltask = insertproxytask(ts, opkey, TASKTYPE_TRAVEL, forkliftkey, NULL);
insertsynctask(ts, traveltask);
insertproxytask(ts, forkliftkey, TASKTYPE_MOVEOBJECT, opkey, forkliftkey);
insertproxytask(ts, forkliftkey, TASKTYPE_TRAVEL, loadstation, NULL);
```

```

insertproxytask(ts, forkliftkey, TASKTYPE_LOAD, item, loadstation);
insertproxytask(ts, forkliftkey, TASKTYPE_TRAVEL, unloadstation, NULL);
insertproxytask(ts, forkliftkey, TASKTYPE_UNLOAD, item, unloadstation);
insertproxytask(ts, forkliftkey, TASKTYPE_TRAVEL, forkliftteam, NULL);
insertproxytask(ts, forkliftkey, TASKTYPE_MOVEOBJECT, opkey, model());
insertdeallocatetask(ts, forkliftkey);
insertdeallocatetask(ts, opkey);
dispatchcoordinatedtasksequence(ts);

```

**Note on the above example:** There are some model maintenance issues involved here. For example, if you happen to stop and reset the model while the operator is inside of the forklift, you will need to move the operator out of the forklift and back into the model from a reset trigger. Also, whenever you move the operator back into the model, you will need to set its location appropriately, since it is transferring between two different coordinate spaces.

### createcoordinatedtasksequence

The createcoordinatedtasksequence command takes one parameter, namely a reference to an object. This object is designated as the “task coordinator” who holds the task sequence, as well as coordinates the tasks. The task coordinator can also be one of the objects that is allocated within the task sequence. It can be any Dispatcher or TaskExecutor object. Note that selecting a task coordinator doesn't mean allocating that task coordinator. A task coordinator can be coordinating any number of coordinated task sequences at any one time. Also, unlike regular task sequences, coordinated task sequences are not queued up. The task coordinator will start executing the coordinated task sequence immediately when you dispatch it, no matter how many other coordinated task sequences it is coordinating.

### insertallocatetask

The insertallocatetask command takes four parameters. The first is the task sequence. Second is the TaskExecutor or Dispatcher to give an “allocated” task to. When the task coordinator gets to an allocate task, it will actually create a separate task sequence with an “allocated” task in it, and pass that task sequence to the specified TaskExecutor or Dispatcher. In the case that it is a dispatcher, meaning you want to allocate any one of several TaskExecutors, then you can use the return value of this command as a key to reference the specific one that gets allocated, since you don't know exactly which one it is at the time that you build the task sequence. The second and third parameters are the priority and preempting values of the separate task sequence that will be created. The third parameter is optional, and specifies whether the task is blocking. By default (0), the task is blocking. If 1 is passed in, then the task will not be blocking.

### insertproxytask

The insertproxytask command is similar to the inserttask command, with one parameter, the second, added. The second parameter specifies which allocated object you want to do the task. As the task coordinator is the one actually “executing” the task sequence, once he gets to a proxy task, he will instruct the allocated object to do the task “by proxy.” Notice that for involved1 and involved2, you can either pass in a key or a straight reference to an object.

### insertsync task

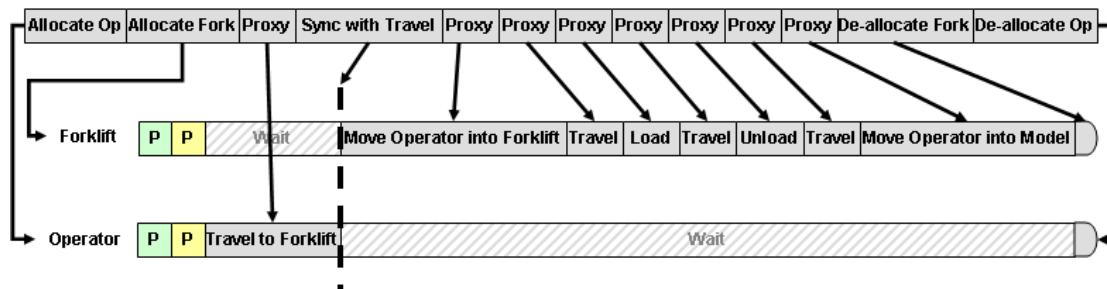
The insertsync task halts execution of the task sequence until a specified task, referenced by its key, is finished. It takes two parameters: the task sequence, and a key value of a given proxy task. It is important to note that proxy tasks which are specified for different TaskExecuters, by default, will be done in parallel, unless a sync task is specified, whereas proxy tasks given to the same TaskExecuter will automatically be done in sequential order, without the need for a sync task.

### insertdeallocatetask

The insertdeallocatetask command de-allocates a specific TaskExecuter, referenced by its key. The first parameter references the coordinated task sequence. The second parameter is the allocation key for the resource you want to de-allocate. The third parameter is optional, and specifies whether the task is blocking. By default (0), the task is blocking. If 1 is passed in, then the task will not be blocking.

The above code creates a coordinated task sequence that organizes the two task sequences, as shown in the diagram below.

### Coordinated Task Sequence



### Things to Remember

- The first thing you must do before giving any resource proxy tasks is to allocate that resource.
- You must get the key back from each allocate task, because you will use it later. The insertproxytask command takes a key for the executer of the proxy task. This is the key that the allocation task returns. You also will use this key when de-allocating the object.
- While all proxy tasks for the same allocated resource are executed in sequence, proxy tasks for different allocated resources are executed in parallel, unless you explicitly put blocking tasks in the coordinated task sequence.
- Blocking tasks are ones that block the parallel execution of the coordinated task sequence. The task coordinator goes straight through the task sequence, giving proxy tasks to the appropriate allocated resources, until a blocking task is encountered. It will then wait until that task's blocking requirement is met before continuing the task sequence. In other words, execution of all tasks occurring after that blocking task (regardless of which resource they apply to) will be stopped until the blocking task's requirement is met. The blocking tasks and their blocking requirements are as follows.

1. Allocation Task: By default this task will block until the specified resource has been allocated. However, if the fifth parameter of insertallocatetask is 1, then the allocate task will not block.
  2. Sync Task: This task will block until the proxy task specified by its key is finished.
  3. De-allocation Task: By default this task will block until the specified resource has finished all its proxy tasks and is de-allocated. However, if the third parameter of insertdeallocatetask is 1, then the de-allocate task will not block.
- The order in which you insert your tasks can have subtle yet important implications. This is especially true for placing your proxy tasks in relation to blocking tasks. Proxy tasks placed after certain blocking tasks can be executed very differently than if those proxy tasks were inserted before the blocking tasks.
  - Make sure that you de-allocate all objects that you allocate, or the task sequence won't properly release the objects it has allocated.
  - Once you have de-allocated a resource, do not give it any more proxy tasks.

**Note on non-blocking de-allocate and allocate tasks:** The functionality for allowing these tasks to be non-blocking is still in the beta state. Although we encourage you to use this feature, and there are no known bugs at the time of writing, know that you may run into some problems because this functionality hasn't yet been used extensively.

## Task Type Quick Reference

Task Type	involved1	involved2	var1	var2	var3	var4
TASKTYPE_TRAVEL	destination	NULL	end speed	forcetravel		
TASKTYPE_LOAD	item to load	station	output port			
TASKTYPE_FRLOAD	item to load	station	output port			
TASKTYPE_UNLOAD	item to unload	station	input port			
TASKTYPE_FRUNLOAD	item to unload	station	input port			
TASKTYPE_UTILIZE	involved	station	state			
TASKTYPE_DELAY	NULL	NULL	time	state		
TASKTYPE_BREAK	send message to	task sequence	check content	check receive	pv(5)* msgp(3)**	pv(6)*
TASKTYPE_CALLSUBTASKS	send message to	task sequence	pv(3)* msgp(1)**	pv(4)* msgp(2)**	pv(5)* msgp(3)**	pv(6)*
TASKTYPE_STOPREQUESTBEGIN	object to stop	NULL	state	repeat	id	priority
TASKTYPE_STOPREQUESTFINISH	object to resume	NULL	repeat	id		already executed
TASKTYPE_SENDMESSAGE	to object	from object	msgp(1)**	msgp(2)**	msgp(3)**	delay time
TASKTYPE_TRAVELTOLOC	NULL	NULL	x	y	z	end speed
TASKTYPE_TRAVELRELATIVE	NULL	NULL	x	y	z	end speed
TASKTYPE_PICKOFFSET	item	station	x	y	z	end speed
TASKTYPE_PLACEOFFSET	item	station	x	y	z	end speed
TASKTYPE_MOVEOBJECT	object to move	container	output port			
TASKTYPE_DESTROYOBJECT	object to destroy	NULL				
TASKTYPE_SETNODENUM	node to set	NULL	value	increment y/n		
TASKTYPE_TAG	user-defined	user-defined	user-defined	user-defined	user-defined	user-defined
TASKTYPE_MILESTONE	NULL	NULL	range	N/A	N/A	N/A
TASKTYPE_NODEFUNCTION	node	parnode(1)	pv(2)*	pv(3)*	pv(4)*	N/A
Non-user tasks						
TASKTYPE_TE_STOP	NULL	NULL	state			
TASKTYPE_TE_RETURN	task	task				

TASKTYPE_TE_ALLOCATED	sequence coordinator task sequence								
TASKTYPE_CT_ALLOCATE	dispatcher	allocated object	priority	preempt	front-most proxy task	blocking, (0)yes,(1)no			
TASKTYPE_CT_SYNC	NULL	NULL	key or task rank						
TASKTYPE_CT_DEALLOCATE	NULL	NULL	key or task rank	blocking, (0)yes,(1)no					

\* pv = parval

\*\* msgp = msgparam

## Task Types

You can also refer to the [task type quick reference guide](#). Task types are as follows.

### TASKTYPE\_TRAVEL

Here the TaskExecutor travels to the object specified. This is done by make a travel request to the navigator that it is connected to. If it is connected to a network, then it will make a request from the network navigator.

**Note on TaskExecutors and navigators:** Some objects by default are not connected to a navigator at all. If the TaskExecutor is not connected to a navigator, then it will do nothing for the travel task. The following objects do not connect to any navigators by default: ASRSvehicle, Elevator, Robot.

<b>involved1:</b>	The object to travel to.
<b>involved2:</b>	Not used. Use NULL for this parameter.
<b>var1:</b>	This specifies the desired end speed for the travel operation. If 0, then the desired end speed will be the maximum speed of the TaskExecutor. If -1, then the desired end speed will be 0. Otherwise, the desired end speed will be the value itself.
<b>var2</b>	If this value is 1, then the object will travel to the destination node even he is already connected to it.
<b>var3</b> - <b>var4:</b>	N/A

### TASKTYPE\_LOAD, TASKTYPE\_FRLOAD

This task causes the TaskExecutor to load an item from a station. If the TaskExecutor's "Travel Offsets for Load/Unload Tasks" checkbox is checked in its parameters page, then it will travel to the location of the given flow item by querying the location from the station and using offset travel. Then the TaskExecutor will figure out the load time. At the end of the load time, the TaskExecutor will move the item into itself. For Frload, it will notify the FixedResource right before it moves the item, so that the FixedResource can update its own tracking data. Refer to the FixedResource for more information on when to use Frload and when to use Load

<b>involved1:</b>	the object to load (usually a flow item).
<b>involved2:</b>	the object to load from (FixedResource is assumed if the tasktype is specified as FRLOAD).
<b>var1:</b>	this is the output port through which the object will exit the station. Usually a 0 is fine.
<b>var2</b> - <b>var4:</b>	N/A

## **TASKTYPE\_UNLOAD, TASKTYPE\_FRUNLOAD**

This task causes the TaskExecutor to unload an item to a station. If the TaskExecutor's "Travel Offsets for Load/Unload Tasks" checkbox is checked in its parameters page, then it will travel to a drop-off location by querying the station and using offset travel. Then the TaskExecutor will figure out the unload time. At the end of the unload time, the TaskExecutor will move the item into the station. For Frload, it will notify the FixedResource right before it moves the item, so that the FixedResource can update its own tracking data. Refer to the FixedResource for more information on when to use Frunload and when to use Unload.

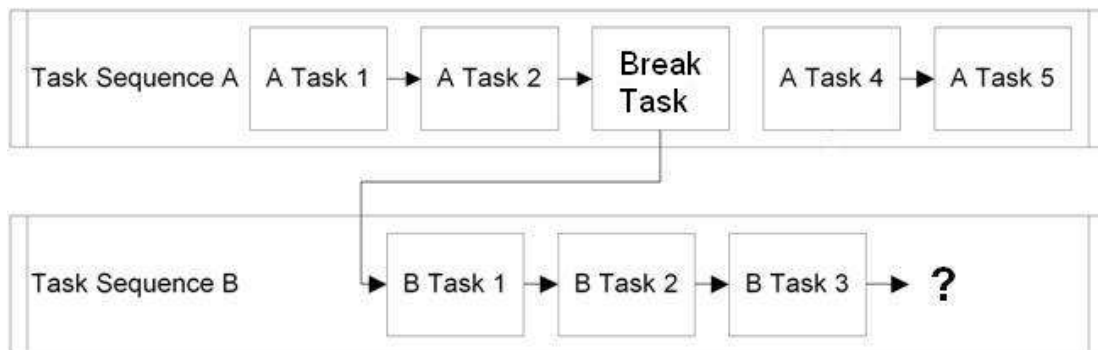
<b>involved1</b>	the object to unload (usually a flow item).
<b>involved2</b>	the object to unload to (FixedResource is assumed if the tasktype is specified as FRLOAD).
<b>var1</b>	this is the input port through which the object will enter the station. Usually a 0 is fine, unless you are unloading to a Combiner, which needs to know the input port an item enters through in order to



	update its input table.
<b>var2</b> - <b>var4</b>	N/A

### TASKTYPE\_BREAK

This task causes the TaskExecutor to "break" from its currently active task sequence to a new task sequence as the diagram below illustrates.



The involved objects and variables allow you to customize how to find the task sequence to break to. In the default case, the TaskExecutor will call its "Break To Requirement" function. This function should return a reference to the task sequence that you want the TaskExecutor to break to. In your break logic you may search through the TaskExecutor's task sequence queue by using task sequence query commands, or you can create the task sequence explicitly using `createemptytasksequence`. If you don't want the TaskExecutor to break at all, then return `NULL`.

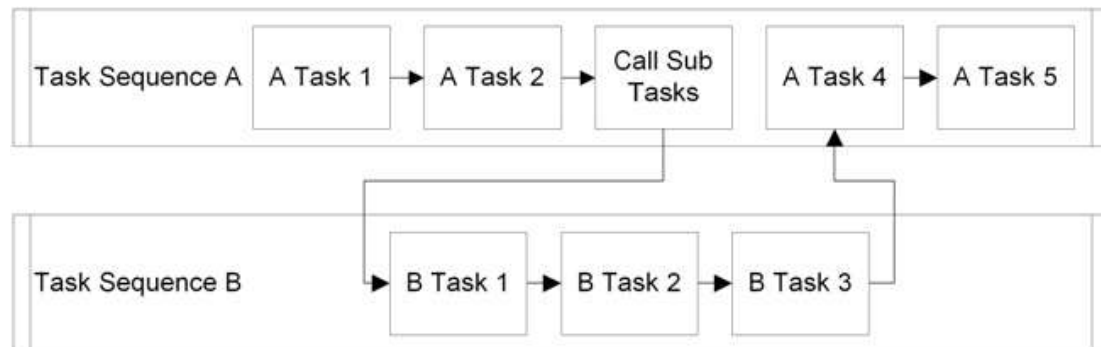
<b>involved1</b>	<p>If involved1 is specified, then it should be a reference to some object in the model. Instead of calling the "Break To Requirement" the TaskExecuter will send a message to this object. The only difference here is the place in which you place your logic for finding a task sequence to break to. By default, the logic executes in the Break To Requirement, but if this parameter is specified, then you will write your logic in a message trigger. Again, the return value of the message should be a reference to the task sequence. You would most likely use this feature if you want to centralize your logic through messages to a central "Model Control Center."</p> <p><b>Note on the return value:</b> In the message you will need to cast the reference to the task sequence into a number, because message triggers return a number type. You can do this by using the <code>tonum()</code> command: <code>tonum(mytasksequence)</code></p>
<b>involved2</b>	<p>If involved2 is specified, then it is interpreted as a straight reference to the task sequence that the TaskExecuter should break to. This would only be used if you know exactly which task sequence you want to break to at the time that you create the task sequence with the break in it. This parameter is not used very often, because if you know exactly which task sequence to break to when you create a task sequence, then you can just add the tasks into the original task sequence when you create it.</p> <p><b>Note on using both involved parameters:</b> If the involved1 parameter of this task is specified, then involved2 should be NULL. Likewise, if involved2 is specified, then involved1 should be NULL. These parameters are mutually exclusive. You can also just use the default case by specifying both of the involved parameters as NULL.</p>
<b>var1</b>	<p>This parameter specifies whether or not the content of the TaskExecuter should be screened before performing the break task. By default (0), the TaskExecuter will only perform a break task if its current content is less than the maximum content specified in its parameters page. If var1 is not 0, however, then the TaskExecuter will ignore its current content, and perform the break task anyway. This parameter is also passed in as <code>parval(3)</code> if it is to call its Break To Requirement function, and as <code>msgparam(1)</code> if it is to send a message.</p>

<b>var2</b>	This parameter specifies whether or not the TaskExecutor should check to receive task sequences from an upstream Dispatcher. By default (0), the TaskExecutor will see if it has any task sequences in its queue. If the queue is empty, or if all of the task sequences in its queue are task sequences that have already been start and broken out of, then it will open its input ports and receive a task sequence from an upstream dispatcher. However, if var2 is not zero, then the TaskExecutor will not receive anything from an upstream dispatcher before calling its break logic. This parameter is also passed in as parval(4) if it is to call its Break To Requirement function, and as msgparam(2) if it is to send a message.
<b>var3- var4</b>	These parameters are passed into the Break To Requirement as parval(5) and parval(6), and var 3 is passed into the message as msgparam(3) if the task is to send a message.

---

## TASKTYPE\_CALLSUBTASKS

This task is just like the break task, except that it ensures that as soon as the second task sequence is finished, it will return immediately to the next task of the original task sequence. The following illustration shows how this works.



As the diagram shows, Task Sequence A comes to a call sub tasks type, upon which it breaks to Task Sequence B. Immediately after Task Sequence B is finished, it returns to the next task of Task Sequence A. Often Task Sequence B won't be created until Task Sequence A actually gets to the call sub tasks task. This is because often when you create Task Sequence A, you don't know exactly what you want the TaskExecutor to do when he gets to the point of the call sub tasks task, or you don't have an up-front reference to the objects you need. For example, what if you want the TaskExecutor to travel to one part of the model, then load an item, then travel to another part of the model and unload the item, but you don't have a reference to the item that you want to load. You want the TaskExecutor to travel to that portion of the model, then figure out which item to load. Here you would use call sub tasks so that you can resolve the reference to the item at the time the TaskExecutor arrives at the load location.

Call sub tasks can also be heirarchical. This means that Task Sequence B can also have a call sub tasks type in it.

If you decide to create the task sequence when the TaskExecutor gets to the call sub tasks task, then you will need to create the task sequence using `createemptytasksequence()`, and insert tasks using `inserttask()`, but do not dispatch the task sequence using `dispatchtasksequence()`. Simply return the reference to the task sequence.

**Note on the Break To Requirement function:** When the TaskExecutor comes to this task type, by default, he will call his "Break To Requirement" function. He will pass in a 1 as `parval(2)`, so that within the function you can tell that it is a call sub tasks instead of the usual break task.

**Note on Coordinated Task Sequences:** Using the diagram above, if Task Sequence B is a coordinated task sequence, then the TaskExecutor that executes the call sub tasks task from Task Sequence A must be the first object to be allocated in the Task Sequence B.

<b>involved1</b>	<p>If involved1 is specified, then it should be a reference to some object in the model. Instead of calling the "Break To Requirement" the TaskExecuter will send a message from itself to the object specified by the involved1 parameter. The only difference here is the place in which you put your logic for finding a task sequence to break to. By default, the logic executes in the Break To Requirement, but if this parameter is specified, then you will write your logic in a message trigger. Again, the return value of the message should be a reference to the task sequence. You would most likely use this feature if you want to centralize your logic through messages to a central "Model Control Center."</p> <p><b>Note on the return value:</b> In the message you will need to cast the reference to the task sequence into a number, because a message trigger returns a number type. You can do this by using the tonum() command: <code>tonum(mytasksequence)</code></p>
<b>involved2</b>	<p>If involved2 is specified, then it is interpreted as a straight reference to the task sequence that the TaskExecuter should break to. This would only be used if you know exactly which task sequence you want to break to at the time that you create the task sequence with the break in it. This parameter is not used very often, because if you know exactly which task sequence to break to when you create the original task sequence, then you should just add the tasks into the task sequence when you create it. It does, however, allow you to specify different priority and preempting values for different portions of your task sequence, so that if you don't want a certain portion of your task sequence to be preempted, then you can have that portion be a sub-routine task sequence with a different priority than the original task sequence.</p> <p><b>Note on using both involved parameters:</b> If the involved1 parameter of this task is specified, then involved2 should be NULL. Likewise, if involved2 is specified, then involved1 should be NULL. These parameters are mutually exclusive. You can also just use the default case by specifying both of the involved parameters as NULL.</p>
<b>var1- var4</b>	<p>These parameters are passed into the Break To Requirement as parval(3), parval(4), parval(5) and parval(6), and var1, var2, and var3 are passed into the message as msgparam(1), msgparam(2) and msgparam(3) respectively if the task is to send a message.</p>

**TASKTYPE\_UTILIZE**

This task causes the TaskExecutor to go into a given state, and then wait until it is freed from that state with the freeoperators() command. This task is used frequently when you want an operator to "do something" at a station, but at the time you create the task sequence you don't know how long it will take to finish whatever the operator is doing. In such a case, use this task type to cause the operator to go into the state you specify, and then free him when he is finished, using the freeoperators() command. This can be done from a trigger like OnProcessFinish or OnSetupFinish, etc. If you know from the outset how long the operator will have to be "doing something", then you can use the **delay task** instead.

<b>involved1</b>	<p>Often this parameter will be a reference to a flow item, if the operator's job has to do with processing a flow item. Sometimes it references a station, for example in the case that a station goes down, and an operator is called. Here, the operator is working on the station, and not a flow item, so the station would be the involved1 parameter. You can even specify this parameter to be NULL if you like.</p> <p>In more specific terms, this parameter is a key for matching with the freeoperators command. For example, if this parameter is a flowitem, then when the freeoperators command is called, the same flowitem must be passed into the second parameter of the freeoperators command in order for the operator to be freed properly.</p> <p>Often you will use a team of operators, any one of which can do the job you want. In such a case you would give the task sequence to a dispatcher, and the dispatcher would give it to a member of the team. At the time you call freeoperators, you really don't know exactly which operator finally came and worked on your job, so you send the freeoperators command to the dispatcher, and in the freeoperators command, you make the second parameter match the involved1 parameter that you specified for this task. This allows the dispatcher to basically say to his team, "Any of you who are doing a Utilize task whose involved1 parameter is <i>this</i> can now finish that task". This makes it so that the dispatcher can free certain operators from the right tasks without freeing other operators from the wrong tasks.</p>
------------------	---

<b>involved2</b>	<p>This parameter only needs to be specified if it is possible for the operator to be preempted away from his operation. An operator can be preempted away from an operation by a preempting task sequence, or by a stopobject() command, or by a global TimeTable or global MTBF table. If the operator is preempted away from a utilize task, then problems can be caused if the freeoperators command is called before he comes back to the utilize task. If freeoperators is called while he is doing something else, then the operator will simply ignore it, thinking it doesn't apply to him. Then, once he comes back to the operation, he will never be freed because the modeling logic thinks that he's already been freed. This involved2 parameter can be used to help alleviate this problem. If involved2 is specified, then it should point to an object in the model that is responsible for freeing the operator. When the operator is preempted, he will call stopobject() on the specified object, which stop the object, and in most cases thus stop the object from calling freeoperators. Once the operator comes back to the utilize task, he will call resumeobject() on the station, and things will resume as normal, and the operator will eventually be freed. If you would like to know more about preempting, refer to its corresponding help section.</p>
------------------	---

<b>var1</b>	This is the state into which the operator will go during the utilize task. If it is 0, then the TaskExecutor will go into STATE_UTILIZE.
<b>var2</b> - <b>var4</b>	N/A

### TASKTYPE\_STOPREQUESTBEGIN

This task causes the TaskExecutor to call stopobject() on the involved1 object. Refer to the stopobject() command documentation for more information.

<b>involved1</b>	This parameter specifies the object to call stopobject() on. If NULL, then the TaskExecutor will call stopobject on himself.
<b>involved2</b>	Not used. Use NULL for this parameter.

<b>var1</b>	This is the state to request the stopped object to go into.
<b>var2</b>	This variable is necessary only if the TaskExecutor that executes this task may be preempted. It is also only needed if you are using milestone tasks. If this variable is set to 0, then the task will only be executed once, even if it is within a milestone task's range and the object is preempted within that range. If this variable is set to 1 and the task is within a milestone task's range, then the task will be executed again each time the object is preempted and needs to do the task over again. By default, the value is 0, meaning the task will only be executed once. Note that if it is 0, on the first execution of the command, the TaskExecutor will change the variable to 2 as a flag to not execute it again.
<b>var3</b>	This is the id for the stopobject command
<b>var4</b>	This is the priority of the stopobject command

## **TASKTYPE\_STOPREQUESTFINISH**

This task causes the TaskExecutor to call resumeobject() on the involved1 object. Refer to the resumeobject() command documentation for more information.

<b>involved1</b>	This parameter specifies the object to call resumeobject() on.
<b>involved2</b>	N/A
<b>var1</b>	This variable is necessary only if the TaskExecutor that executes this task may be preempted. It is also only needed if you are using milestone tasks. If this variable is set to 0, then the task will only be executed once, even if it is within a milestone task's range and the object is preempted within that range. If this variable is set to 1 and the task is within a milestone task's range, then the task will be executed again each time the object is preempted and needs to do the task over again. By default, the value is 0, meaning the task will only be executed once.
<b>var2</b>	This is the id for the resumeobject() command.
<b>var3</b>	N/A



<b>var4</b>	This variable is managed by the TaskExecutor, and tells whether this task has already been executed once.
-------------	---

## TASKTYPE\_SENDMESSAGE

This task causes the TaskExecutor to send a message to the involved1 object.

<b>involved1</b>	Involved1 is the object that the message is sent to. If NULL, then a message is sent to the TaskExecutor himself.
<b>involved2</b>	Involved2 specifies msgsendingobject in the message trigger. If NULL, then msgsendingobject is the TaskExecutor himself. Usually this will be NULL, because it is the only way that you can access the TaskExecutor within the message trigger. However, you may want the message to be sent "from" a different object, so you have the option here.
<b>var1</b>	This parameter is passed in as msgparam(1) in the message trigger.
<b>var2</b>	This parameter is passed in as msgparam(2) in the message trigger.
<b>var3</b>	This parameter is passed in as msgparam(3) in the message trigger.
<b>var4</b>	<p>This parameter tells whether the message sent is to be a delayed message. If 0, then the message is sent immediately. If -1, then the message is sent delayed in zero time. Otherwise, the message is sent in the specified number of seconds.</p> <p>You might think that delayed message sending is a bit redundant, because if you want to send a delayed message, why not insert a delay task followed by a regular send message task. There is a subtle difference. Say, for example, you want the TaskExecutor to wait until a certain number of requirements are met, and the only way you can check those requirements is by executing code. The way that you would do this is, when the TaskExecutor gets to the point where he needs to wait for the requirements to be met, he sends a message to some object, and then either does a utilize task, or a stop request begin task. When the other object gets the message, he is responsible for checking if the requirements are met. If they are already met, then he is to immediately call</p>

resumeobject() or freeoperators() on the TaskExecuter. Otherwise he must wait until the requirements are met, and then call resumeobject() or freeoperators(). A problem arises, however, when the requirements are already met and he can immediately allow the TaskExecuter to continue. If the message has been sent immediately, then the TaskExecuter hasn't started the utilize of stoprequestbegin task yet. He is still working on the send message task. So the other object can't immediately call freeoperators() or resumeobject() because he must wait until the TaskExecuter finishes the send message task, and goes on to the utilize or stop request begin. Sending a delayed message in 0 time allows the TaskExecuter to do exactly that, and thus allow the other object to immediately free him if the requirements are met.

## TASKTYPE\_DELAY

This task causes the TaskExecuter to go into a given state, and then simply wait for a specified amount of time.

<b>involved1</b>	Not used. Use NULL for this parameter.
<b>involved2</b>	Not used. Use NULL for this parameter.
<b>var1</b>	This is the amount of time that the TaskExecuter will wait in the specified state.
<b>var2</b>	This is the state into which the operator will go during the delay task. If it is 0, then the TaskExecuter will go into STATE_UTILIZE.
<b>var3</b>	This variable is reserved by the TaskExecuter. Do not set this variable yourself, or at least don't expect it to stay the same as what you specified it to be.
<b>var4</b>	N/A

---

**TASKTYPE\_MOVEOBJECT**

This task tells the TaskExecutor to move a specified object into a specified container. This would be used if you want the TaskExecutor to load/unload a flow item without going through the offset travel or the load/unload time. Also, this could be used if you want a flow item to be moved, but not into or out of the TaskExecutor.

<b>involved1</b>	The object to move.
<b>involved2</b>	The object to move involved1 into.
<b>var1</b>	The output port of the object that involved1 will exit. 0 is usually fine.
<b>var2-var4</b>	N/A

---

**TASKTYPE\_DESTROYOBJECT**

This task tells the TaskExecutor to destroy the specified object. Usually this will be done if a flow item is finished in a model, and is ready to go to a sink. You can destroy the flow item explicitly here. You could also use this to destroy labels. Say for example you have a label that acts as a queue of requests. Once a request has been completed, or is ready to be taken out of the queue, you can destroy it.

<b>involved1</b>	The object to destroy.
<b>involved2</b>	Not used. Use NULL for this parameter.
<b>var1-var4</b>	N/A.

---

**TASKTYPE\_SETNODENUM**

This task causes the TaskExecutor to set the value on a specified node. This would be used if you want to set a variable or label on the object.

<b>involved1</b>	The node to set the value on. This can be something like label(current, "mylabel"). or var_s(current, "maxcontent")
<b>involved2</b>	Not used. Use NULL for this parameter.
<b>var1</b>	The value to set the node to.
<b>var2</b>	This parameter allows you to either set the value on the node, or increment the value on the node. By default (0), it will set the value of the node. If 1, then it will increment the value on the node by var1.
<b>var3- var4</b>	N/A.

**TASKTYPE\_TRAVELTOLOC**

This task causes the TaskExecutor to travel to a specified location using offset travel.

<b>involved1</b>	Not used. Use NULL for this parameter.
<b>involved2</b>	Not used. Use NULL for this parameter.
<b>var1</b>	This is the x location to travel to.
<b>var2</b>	This is the y location to travel to.
<b>var3</b>	This is the z location to travel to.
<b>var4</b>	This is the desired end speed.

**TASKTYPE\_TRAVELRELATIVE**

This task causes the TaskExecuter to travel a specified offset using offset travel. This is like the TravelToLoc task, except that instead of traveling to a location, the TaskExecuter offsets from his current location.

<b>involved1</b>	Not used. Use NULL for this parameter.
<b>involved2</b>	Not used. Use NULL for this parameter.
<b>var1</b>	This is the x offset to travel.
<b>var2</b>	This is the y offset to travel.
<b>var3</b>	This is the z offset to travel.
<b>var4</b>	This is the desired end speed.

**TASKTYPE\_PICKOFFSET**

This task causes the TaskExecuter to execute part or all of the travel offset involved in a load task. This also allows you to sequence the travel operation that the TaskExecuter does before doing a load. Let's say, for example, that you have a floor storage area that the TaskExecuter is going to pick an item from. The items are organized in bays (x) and rows (y) on the floor. When the TaskExecuter arrives at the floor storage area, instead of traveling straight to the product to load it, you want him to first travel in the x direction to the right bay, then travel the y and z offsets to the location of the item. This task type allows you to do this. When the TaskExecuter arrives at the floor storage area, you can give him a pick offset task in which you tell to only travel the x portion of the offset. Then you can give him the usual load task, and he will do the y and z offsets once he's finished with the x offset. If you give an object a pick offset task to travel all of the offsets, the effect will be to do the complete travel operation of a load task, without actually loading the object at the end.

<b>involved1</b>	Just like in a load task, this is the reference to the item that would be loaded.
<b>involved2</b>	Just like in a load task, this is the reference to the station from which the item would be loaded.

<b>var1- var3</b>	These parameters are usually either a 0 or 1. They correspond respectively to the x, y, and z portions of the offset travel. If 0, then the TaskExecutor will travel none of the corresponding offset. If 1, then the TaskExecutor will travel all of the corresponding offset. You can also have these values be between 0 and 1. A 0.9 would mean that the TaskExecutor would travel 90% of the corresponding offset.
<b>var4</b>	This is the desired end speed.

---

### TASKTYPE\_PLACEOFFSET

This task is just like the pick offset task, except that it does part of all of the offset travel involved with an unload task.

<b>involved1</b>	Just like in an unload task, this is the reference to the item that would be loaded.
<b>involved2</b>	Just like in an unload task, this is the reference to the station to which the item would be unloaded.
<b>var1- var4</b>	Same as pick offset task.

---

### TASKTYPE\_TAG

This task is exclusively for you to use to "tag" your task sequences. Say for example, that you create 5 general types of task sequences in your model. At certain points in the simulation you need to know which general type a certain task sequence is. By inserting a "tag" task as the first task of all task sequences you create, you can then query that task by using the `gettaskinvolved()` and `gettaskvariable()` commands.

<b>involved1</b>	For your use.
<b>involved2</b>	For your use.

<b>var1- var4</b>	For your use
-----------------------	--------------

## TASKTYPE\_TRAVELRELATIVE

This task causes the TaskExecutor to travel a specified offset using offset travel. This is like the TravelToLoc task, except that instead of traveling to a location, the TaskExecutor offsets from his current location.

<b>involved1</b>	Not used. Use NULL for this parameter.
<b>involved2</b>	Not used. Use NULL for this parameter.
<b>var1</b>	This is the x offset to travel.
<b>var2</b>	This is the y offset to travel.
<b>var3</b>	This is the z offset to travel.
<b>var4</b>	This is the desired end speed.

## TASKTYPE\_MILESTONE

This task type is only useful for task sequences that may be preempted. It defines a "bookmark" in the task sequence that the TaskExecutor can revert back to if it is preempted away from the task sequence. Normally when a TaskExecutor is preempted away from a task sequence, it will resume at the same spot it was at once it comes back to the task sequence. The milestone task allows you to tell the TaskExecutor to repeat a whole section of tasks if preemption occurs. The task has a defined range of subsequent tasks for which it is responsible. If the TaskExecutor is within that range and is preempted, then it will revert back to the milestone task. If it has passed the milestone's range, then it will go back to the default preemption functionality.

**Note on coordinated task sequences:** The milestone task will not work as a proxy task in a coordinated task sequence. If you want to set bookmarks in a coordinated task sequence, then you should insert a CALLSUBTASKS proxy task,

and within the subsequent sub-task sequence, you can insert milestone tasks as needed.

<b>involved1</b>	Not used. Use NULL for this parameter.
<b>involved2</b>	Not used. Use NULL for this parameter.
<b>var1</b>	This parameter is the range of the milestone task, defined in number of tasks. For example, if var1 is set to 3, and the milestone task is the 5th task in the task sequence, then if the TaskExecutor is preempted while executing any one of tasks 6, 7 or 8, then it will revert back to the milestone task.
<b>var2- var4</b>	N/A

### TASKTYPE\_NODEFUNCTION

This task type will call nodefunction() on the specified node.

<b>involved1</b>	The node to call nodefunction() on.
<b>involved2</b>	Not used. Use NULL for this parameter.
<b>var1- var3</b>	These parameters are passed in as parval(1), parval(2), and parval(3) in the nodefunction.
<b>var4</b>	N/A

**Note:** The following task types are only for reference purposes, you should never insert one of these task types explicitly.



**TASKTYPE\_TE\_STOP**

This task is created when you call stopobject on a TaskExecuter. The TaskExecuter creates a preempting task sequence of priority 100000, and inserts this stop task into it.

<b>involved1</b>	Not used. Use NULL for this parameter.
<b>involved2</b>	Not used. Use NULL for this parameter.
<b>var1</b>	The state that the TaskExecuter should go into when he is down.
<b>var2- var4</b>	N/A

**TASKTYPE\_TE\_RETURN**

This task is added onto the end of a task sequence that is returned by a call sub tasks task. It ensures that once the task sequence is finished, it will return to the original task sequence

<b>involved1</b>	This parameter points to the original task sequence to return to. If the original task sequence is a coordinated task sequence, then this will point to the task sequence with the TE_ALLOCATED task in it.
<b>involved2</b>	This parameter points to the actual call sub tasks task as a node.
<b>var1- var4</b>	N/A

**TASKTYPE\_TE\_ALLOCATED**

This is a special task specifically used for coordinated task sequences. The task tells the TaskExecuter to be allocated meaning when the object comes to this task, it will notify the object coordinating the task sequence, and then simply wait

until it is told to do something by that coordinator.

<b>involved1</b>	The object coordinating the task sequence.
<b>involved2</b>	This is a reference to the coordinated task sequence being executed.
<b>var1- var4</b>	N/A

## Coordinated Task Types

**Note:** The following task types should never be inserted explicitly by the user. They should instead be inserted using the `insertallocatetask()`, `insertsynctask()`, and `insertdeallocatetask()` commands.

### TASKTYPE\_CT\_ALLOCATE

Here the task coordinator will try to allocate some TaskExecutor. It is by done by creating a regular task sequence with one TASKTYPE\_TE\_ALLOCATED task in it, and giving the task sequence to a specified object. This task blocks the continuation of the task sequence until an object has been allocated.

<b>involved1</b>	This is a reference to a dispatcher to give the task sequence to. It may be a specific TaskExecutor, or, if the object to allocate can one of several possible objects, then it can reference a dispatcher that dispatches to those task executors.
<b>involved2</b>	This is not specified when the task is created, but will be set once the object has been allocated and will reference that object.
<b>var1</b>	The priority value for the allocation
<b>var2</b>	The preempt value for the allocation
<b>var3</b>	This is changed as the task sequence is executed, and is the rank of the front most proxy task that has been given to the allocated resource
<b>var4</b>	This tells whether the task sequence is blocking. Default (0) is blocking, 1 is non-blocking

**TASKTYPE\_CT\_SYNC**

Here the task coordinator blocks the continuation of the task sequence until some previously specified task (referenced by rank), is finished.

<b>involved1</b>	Not used. Use NULL for this parameter.
<b>involved2</b>	Not used. Use NULL for this parameter.
<b>var1</b>	The rank of the task to sync to.
<b>var2- var4</b>	N/A

**TASKTYPE\_CT\_DEALLOCATE**

Here the task coordinator notifies an object that it can finished its allocated task, and resume to other tasks. The object is specified by the rank of the allocation task that allocated it.

<b>involved1</b>	Not used. Use NULL for this parameter.
<b>involved2</b>	Not used. Use NULL for this parameter.
<b>var1</b>	The rank of the allocation task that allocated the object.
<b>var2</b>	This variable specifies whether the deallocation task is blocking. Default (0) is blocking. 1 is non-blocking.
<b>var3- var4</b>	N/A



# Charting and Reporting

## Charting and Reporting Overview

### Overview

As Flexsim runs, it automatically collects basic statistical information about the objects in the model. This information includes flowitem input and output, times in states, etc. There is a large amount of data that is not collected this way however. For example, the number of flowitems that entered a Processor that all were itemtype 1 cannot be found using Flexsim's basic statistics features. Flexsim can be configured to collect more detailed statistics that can be viewed and analyzed after the model run is complete. These statistics are analyzed using an application called Flexsim Chart. This can be run either from the Start menu, if the user wishes to analyze data that has already been collected, or from within Flexsim, if the user wishes to analyze new data from a model run.

To use Flexsim Chart, the user must first enable Full History before running the model. This is done by selecting Statistics | Full History On from the main menu. Depending on the size of the model and the length of the run, Full History can require a large amount of memory. For this reason, it is turned off by default. Once Full History is enabled, the user can run the model. Once the model has run to completion, or the user stops it, the collected data can be saved to a database file and opened with Flexsim Chart. This is done by selecting Statistics | Reports and Statistics from the main menu.

### Collected Data

There are two main tables of data that are collected by Flexsim during a model run. The first is the Movement Table. This table records every time a flowitem moves from one object to another. Each flowitem is assigned a unique numerical identifier which is used in this table. The table records which object the flowitem is leaving, which object it is entering, what time the movement is taking place and the itemtype of the flowitem. The second table is the State Table. This table records every time an object changes state. The table records the object that is changing state, the time of the change and the state that the object is changing to. Using the data from these two tables, Flexsim Chart can be used to put together detailed charts, graphs and reports about the model that was running.

There are several other, less important tables that are saved as well. If the user wishes to export the Summary Report or State Report, there are tables in the database for these two reports. If the user has created any object groups in their model, there are tables that define the groups and the objects in those groups. Generally, the user does not need to concern themselves with the tables that are created in the database or the specific data in those tables.

The data is all saved in a single .mdb file, which is opened by Flexsim Chart. This is a standard Microsoft Access Database file. Generally, there is no need for users to open these database files with Access. The reports and graphs that they should need are available in Flexsim Chart.

## File types

There are two types of files that Flexsim Chart interacts with.

**Full History database files (.mdb):** These are Microsoft Access database files. They store the information collected by Flexsim when Full History is enabled. One of these files will be created whenever the user generates a Full Report using the Reports and Statistics GUI. While it is possible to open these files with Microsoft Access, there should be very little need to do so. Flexsim Chart will read the data from these files and generate charts and reports from that data.

**Flexsim Chart files (.fsc):** These are saved Flexsim Chart sessions. They store a reference to the database (.mdb file) that they were created with. They also store a list of the charts that have been generated, the tabs that are visible, and any customizations the user has done to the charts and graphs that were generated. These files are used to return to an analysis of a model run without having to rerun the model. An .fsc file can be distributed to other Flexsim users, but the .mdb file that it is associated to must also be distributed.

When Flexsim Chart is run from the Start menu, the user is given a list of .mdb files as well as .fsc files that they can open. If they choose to open an .mdb file, it will be opened and there will be no tabs or charts already created. If they choose to open an .fsc file, first the corresponding database file is opened and then the charts and tabs that were created when the .fsc file was saved will become visible. If Flexsim Chart is unable to open the .fsc file correctly for any reason, it will still attempt to open the database file. This gives the same result as if the user selected the .mdb file initially.

## Flexsim Chart

### The Main Window

When the user opens Flexsim Chart using the Reports and Statistics GUI, the newly-generated database file is automatically opened by Flexsim Chart. If the user opens Flexsim Chart from the Start menu, they are prompted to select a file to open. Once the file is open the user sees the main Flexsim Chart window (see Figure 1).

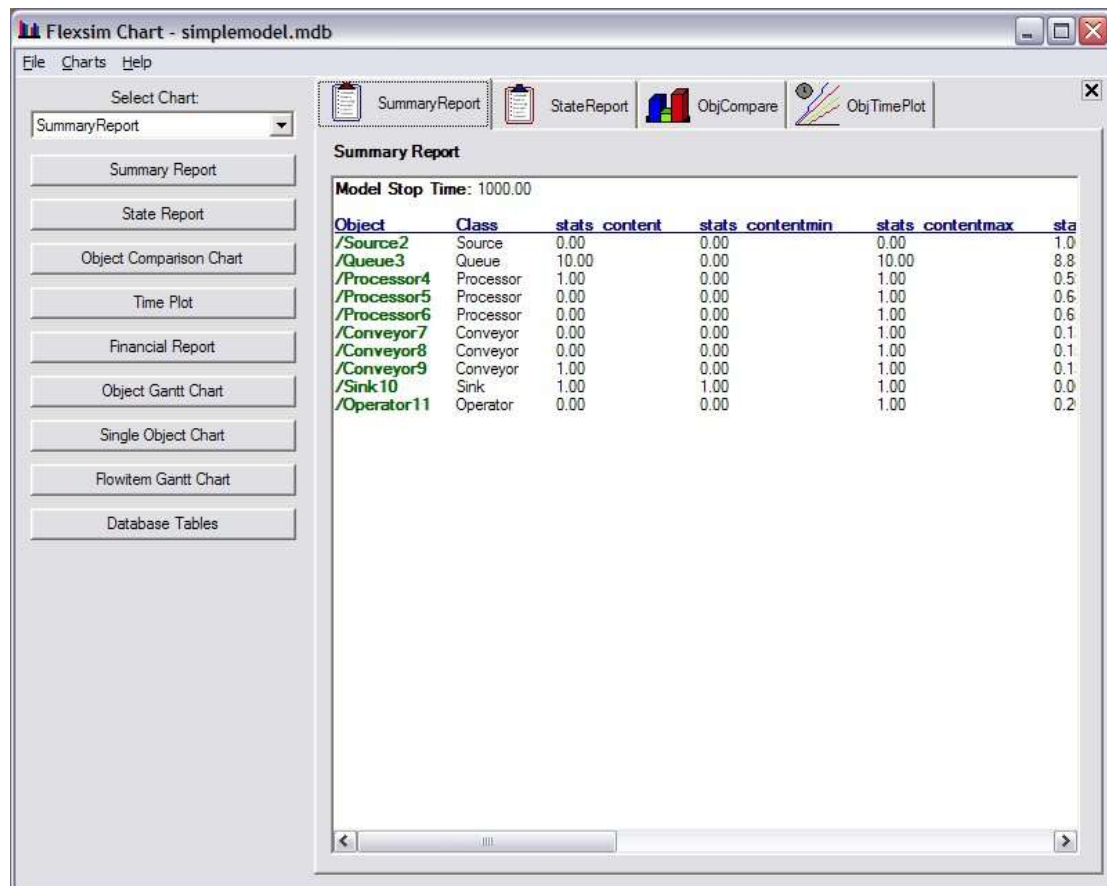



Figure 1 - The Flexsim Chart main window

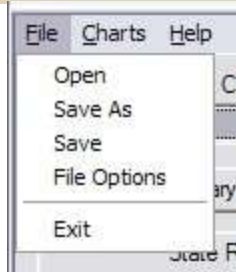
The main window has three main sections:

**The Select Chart drop-down list:** This list is used to jump to a specific chart that has been generated in the main window. If the tab for that chart is currently open, that tab will become the active tab. If there is no tab for the selected chart, a tab will be created for it at the right-hand side of the tab list.

**Chart Creation buttons:** Each of these buttons creates a new tab that contains a specific type of chart or report. There is no limit on the number of charts or reports that can be created, and multiple instances of the same type of report can be created. Each instance can be configured to display a different result from the model run.

**Chart Tab list:** Each of the tabs in the list is a single chart or report. There are nine different types of tabs that can be created. Each type has a different icon in the tab list. The names on the tabs are the names that the user gave to the charts. There are default names assigned when tabs are created, but it is recommended that the user change these names. If the user feels that there are too many tabs on the screen, they can use the  button to hide the current tab. They are then asked if they wish to remove the chart from memory. If they do wish to remove it from memory, it will no longer be visible in the Select Chart drop-down list and all changes made to that chart will be lost. If the user wishes to keep the chart in memory, it will still be visible in the Select Chart drop-down list, but will no longer be visible in the tab list. Any changes made to the chart will not be lost in this case.

## File Menu



**Open:** This prompts the user for a Flexsim Chart (.fsc) file to open.

**Save As:** This allows the user to save the current charts and tabs as a new Flexsim Chart (.fsc) file. The user will first be prompted for which charts should have their custom formatting saved. If custom formatting is saved, the resulting file is be much larger than if the formatting is not saved.

**Save:** This saves the current charts and tabs as the last saved Flexsim Chart (.fsc) file.

**File Options:** This prompts the user for the charts that should have custom formatting saved. If custom formatting is saved, the resulting file is much larger than if the formatting is not saved.


**Exit:** This closes Flexsim Chart.


## Charts Menu



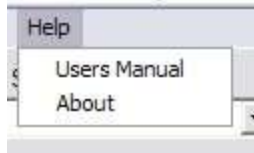
**Open All:** This creates a tab for all of the charts that have been created. Charts that already have a tab and are open will not be reopened or moved in the tab list.



**Hide All:** This hides all of the tabs that are currently visible. The charts are not removed from memory. They can still be accessed from the Select Chart drop-down list. This is the same as clicking on the  button for each tab and choosing to keep the tabs in memory.

**Close All:** This closes all of the tabs that are currently visible and removes them from memory. It does not affect any charts that are not in the tab list. This is the same as clicking on the  button for each tab and choosing to remove the tabs from memory.

### Help Menu



**Users Manual:** This opens the on-line User's Manual so the user can find detailed information about Flexsim Chart.

**About:** This opens the About Flexsim Chart dialog box.

## Summary Report

### Overview

The Summary Report tab displays the same information that is displayed by Flexsim's standard Summary Report. Each row in the table is a single object in the model. The first column is the object's name and the second is the object's class. All the other columns are variables, attributes and labels that can be selected by the user. The Summary Report can only display 32 columns. If the table has more than that, only the first 32 will be visible. The rest can be seen by exporting the Summary Report to Excel from within Flexsim.

### Tab Description

Object	Class	stats content	stats contentmin	stats contentmax	sta
/Source2	Source	0.00	0.00	0.00	1.0
/Queue3	Queue	10.00	0.00	10.00	8.8
/Processor4	Processor	1.00	0.00	1.00	0.5
/Processor5	Processor	0.00	0.00	1.00	0.6
/Processor6	Processor	0.00	0.00	1.00	0.6
/Conveyor7	Conveyor	0.00	0.00	1.00	0.1
/Conveyor8	Conveyor	0.00	0.00	1.00	0.1
/Conveyor9	Conveyor	1.00	0.00	1.00	0.1
/Sink10	Sink	1.00	1.00	1.00	0.0
/Operator11	Operator	0.00	0.00	1.00	0.2

Figure 1 - A Summary Report tab

The Summary Report tab does not allow the user to edit anything, but they can see the data.

## State Report

### Overview

The State Report tab displays the same information as Flexsim standard State Report. Each row in the table represents a single object in the model. The first column contains the path in the model to the object and the second is the object's class. Each of the other columns is a state that the object could have been in during the run. The values displayed are either the actual amount of time spent in each state or the percentage of the model run time that was spent in each state. The user decides with the Reports and Statistics GUI which way the values will be displayed. The State Report tab can only display up to 32 columns. If there are more than 32 columns in the state table, only the first 32 will be visible. The full table can be seen by exporting the State Report to Excel from within Flexsim.

### Tab Description

Object	Class	idle	processing	busy	blocked	generating	empty	colle
/Source2	Source	0.00%	0.00%	0.00%	52.20%	47.80%	0.00%	0.00%
/Queue3	Queue	0.00%	0.00%	0.00%	0.00%	0.00%	6.17%	0.00%
/Processor4	Processor	40.48%	31.94%	0.00%	0.00%	0.00%	0.00%	0.00%
/Processor5	Processor	41.04%	34.02%	0.00%	0.00%	0.00%	0.00%	0.00%
/Processor6	Processor	35.01%	33.92%	0.00%	0.00%	0.00%	0.00%	0.00%
/Conveyor7	Conveyor	0.00%	0.00%	0.00%	0.00%	0.00%	87.00%	0.00%
/Conveyor8	Conveyor	0.00%	0.00%	0.00%	0.00%	0.00%	86.00%	0.00%
/Conveyor9	Conveyor	0.00%	0.00%	0.00%	0.00%	0.00%	86.46%	0.00%
/Sink10	Sink	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	100.00%
/Operator11	Operator	6.69%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%

Figure 1 - A State Report tab

The State Report tab does not allow the user to edit anything, but they can see the data.

## Object Comparison Chart

### Overview

The Object Comparison Chart tab is used to create bar graphs that compare statistical results about multiple objects. Each graph can display several variables for all of the objects. Each variable is given its own bar in the final graph. Each object in the graph is assigned a color, and all of the bars for the object are in that color. The user can change the color if they want.

### Tab Description

**Object Comparison Chart**

Name:

Variables

Total In  
Total Out

Chart Variables

Total In  
Total Out

Combine Selected

Objects

/Source2  
/Queue3  
/Processor4  
/Processor5  
/Processor6  
/Conveyor7  
/Conveyor8  
/Conveyor9  
/Sink10  
/Operator11

Chart Objects

/Queue3  
/Processor4  
/Processor5  
/Processor6  
/Conveyor7  
/Conveyor8  
/Conveyor9

☒ Use All Itemtypes ☐ Use Time Range

☐ Specific Itemtypes:  From:  To:

Figure 1 - The Object Comparison tab, with variables and objects selected

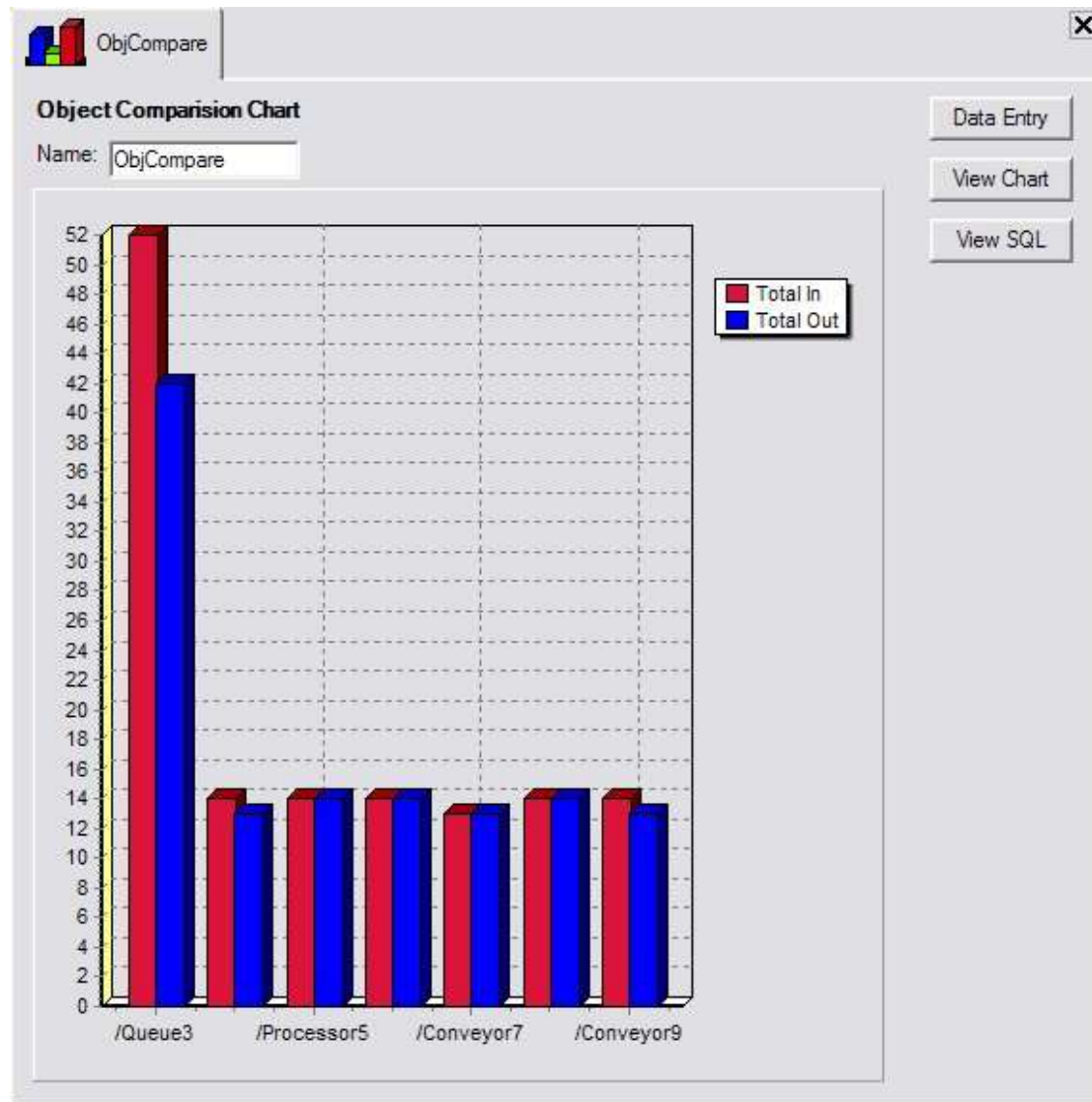






Figure 2 - The graph generated from the settings in Figure 1


**Name:** The name of the chart. This is given a default value when the tab is created, but the user should change it to something more descriptive. This name will be displayed on the tab and in the Select Chart drop-down list.




**Variables:** The drop-down list contains categories of variables that are available for Object Comparison charts. Once a category has been selected, the available variables in that category are listed below the drop-down list. The user can highlight the variables that they would like to put in the chart and press the  button. They can also double-click on individual variables to put them in the chart.

**Chart Variables:** This list displays the variables that will be displayed in the chart. To remove variables from the chart, the user highlights the ones that should be removed and presses the  button. They can also double-click on individual variables to

remove them from the chart. The user can move highlighted entries in this list up and down by using the  and  buttons.

**Combine Selected:** Pressing this button will create a new variable that has the sum of all of the selected chart variables. When this new variable is created, the user will be prompted to give it a name. This name should be something descriptive. The name can not be changed once it has been assigned. Once the combined variable has been created, the originals do not need to be in the Chart Variables list any more. It is not possible to create combined variables using one or more combined variables.

**Objects:** The drop-down list displays all of the groups that were defined in the Flexsim model. There is also a group called "All" that is always available. When the user selects a group from the drop-down list, all of the objects in that group will be listed below the drop-down list. The user highlights the objects that they want to have in the model and presses the  button to place them in the Chart Objects list.

**Chart Objects:** This list contains the objects that will have their data reported in the chart. To remove objects from the list, the user should highlight the objects that they wish to remove and press the  button. They can also double-click on individual objects to remove them from the list. The user can move highlighted entries in this list up and down by using the  and  buttons.

**Use All Itemtypes:** If this box is checked, all flowitems will be used when the tab is calculating the results of flowitem-based variables. Variables that are flowitem-based include throughput and content variables. Variables that are not flowitem-based include state variables.

**Specific Itemtypes:** If this box is checked, only flowitems with certain itemtypes will be used when calculating the variables. The user lists the itemtypes in the text area to the right of the check box. They can list itemtypes as a comma-separated list of individual types, or they can use ranges. For example: 1, 3, 6-10.

**Use Time Range:** If this box is checked the calculations for the variables will only include flowitem movement and object state changes that happened within the range of time the user specifies. The range of time is entered in the boxes labeled "From" and "To".

**Data Entry:** Pressing this button makes the data entry section of the tab visible so that the user can change the settings of the graph.

**View Chart:** Pressing this button calculates the variables for the objects, then creates the bar graph and makes it visible.

**View SQL:** Pressing this button displays the SQL statements generated by each of the variables. This data is strictly read-only. It is provided so that users can gain a feel for the behind-the-scenes behavior of Flexsim Chart. For most variables, the SQL statement is not enough to fully calculate the values. There is also a post-SQL

step that is often used to calculate the values. The user can not change how this post-SQL step works.

## Variables

The Object Comparison Chart has more variables available than any of the other types of tabs, so its variables are grouped into categories. For each of the variables listed below, the category is also listed.

**Total In:** This returns the total number of flowitems of the correct itemtypes that entered each of the objects during the specified time range. This is in the Throughput category.

**Total Out:** This returns the total number of flowitems of the correct itemtypes that exited each of the objects during the specified time range. This is in the Throughput category.

**Minimum Content:** This returns the minimum number of flowitems of the correct itemtypes that were in each object at any point during the specified time range. This is in the Content category.

**Maximum Content:** This returns the maximum number of flowitems of the correct itemtypes that were in each object at any point during the specified time range. This is in the Content category.

**Average Content:** This returns the average number of flowitems of the correct itemtypes that were in each object during the specified time range. This is in the Content category.

**Current Content:** This returns the number of flowitems of the correct itemtypes that were in each object when the model stopped and the data was recorded. This is in the Content category.

**Minimum Staytime:** This returns shortest amount of time that any flowitem that was in the specified itemtype range stayed in each object. It only takes into account flowitems that entered and exited during the time range. This is in the Staytime category.

**Maximum Staytime:** This returns longest amount of time that any flowitem that was in the specified itemtype range stayed in each object. It only takes into account flowitems that entered and exited during the time range. This is in the Staytime category.

**Average Staytime:** This returns average amount of time that any flowitem that was in the specified itemtype range stayed in each object. It only takes into account flowitems that entered and exited during the time range. This is in the Staytime category.



**Last Recorded Staytime:** This returns the amount of time the last flowitem to leave each object stayed in the object. It only takes into account flowitems that entered and exited during the time range. This is in the Staytime category.

**State Totals:** The variables in this category return the total amount of time each object spent in each state during the time range. There is one variable available for each of Flexsim's standard states.

**State Percents:** The variables in this category return the percentage of the time range each object spent in each state. There is one variable available for each of Flexsim's standard states.

**State Averages:** The variables in this category return the average amount of time each object spent in each state whenever they entered that state.

## Time Plot

### Overview

The Time Plot tab is used to create line graphs of variables whose data is meaningful or interesting over a time range. This includes time spent in states, content, flowitem staytime and throughput. Each line in the graph displays the results of a single variable from a single selected object. This means that it is fairly easy to generate graphs with too many lines to be really useful. Users should consider using each graph for a small amount of information, and they should use multiple graphs if one starts getting too cluttered.

### Tab Description

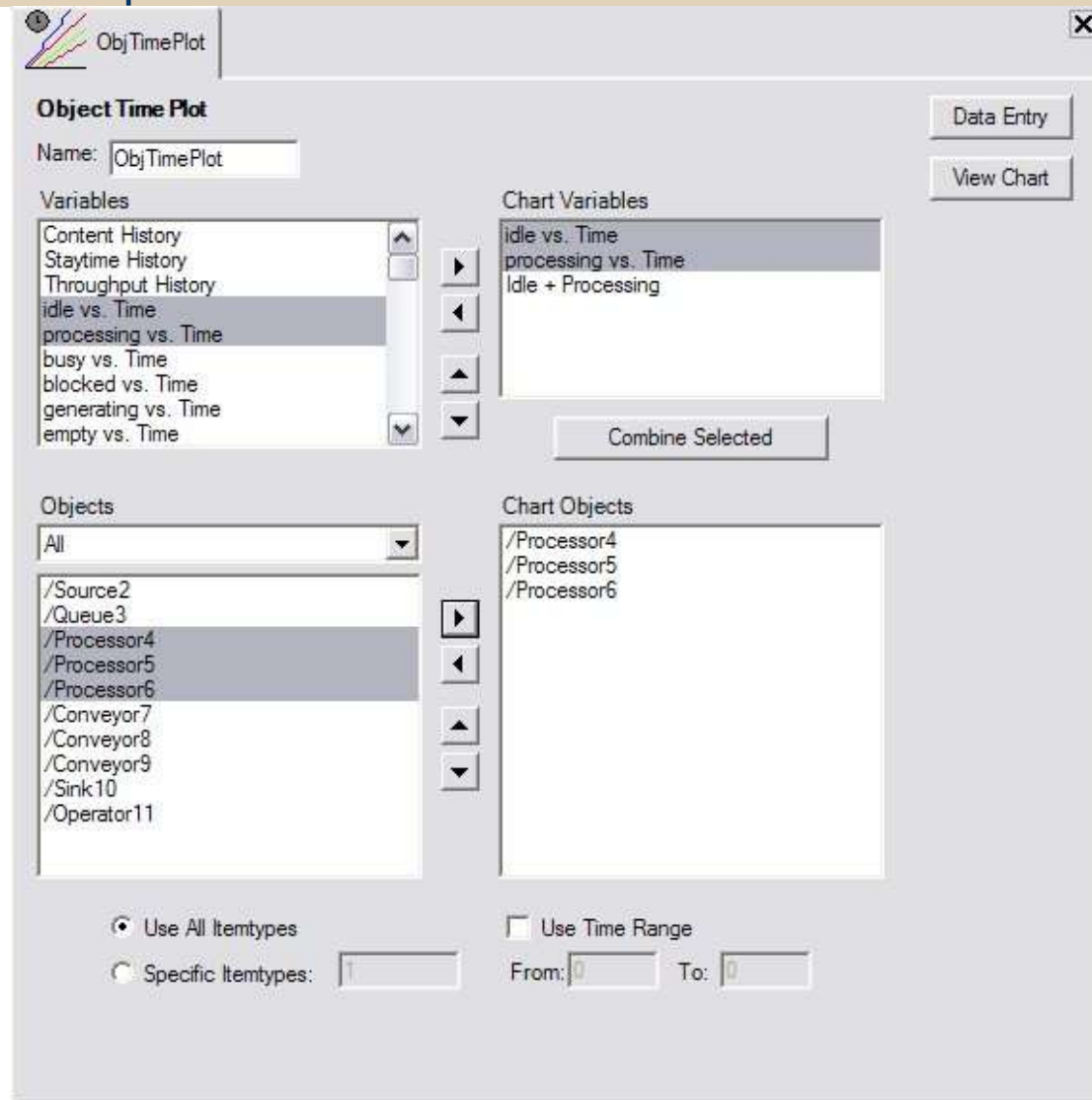


Figure 1 - A Time Plot with variables and objects selected

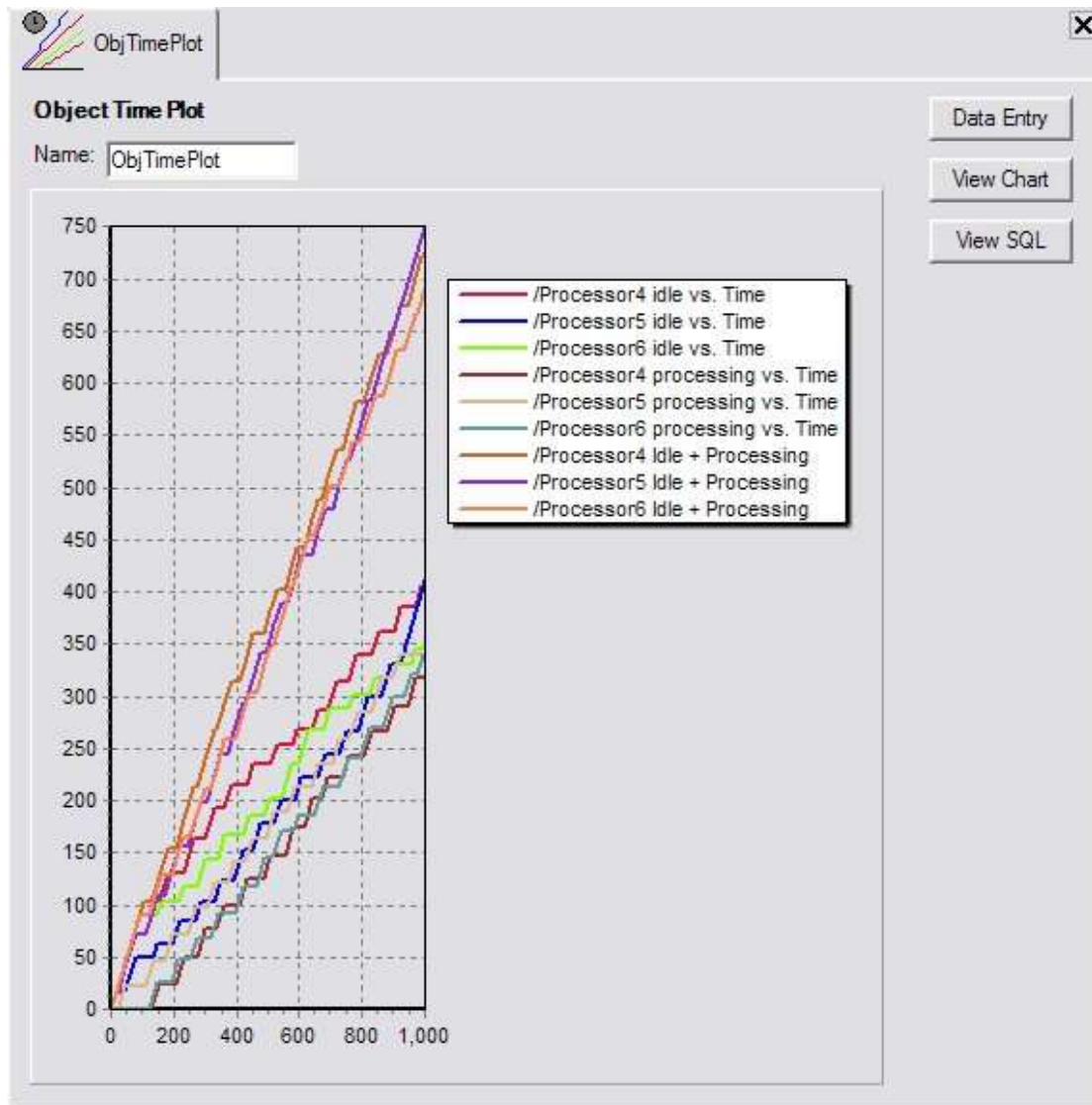







Figure 2 - The graph generated from the settings in Figure 1




**Name:** The name of the chart. This is given a default value when the tab is created, but the user should change it to something more descriptive. This name will be displayed on the tab and in the Select Chart drop-down list.

**Variables:** This is a list of variables that are available for Time Plots. The user can highlight the variables that they would like to put in the chart and press the  button. They can also double-click on individual variables to put them in the chart.

**Chart Variables:** This list displays the variables that will be displayed in the chart. To remove them from the chart, the user highlights the ones that should be removed and presses the  button. They can also double-click on individual variables to remove them from the chart. The user can move highlighted entries in this list up and down by using the  and  buttons.

**Combine Selected:** Pressing this button will create a new variable that has the sum of all of the selected chart variables. When this new variable is created, the user will be prompted to give it a name. This name should be something descriptive. The name can not be changed once it has been assigned. Once the combined variable has been created, the originals do not need to be in the Chart Variables list any more. It is not possible to create combined variables using one or more combined variables.

**Objects:** The drop-down list displays all of the groups that were defined in the Flexsim model. There is also a group called "All" that is always available. When the user selects a group from the drop-down list, all of the objects in that group will be listed below the drop-down list. The user highlights the objects that they want to have in the model and presses the  button to place them in the Chart Objects list.

**Chart Objects:** This list contains the objects that will have their data reported in the chart. To remove objects from the list, the user should highlight the objects that they wish to remove and press the  button. They can also double-click on individual objects to remove them from the list. The user can move highlighted entries in this list up and down by using the  and  buttons.

**Use All Itemtypes:** If this box is checked, all flowitems will be used when the tab is calculating the results of flowitem-based variables. Variables that are flowitem-based include throughput and content variables. Variables that are not flowitem-based include state variables.

**Specific Itemtypes:** If this box is checked, only flowitems with certain itemtypes will be used when calculating the variables. The user lists the itemtypes in the text area to the right of the check box. They can list itemtypes as a comma-separated list of individual types, or they can use ranges. For example: 1, 3, 6-10.

**Use Time Range:** If this box is checked the calculations for the variables will only include flowitem movement and object state changes that happened within the range of time the user specifies. The range of time is entered in the boxes labeled "From" and "To".

**Data Entry:** Pressing this button makes the data entry section of the tab visible so that the user can change the settings of the graph.

**View Chart:** Pressing this button calculates the variables for the objects, then creates the line graph and makes it visible.

**View SQL:** Pressing this button displays the SQL statements generated by each of the variables. This data is strictly read-only. It is provided so that users can gain a feel for the behind-the-scenes behavior of Flexsim Chart. For most variables, the SQL statement is not enough to fully calculate the values. There is also a post-SQL step that is often used to calculate the values. The user can not change how this post-SQL step works.

## Variables

**Content History:** This variable displays the content of the selected objects as it changes over the specified time range. Only flowitems whose itemtype matches the specified list will be included in this result.

**Staytime History:** Whenever a flowitem leaves one of the selected objects, its staytime and the time it leaves are used to add a point to the graph this variable displays. Only flowitems whose itemtype matches the specified list will be included in the result.

**Throughput History:** This variable displays the output of the selected objects as it changes over the specified time range. Only flowitems whose itemtype matches the specified list will be included in the result.

**State vs. Time:** There is a state vs. time variable available for each of Flexsim's standard states. Each of these display the total amount of time the selected objects were in the corresponding states during the specified time range.

## Financial Reports

### Overview

The Financial Reports tab allows the user to assign financial information to the objects in the model so that they can translate numerical model results into monetary results. The values assigned to the objects can be negative to indicate cost or positive to indicate gains. There are several categories of financial information that can be assigned. The user is not required to use all of them, but they are available if needed. These categories include fixed and time-based values for the object, fixed and time-based values for flowitems, and time-based values for object states. There are two reports available to the user. One is called the "Totals Report". This is the grand total of all of the financial data assigned to all of the objects in the model. The other report is called the "Details Report". This shows the results of the financial data for each individual object. The sum of all this information makes up the Totals Report. The financial reports will tell the user if the totals are positive (profit) or negative (loss).

### Tab Description

**Financial Report**

Name:

Fixed Value:  Time Value:

Itemtype(s)	Fixed Value	Time Value
1	100	-1
2	150	-1
3-4	200	-1

☐ Use Time Range

From:  To:

Figure 1 - A Financial Reports tab with financial data assigned to some objects



Figure 2 - A Totals Report generated by the settings in Figure 1





Figure 3 - A Details Report generated by the settings in Figure 1

**Name:** The name of the chart. This is given a default value when the tab is created, but the user should change it to something more descriptive. This name will be displayed on the tab and in the Select Chart drop-down list.

**Objects:** The drop-down list displays all of the groups that were defined in the Flexsim model. There is also a group called "All" that is always available. When a group is selected, all of the objects in that group are listed below the drop-down list. An object that has asterisks (\*) by its name has financial data assigned to it. To add financial data to an object, highlight it in the list and fill in either the Fixed Value or Time Value field. It is acceptable to enter the number 0 in either of these fields to assign financial data. Once a value (even 0) has been assigned to one of these fields, all of the financial fields will become available. If an object does not have financial data assigned to it, it will not be visible in the Details Report and will not affect the Totals Report.

**Fixed Value:** This value is assigned to the object once in the results calculation. It can be thought of in many cases as the cost of the installation of the object.

**Time Value:** This value is multiplied by the length of the time range of the report and added to the result calculation. It can represent the cost of keeping the object running (electricity, etc).

**Table drop-down list:** This drop-down list allows the user to select the table that will be displayed below it. There are two tables available: the Flowitem Values Table and the State Values Table.

**Flowitem Values Table:** This table allows the user to enter financial information that relates to the flowitems that the object processed. There are three columns that the user can fill out. The first is the range of itemtypes that the row applies to. This range can be a comma-separated list of values, or it can include ranges. For example: 1,3,5-7. The second column is the fixed value for each flowitem that matches the itemtype criteria. This value is applied to the total for each flowitem in the itemtype range. The third column is the time value for the flowitems that match the itemtype criteria. This value is multiplied by the time that each flowitem spent in the object and added to the total. The user can add new rows to this table, one row for each range of itemtypes that need financial data assigned to them

**State Values Table:** This table allows the user to enter financial information about each state that the object went through. Each row represents a single state. The first column is the name of the state, and can not be changed by the user. The second column is the time value for that state. This value is multiplied by the amount of time the object spent in the state and is added to the total. The user can not change the number of rows in this table.

**Remove:** Pressing this button will remove the financial data from the currently selected object. Once this is done, all the data that the user entered for the current object will be lost.

**Use Time Range:** If this box is checked the calculations for the financial reports will only include flowitem movement and object state changes that happened within the range of time the user specifies. The range of time is entered in the boxes labeled "From" and "To".

**Data Entry:** Pressing this button makes the data entry section of the tab visible so that the user can change the settings of the reports.

**View Totals:** Pressing this button will calculate the Totals Report and display it. Depending on the number of objects in the model, the number of flowitems they processed and the length of the time range this calculation may take a little while. The report will appear as soon as the calculations are complete.

**Details:** Pressing this button will calculate the Details Report and display it. Depending on the number of objects in the model, the number of flowitems they processed and the length of the time range this calculation may take a little while. The report will appear as soon as the calculations are complete.

## Object Gantt Chart

### Overview

The Object Gantt Chart tab is used to display object data that changes over time. Each object in the chart is given a horizontal bar that represents the data corresponding to that object. This bar is broken into smaller sections whose meaning is different depending on the variable being displayed. Sometimes the small bars represent a flowitem's staytime, other times they represent the time the object spent in a specific state. For variables that involve flowitems traveling between objects, there are lines drawn between the bars of the Gantt chart in order to help the user see where the flowitem traveled to next.

### Tab Description

**Object Gantt Chart**

Name:

**Variables**

- Object State
- Flowitem Trace
- Itemtype Trace

**Chart Variables**

- Flowitem Trace

**Objects**

- /Source2
- /Queue3
- /Processor4
- /Processor5
- /Processor6
- /Conveyor7
- /Conveyor8
- /Conveyor9
- /Sink10
- /Operator11

**Chart Objects**

- /Queue3
- /Processor4
- /Processor5
- /Processor6
- /Conveyor7
- /Conveyor8
- /Conveyor9

☒ Use All Itemtypes      ☐ Use Time Range

☐ Specific Itemtypes:       From:  To:

Figure 1 - An Object Gantt tab with variables and objects selected

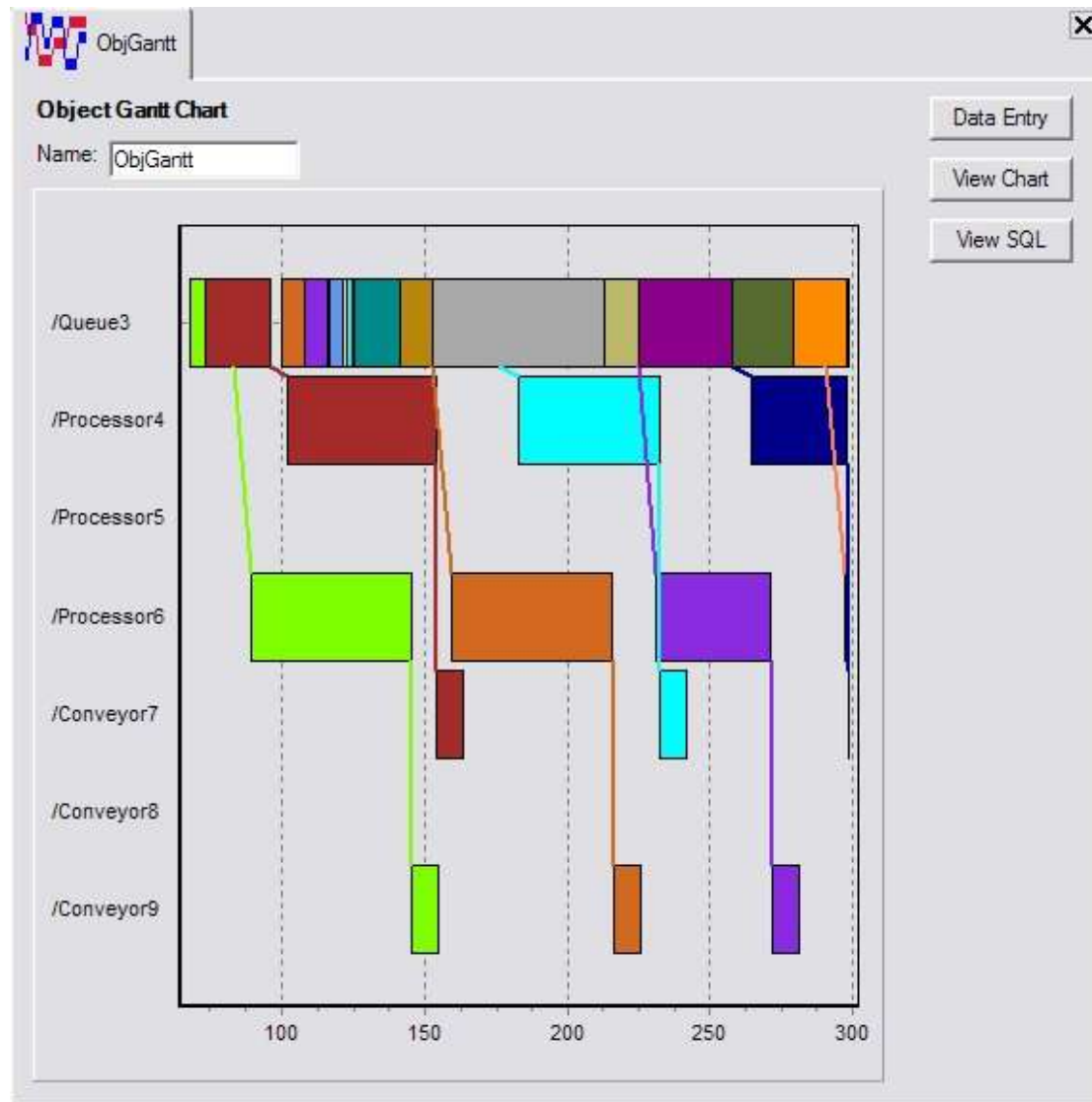










Figure 2 - The graph generated by the settings in Figure 1

**Name:** The name of the chart. This is given a default value when the tab is created, but the user should change it to something more descriptive. This name will be displayed on the tab and in the Select Chart drop-down list.

**Variables:** This is a list of variables that are available for Object Gantt charts. The user can highlight the variables that they would like to put in the chart and press the  button. They can also double-click on individual variables to put them in the chart.

**Chart Variables:** This list displays the variables that will be displayed in the chart. To remove them from the chart, the user highlights the ones that should be removed and presses the  button. They can also double-click on individual variables to remove them from the chart. The user can move highlighted entries in this list up and down by using the  and  buttons.

**Objects:** The drop-down list displays all of the groups that were defined in the Flexsim model. There is also a group called "All" that is always available. When the user selects a group from the drop-down list, all of the objects in that group will be listed below the drop-down list. The user highlights the objects that they want to have in the model and presses the  button to place them in the Chart Objects list.

**Chart Objects:** This list contains the objects that will have their data reported in the chart. To remove objects from the list, the user should highlight the objects that they wish to remove and press the  button. They can also double-click on individual objects to remove them from the list. The user can move highlighted entries in this list up and down by using the  and  buttons.

**Use All Itemtypes:** If this box is checked, all flowitems will be used when the tab is calculating the results of flowitem-based variables. Variables that are flowitem-based include flowitem and itemtype trace variables. Variables that are not flowitem-based include state variables.

**Specific Itemtypes:** If this box is checked, only flowitems with certain itemtypes will be used when calculating the variables. The user lists the itemtypes in the text area to the right of the check box. They can list itemtypes as a comma-separated list of individual types, or they can use ranges. For example: 1, 3, 6-10.

**Use Time Range:** If this box is checked the calculations for the variables will only include flowitem movement and object state changes that happened within the range of time the user specifies. The range of time is entered in the boxes labeled "From" and "To".

**Data Entry:** Pressing this button makes the data entry section of the tab visible so that the user can change the settings of the graph.

**View Chart:** Pressing this button calculates the variables for the objects, then creates the Gantt chart and makes it visible.

**View SQL:** Pressing this button displays the SQL statements generated by each of the variables. This data is strictly read-only. It is provided so that users can gain a feel for the behind-the-scenes behavior of Flexsim Chart. For most variables, the SQL statement is not enough to fully calculate the values. There is also a post-SQL step that is often used to calculate the values. The user can not change how this post-SQL step works.

## Variables

**Object State:** This variable generates a Gantt chart that displays the state the object was in at any given time during the specified time range. Each individual state is assigned a different color that can be changed by the user if they desire.

**Flowitem Trace:** This variable generates a Gantt chart that displays how flowitems travel between objects. Each unique flowitem is assigned a different color. Each time a flowitem enters an object a box is drawn that represents the staytime in that object. When the flowitem moves to another object, another box is drawn for that object and

a line is drawn between them. This indicates to the user that the flowitem has traveled to another object and shows them which object.

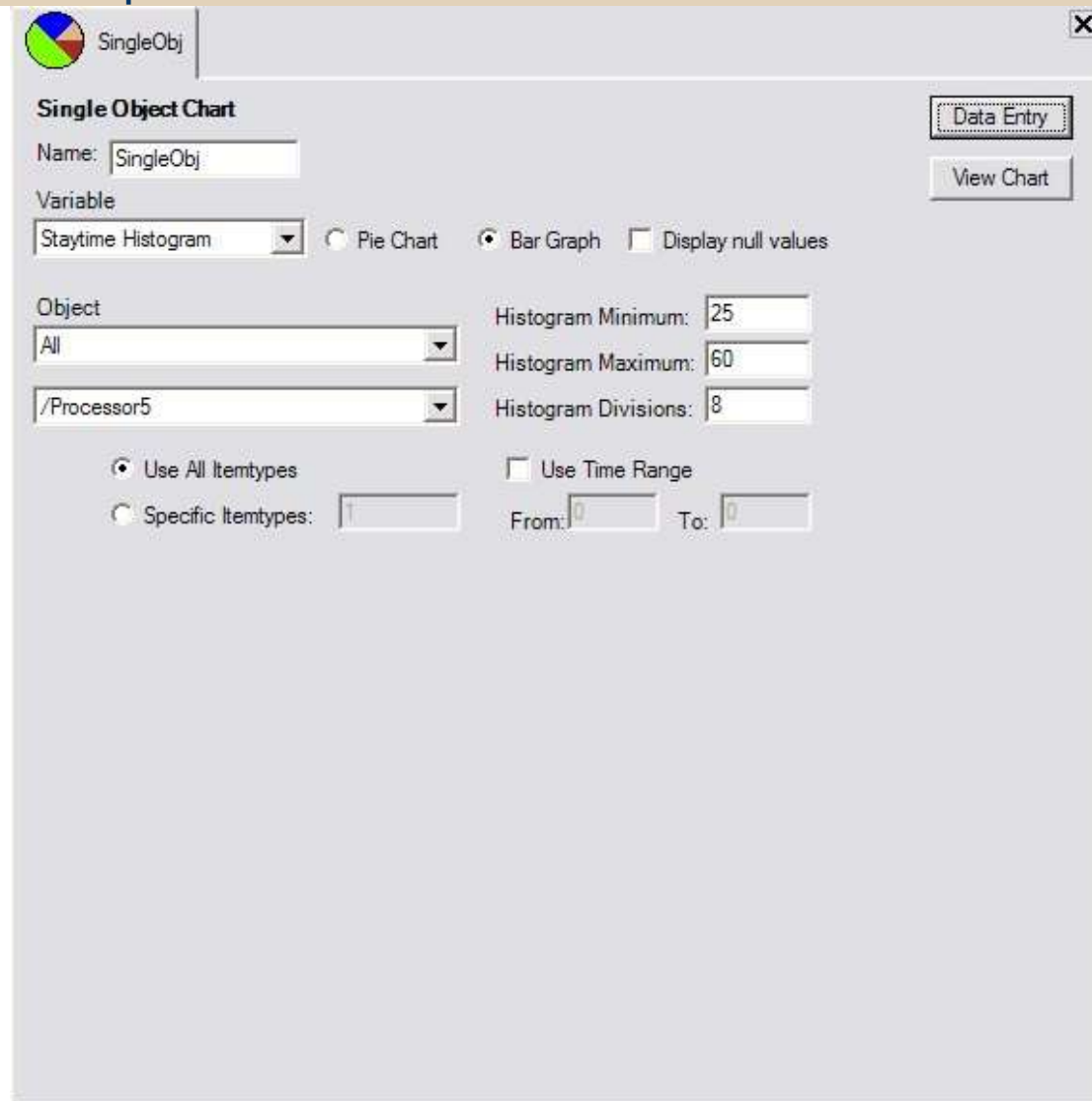
**Itemtype Trace:** This variable generates a Gantt chart that displays when flowitems of different itemtypes are in each object. Each itemtype is assigned a color that can be changed by the user. This graph will look similar to the Flowitem Trace graph, but will have colors for itemtypes, not individual flowitems. There are also no lines drawn to connect the bars in this chart.

## Single Object Chart

### Overview

The Single Object Chart tab is used to display either a pie chart or bar graph that represents one particular type of data about a single object. The data can be information about staytime of flowitems, time spent in different states or object content.

### Tab Description



**Single Object Chart**

Name:

Variable:  ☐ Pie Chart ☒ Bar Graph ☐ Display null values

Object:

Histogram Minimum:  Histogram Maximum:  Histogram Divisions:

☒ Use All Itemtypes ☐ Use Time Range

☐ Specific Itemtypes:  From:  To:

Data Entry View Chart

Figure 1 - A Single Object tab with a variable and object selected

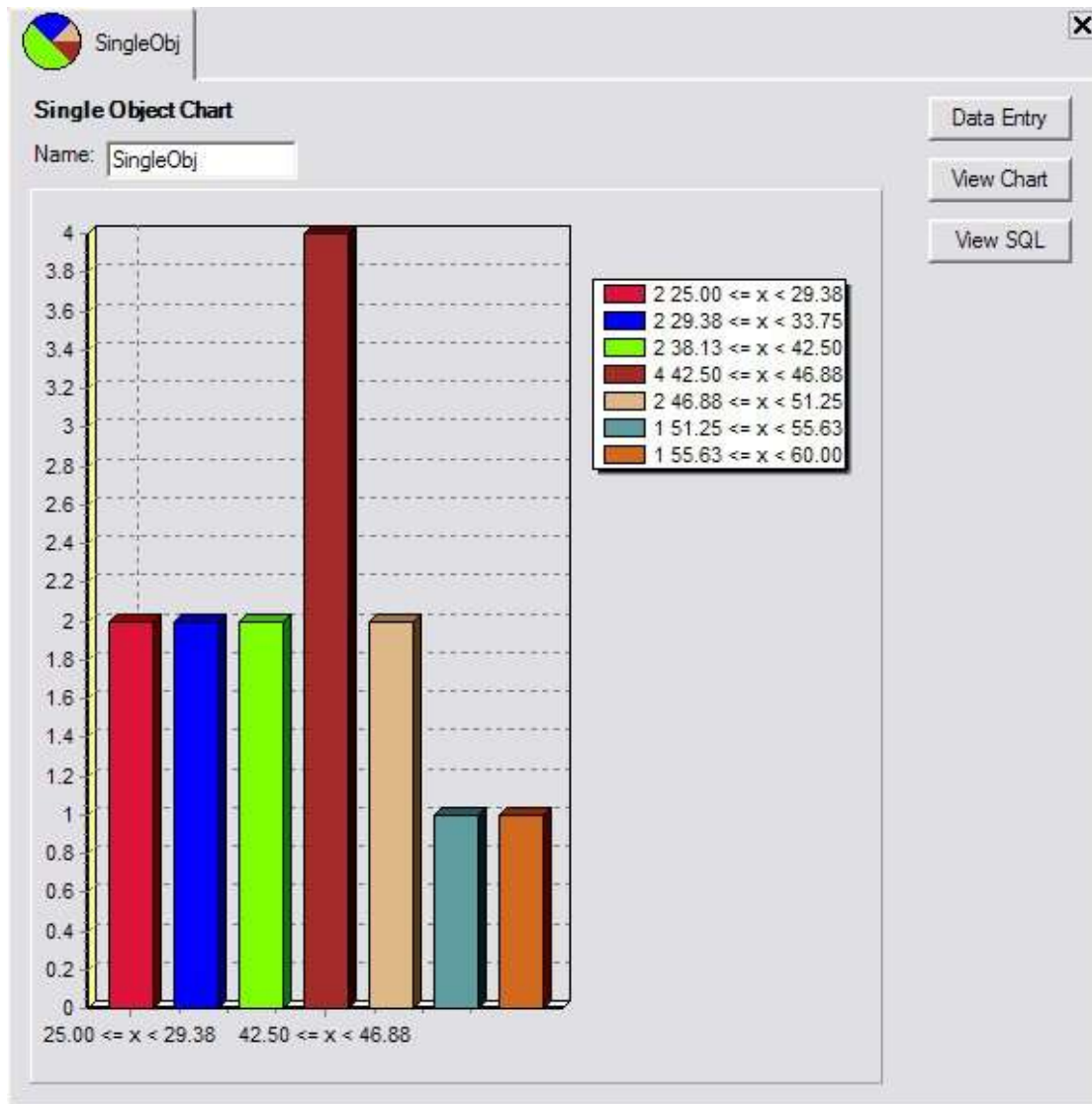
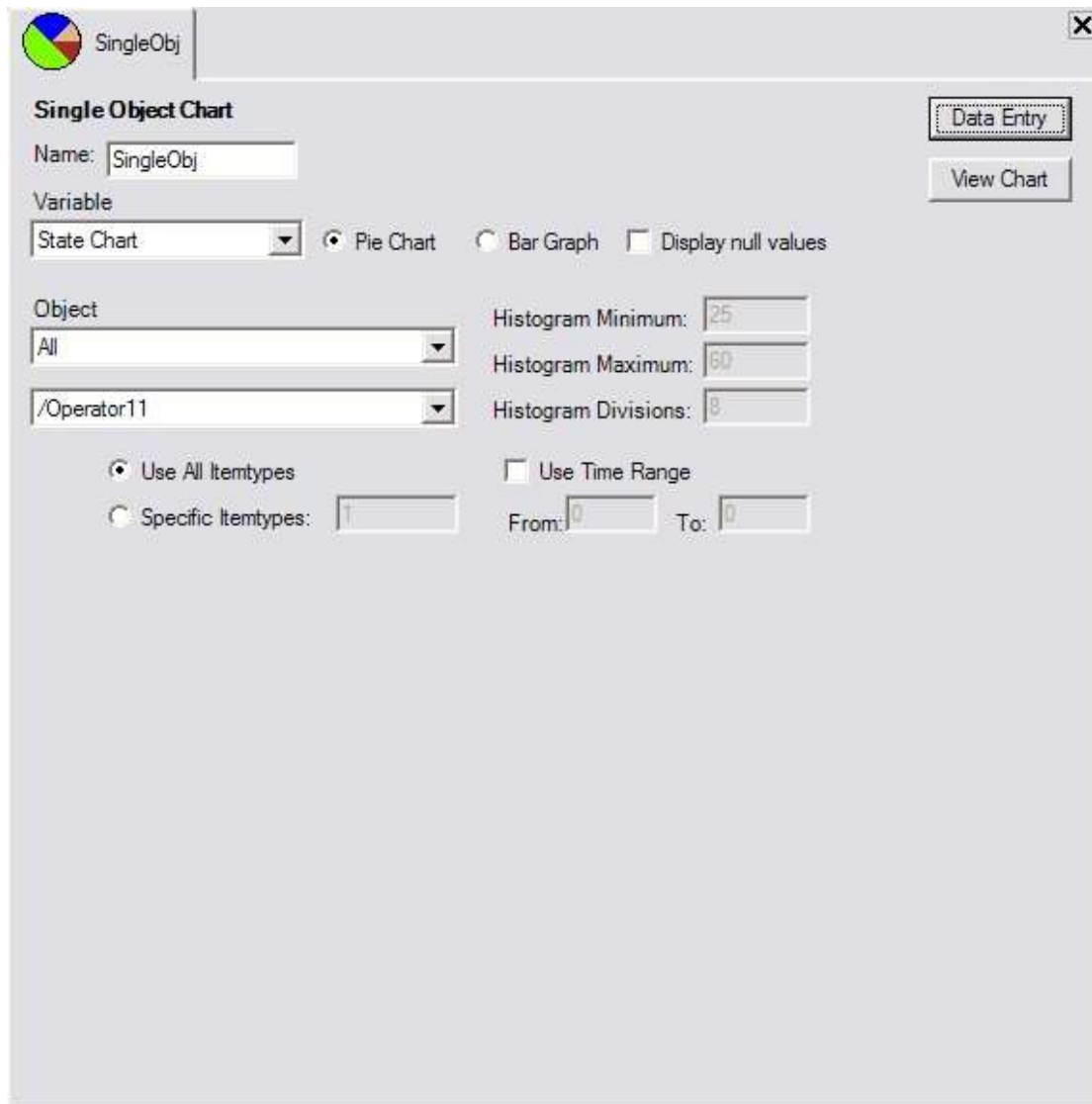


Figure 2 - The graph generated by the settings in Figure 1





The image shows a software window titled "SingleObj" with a close button (X) in the top right corner. The window contains a "Single Object Chart" section with the following settings:

- Name:** SingleObj
- Variable:** State Chart (dropdown menu)
- Object:** All (dropdown menu)
- Object:** /Operator11 (dropdown menu)
- Chart Type:** ☒ Pie Chart, ☐ Bar Graph
- Display null values:** ☐
- Histogram Minimum:** 25
- Histogram Maximum:** 60
- Histogram Divisions:** 8
- Use All Itemtypes:** ☒
- Specific Itemtypes:** 1
- Use Time Range:** ☐
- From:** 0
- To:** 0

Buttons: Data Entry, View Chart

Figure 3 - Different settings for a Single Object tab

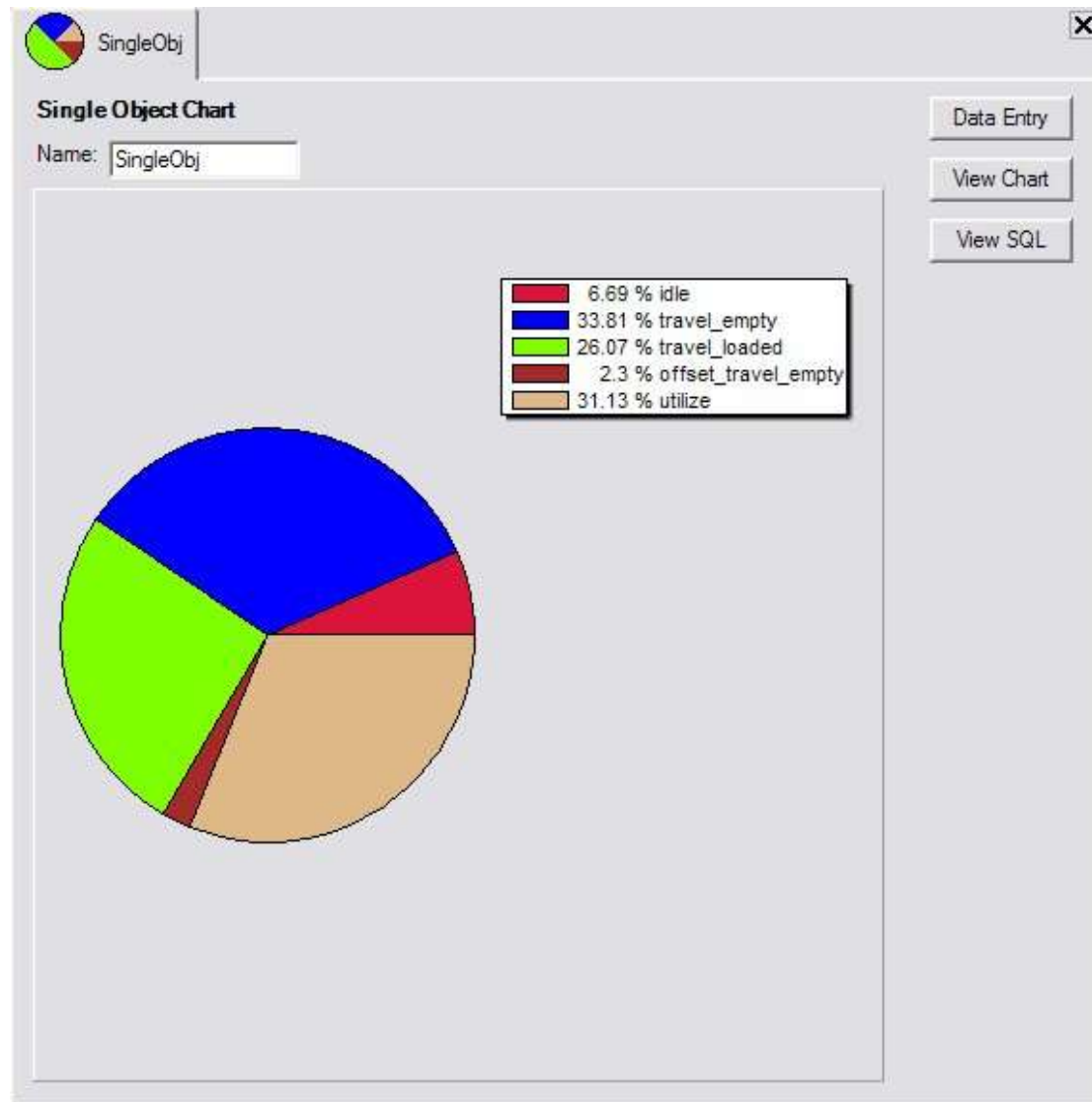


Figure 4 - The graph generated by the settings in Figure 3

**Name:** The name of the chart. This is given a default value when the tab is created, but the user should change it to something more descriptive. This name will be displayed on the tab and in the Select Chart drop-down list.

**Variable:** This is a list of variables that are available for Single Object graphs. The user selects from the list the variable that they would like to put in the chart.

**Pie Chart:** If this button is checked, the resulting graph will be a pie chart. The data displayed will not be affected, only the presentation of it.

**Bar Graph:** If this button is checked, the resulting graph will be a bar graph. The data displayed will not be affected, only the presentation of it.

**Display null values:** If this box is checked, the graph will include bars or pie slices whose value is 0. If this button is not checked, no 0 values will be displayed. This is useful to keep the legend and the graph less cluttered. It is not checked by default.

**Object:** The first drop-down list displays all of the groups that were defined in the Flexsim model. There is also a group called "All" that is always available. When the user selects a group from the drop-down list, all of the objects in that group will be listed in the second drop-down list. The user selects the object that they want generate a graph about.

**Histogram Minimum:** This is the lowest value that will be displayed in the bar graph.

**Histogram Maximum:** This is the highest value that will be displayed in the bar graph.

**Histogram Divisions:** This is the maximum number of divisions (bars or pie slices) that will be drawn in the graph. They will all be drawn if none of them have a value of 0 or the "Display null values" box is checked.

**Use All Itemtypes:** If this box is checked, all flowitems will be used when the tab is calculating the results of flowitem-based variables. Variables that are flowitem-based include throughput and content variables. Variables that are not flowitem-based include state variables.

**Specific Itemtypes:** If this box is checked, only flowitems with certain itemtypes will be used when calculating the variables. The user lists the itemtypes in the text area to the right of the check box. They can list itemtypes as a comma-separated list of individual types, or they can use ranges. For example: 1, 3, 6-10.

**Use Time Range:** If this box is checked the calculations for the variables will only include flowitem movement and object state changes that happened within the range of time the user specifies. The range of time is entered in the boxes labeled "From" and "To".

**Data Entry:** Pressing this button makes the data entry section of the tab visible so that the user can change the settings of the graph.

**View Chart:** Pressing this button calculates the variables for the objects, then creates the bar or pie graph and makes it visible.

**View SQL:** Pressing this button displays the SQL statements generated by each of the variables. This data is strictly read-only. It is provided so that users can gain a feel for the behind-the-scenes behavior of Flexsim Chart. For most variables, the SQL statement is not enough to fully calculate the values. There is also a post-SQL step that is often used to calculate the values. The user can not change how this post-SQL step works.

## Variables

**State Chart:** This returns a graph whose values are the percentage of the time range the object was in each state. It is most commonly viewed as a pie chart.

**Staytime Histogram:** Each value in the graph this variable generates indicates how many flowitems had a particular staytime in the selected object. Each section of the graph represents a range of times, based on the histogram minimum, maximum and number of divisions. It is most commonly viewed as a bar graph.

**Content Histogram:** Each value in the graph this variable generates indicates how many times a particular number of flowitems were in the object immediately after a new flowitem arrived. The new flowitem is included in this count. Each section of the graph represents a range of values, based on the histogram minimum, maximum and number of divisions. It is most commonly viewed as a bar graph.

## Flowitem Gantt Chart

### Overview

The Flowitem Gantt Chart tab is used to display flowitem data that changes over time. Each flowitem in the chart is given a horizontal bar that represents the data corresponding to that flowitem. This bar is broken into smaller, colored bars. These small bars represent the time the flowitem spent in the objects in the model. Each object is given a different color. This lets the user see where a specific flowitem went during a model run. In most models, the number of flowitems that are processed is very large. This graph becomes difficult to read and analyze if there are too many flowitems selected. However, models that have a very limited number of flowitems will produce excellent results in this tab.

### Tab Description

**Flowitem Gantt Chart**

Name:

Variables

Object Trace

Chart Variables

Object Trace

Flowitems

FlowitemID: 1  
FlowitemID: 2  
FlowitemID: 3  
FlowitemID: 4  
FlowitemID: 5  
FlowitemID: 6  
FlowitemID: 7  
FlowitemID: 8  
FlowitemID: 9  
FlowitemID: 10  
FlowitemID: 11  
FlowitemID: 12  
FlowitemID: 13  
FlowitemID: 14

Chart Flowitems

FlowitemID: 1  
FlowitemID: 2  
FlowitemID: 3  
FlowitemID: 4  
FlowitemID: 5  
FlowitemID: 6  
FlowitemID: 7  
FlowitemID: 8  
FlowitemID: 9  
FlowitemID: 10  
FlowitemID: 11  
FlowitemID: 12  
FlowitemID: 13  
FlowitemID: 14

☐ Use Time Range

From:  To:

Figure 1 - A Flowitem Gantt tab with variables and flowitems selected

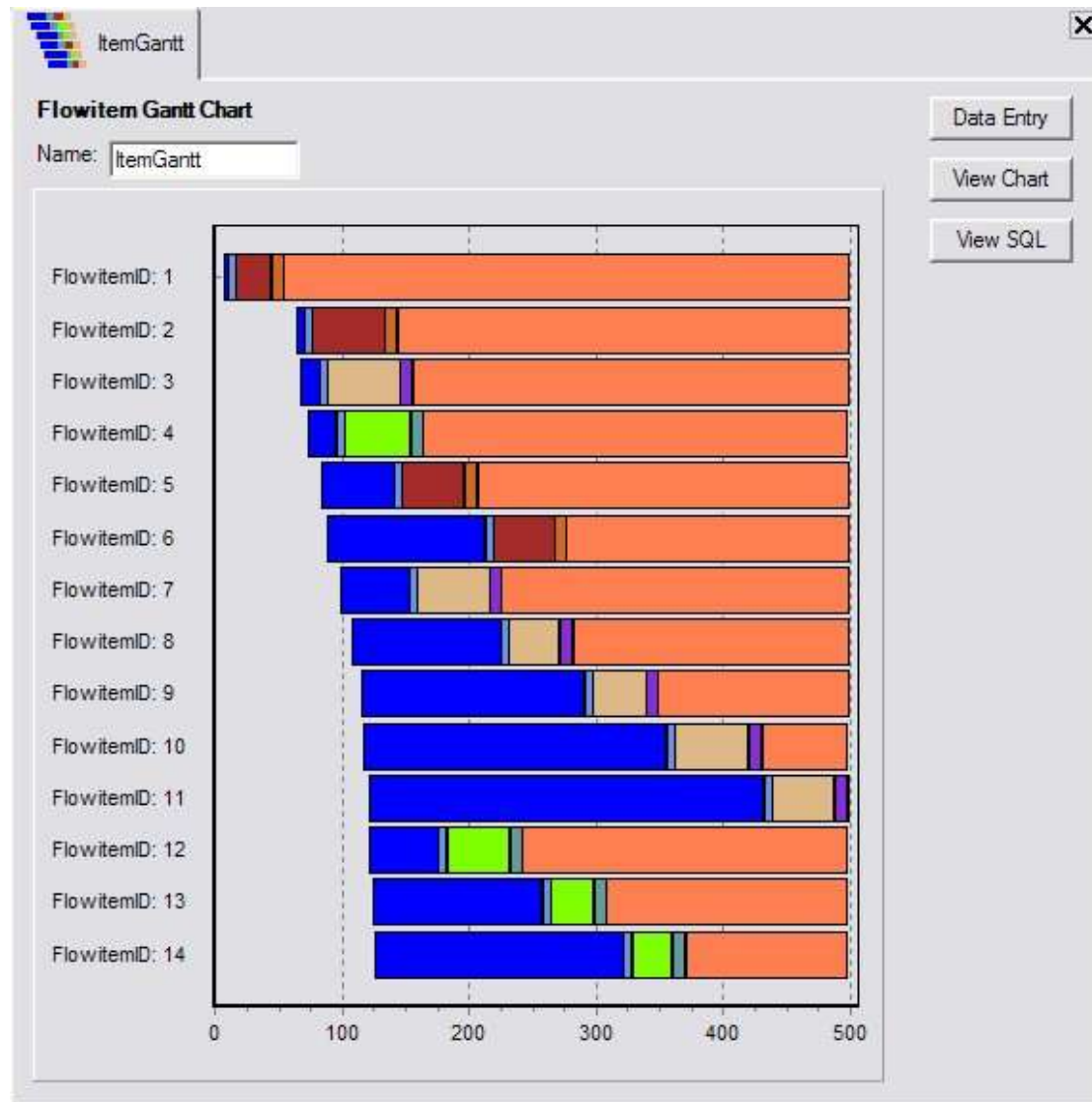










Figure 2 - The graph generated by the settings in Figure 1

**Name:** The name of the chart. This is given a default value when the tab is created, but the user should change it to something more descriptive. This name will be displayed on the tab and in the Select Chart drop-down list.

**Variables:** This is a list of variables that are available for Object Gantt charts. The user can highlight the variables that they would like to put in the chart and press the  button. They can also double-click on individual variables to put them in the chart.

**Chart Variables:** This list displays the variables that will be displayed in the chart. To remove them from the chart, the user highlights the ones that should be removed and presses the  button. They can also double-click on individual variables to remove them from the chart. The user can move highlighted entries in this list up and down by using the  and  buttons.

**Flowitems:** This list is all of the flowitems that were created in the model. The user highlights the flowitems that they want to have in the model and presses the  button to place them in the Chart Flowitems list.

**Chart Flowitems:** This list contains the flowitems that will have their data reported in the chart. To remove flowitems from the list, the user should highlight the flowitems that they wish to remove and press the  button. They can also double-click on individual flowitems to remove them from the list. The user can move highlighted entries in this list up and down by using the  and  buttons.

**Use Time Range:** If this box is checked the calculations for the variables will only include flowitem movement that happened within the range of time the user specifies. The range of time is entered in the boxes labeled "From" and "To".

**Data Entry:** Pressing this button makes the data entry section of the tab visible so that the user can change the settings of the graph.

**View Chart:** Pressing this button calculates the variables for the objects, then creates the Gantt chart and makes it visible.

**View SQL:** Pressing this button displays the SQL statements generated by each of the variables. This data is strictly read-only. It is provided so that users can gain a feel for the behind-the-scenes behavior of Flexsim Chart. For most variables, the SQL statement is not enough to fully calculate the values. There is also a post-SQL step that is often used to calculate the values. The user can not change how this post-SQL step works.

## Variables

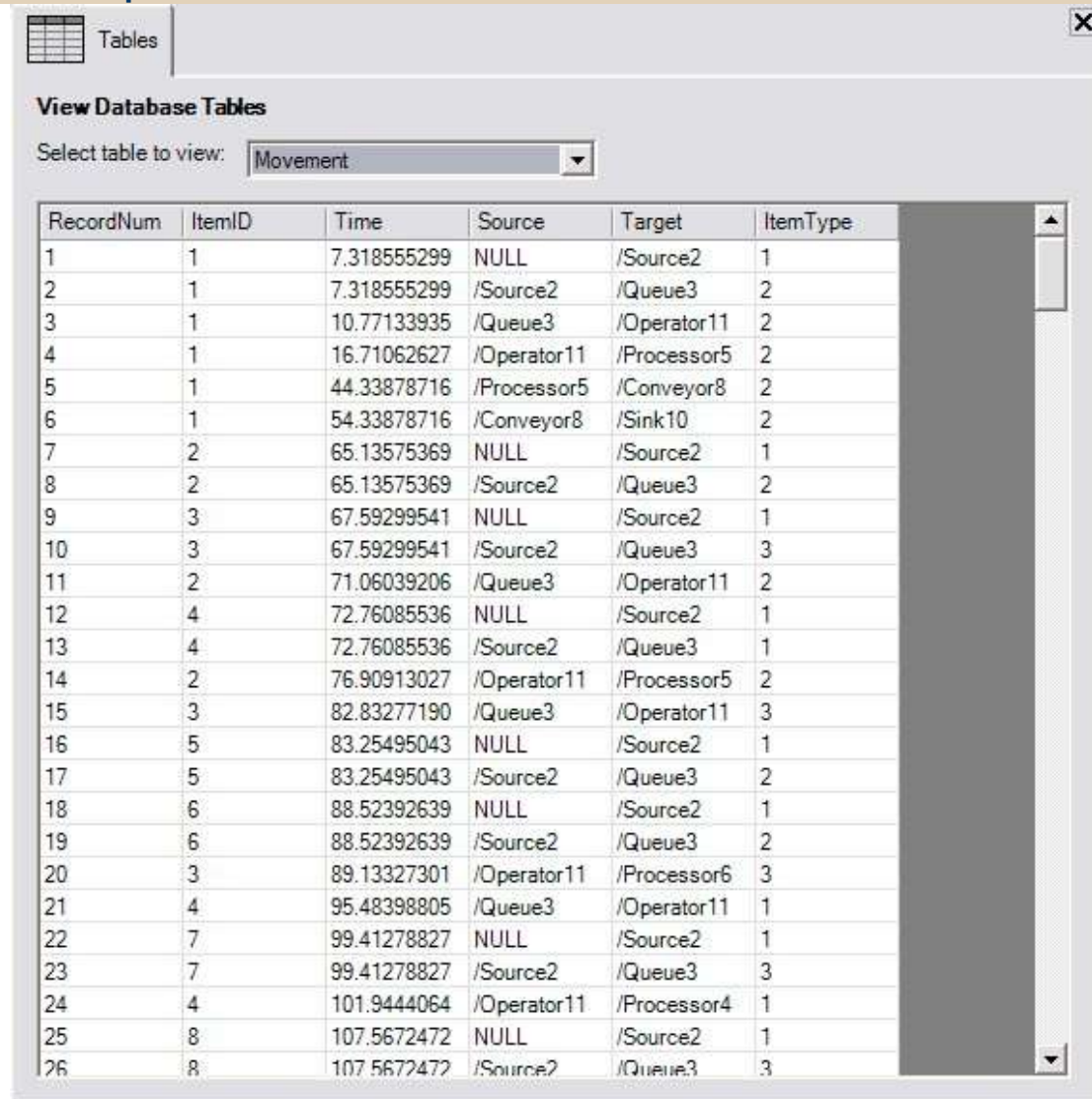
**Object Trace:** This variable generates a Gantt chart that represents the objects that each flowitem traveled through. Each object is assigned a different color. This can be changed by the user. The length of each small bar on a flowitem's main bar is the length of time that the flowitem was in the object indicated by the small bar's color.

## Database Tables

### Overview

The Database Tables tab is used to view the actual tables that are stored in the database file that Flexsim created. They can not be changed using this tab. This tab is not typically needed to analyze the results of a model run. It is provided simply to show the user the raw data that was collected.

### Tab Description



RecordNum	ItemID	Time	Source	Target	ItemType
1	1	7.318555299	NULL	/Source2	1
2	1	7.318555299	/Source2	/Queue3	2
3	1	10.77133935	/Queue3	/Operator11	2
4	1	16.71062627	/Operator11	/Processor5	2
5	1	44.33878716	/Processor5	/Conveyor8	2
6	1	54.33878716	/Conveyor8	/Sink10	2
7	2	65.13575369	NULL	/Source2	1
8	2	65.13575369	/Source2	/Queue3	2
9	3	67.59299541	NULL	/Source2	1
10	3	67.59299541	/Source2	/Queue3	3
11	2	71.06039206	/Queue3	/Operator11	2
12	4	72.76085536	NULL	/Source2	1
13	4	72.76085536	/Source2	/Queue3	1
14	2	76.90913027	/Operator11	/Processor5	2
15	3	82.83277190	/Queue3	/Operator11	3
16	5	83.25495043	NULL	/Source2	1
17	5	83.25495043	/Source2	/Queue3	2
18	6	88.52392639	NULL	/Source2	1
19	6	88.52392639	/Source2	/Queue3	2
20	3	89.13327301	/Operator11	/Processor6	3
21	4	95.48398805	/Queue3	/Operator11	1
22	7	99.41278827	NULL	/Source2	1
23	7	99.41278827	/Source2	/Queue3	3
24	4	101.9444064	/Operator11	/Processor4	1
25	8	107.5672472	NULL	/Source2	1
26	8	107.5672472	/Source2	/Queue3	3

Figure 1 - The Tables tab displaying the Movement table

**Select table to view:** This drop-down list lists all of the tables that are in the database. Whenever the user selects one, it will be displayed in the table view. Some very large tables may take a little while to become fully visible.



## Interacting with Graphs

### Overview

The graphs generated by Flexsim Chart can be customized to a fairly high degree by the user. They can zoom in or out to see sections of the graph more clearly. They can change the colors that are used to display the graph. They can change the background color, the legend position, color and visibility and many other aspects of the graph. They can also save or print the graphs.

### Mouse interaction

When a graph is first generated, Flexsim Chart tries to fit it all on the screen at once. This may make the graph appear crowded. The user can zoom into the graph to see certain parts better. Once zoomed, they can also move the graph around to see the portions that are no longer on the screen.

There are two ways to zoom into a graph. The user can use the scroll wheel of their mouse if they have one. Scrolling the wheel forward will zoom in on the graph, scrolling backwards will zoom out. They can also hold the Shift key and then, using the left mouse button, drag a box around an area of the graph. The graph will zoom in to show the selected area only.

To move the graph around, the user clicks and drags with the left mouse button. This allows them to see the sections of the graph that may no longer be visible because the user has zoomed in.

If the user right-clicks on the graph a menu appears that allows them to customize the graph further.

### Right-click Menu

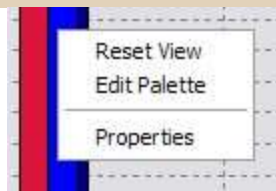


Figure 1 - A Chart's right-click menu

**Reset View:** This option resets the graph view back to the way it was when the graph was first created. It attempts to fit the entire graph on the screen at once. This may cause certain graphs to appear crowded and difficult to read.

**Edit Palette:** This options allows the user to change the colors that are drawn in the graph. Any changes made to a color will apply to all of the parts of that graph that were previously drawn in the changed color. For example, if the user changes the red sections of a graph to green, all of the sections that were red before will now be green.

**Properties:** This opens the Graph Editing GUI. This dialog is used to customize nearly everything about the graph. Most of the features in this GUI will not be needed by most users. The most commonly used features are described below.

### Changing the Color Palette

Every graph has a palette of one hundred colors that are used to draw the bars, lines or pie slices in the graph. Each section of the graph is assigned a palette entry, based on the value of that section. Changing a color in the palette will affect all of the sections in the graph that are assigned to that palette entry.

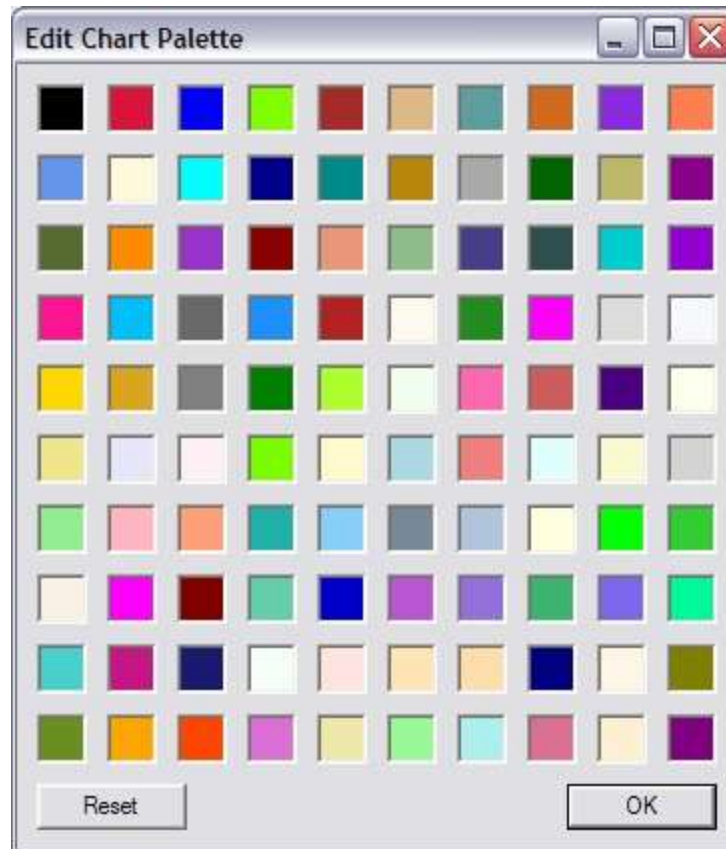


Figure 2 - The GUI used to change a graph's palette

**Palette Entries:** When the user double-clicks on a colored block in this GUI they are shown a standard Windows color choosing dialog box. The color that they choose in that dialog will be assigned to the palette entry that they clicked on. The entries are numbered beginning with 0 in the top-left corner of the GUI. The entries are then numbered along the rows. So entry 1 is the red square next to the black one in Figure 2. Entry 2 is blue, 3 is green, and so on.

**Reset:** When the user presses this button, all one hundred entries are reset to their default colors. The colors shown in Figure 2 are the defaults.

**OK:** When the user presses this button, the GUI is closed and the colors in the palette are applied to the graph.

### Graph Editing GUI

The Graph Editing GUI is a large, complicated GUI that allows the user to customize nearly everything about the graph (see Figure 3). Certain features of this GUI may be overridden by the standard behavior of the graphs. However, many things can still be customized. The most common features to change are the legend and the background. This GUI can also be used to print the graph. When the user saves a Flexsim Chart file (.fsc file) the settings that they choose in this GUI can be saved if the user wants. If they choose to save these settings, the saved file will be considerably larger than if they do not.

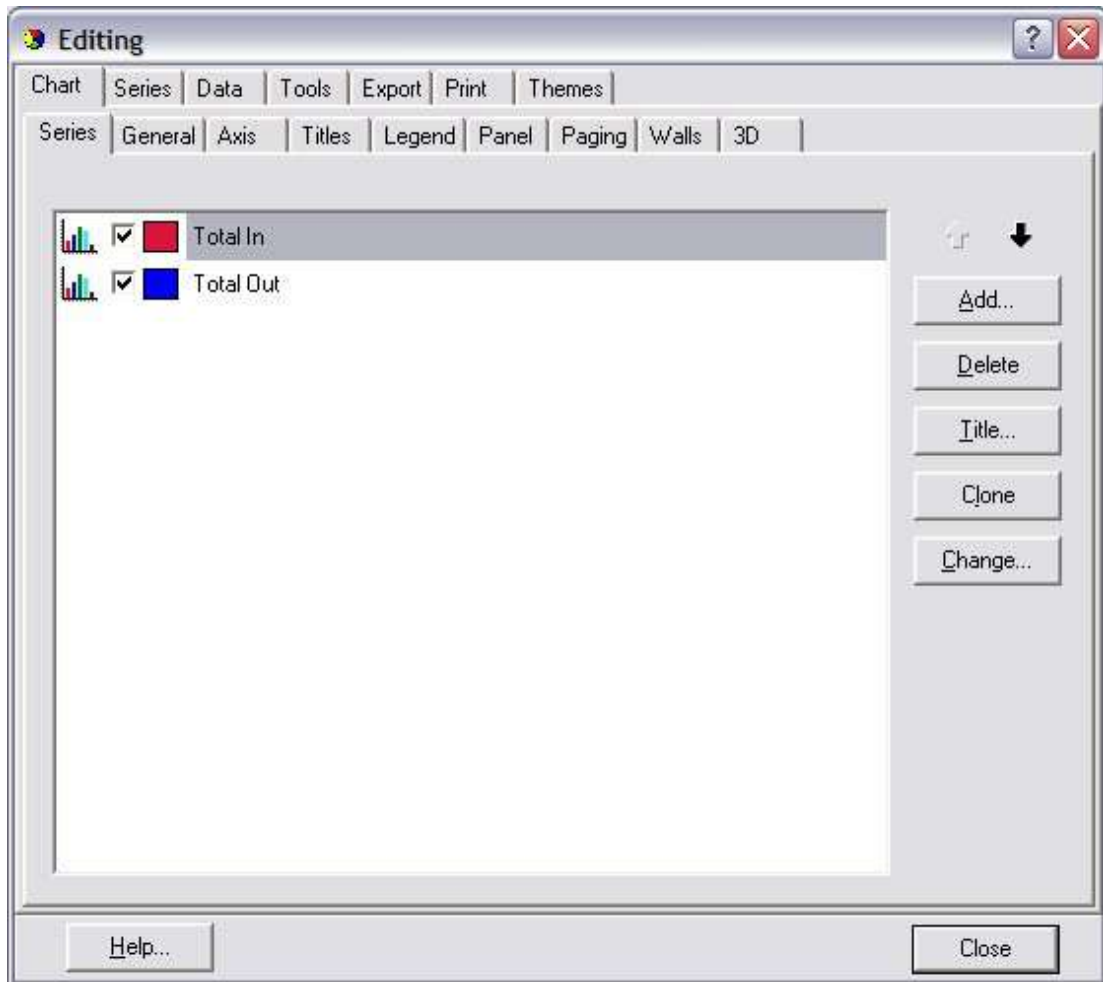


Figure 3 - The Graph Editing GUI

**Changing the Legend:** The position and appearance of the graph's legend can be changed by selecting the Legend sub-tab from the Chart main tab. This gives the user the ability to move the legend. They can also assign a background color, gradient or image to the legend. See Figure 4.

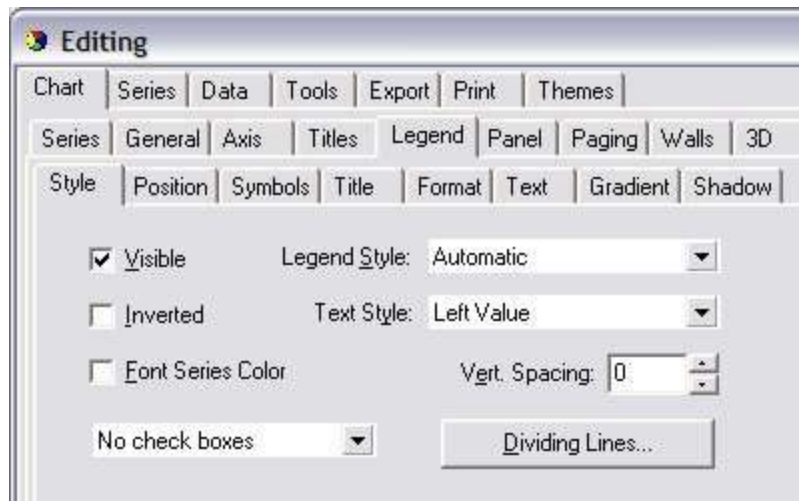


Figure 4 - Editing the legend of a graph

**Changing the Background:** The user can assign a background color or image by selecting the Panel sub-tab from the Chart main tab. See Figure 5.

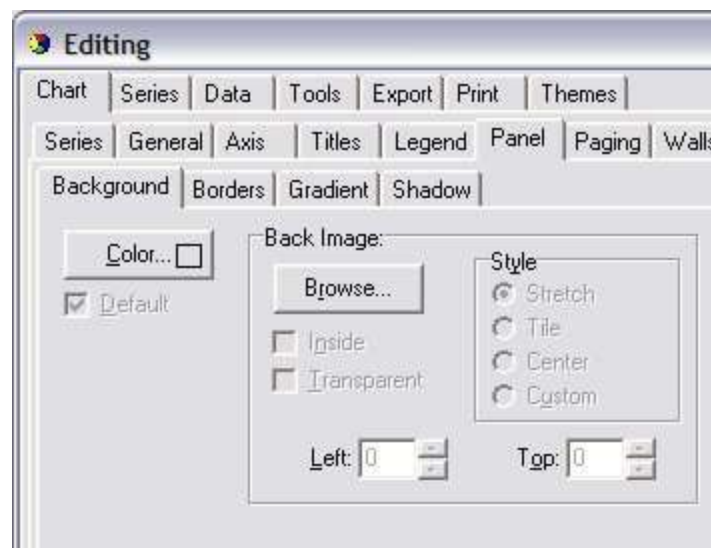


Figure 5 - Editing the background of a graph

**Printing the Graph:** The user can print the current graph by selecting the Print main tab. See Figure 6.

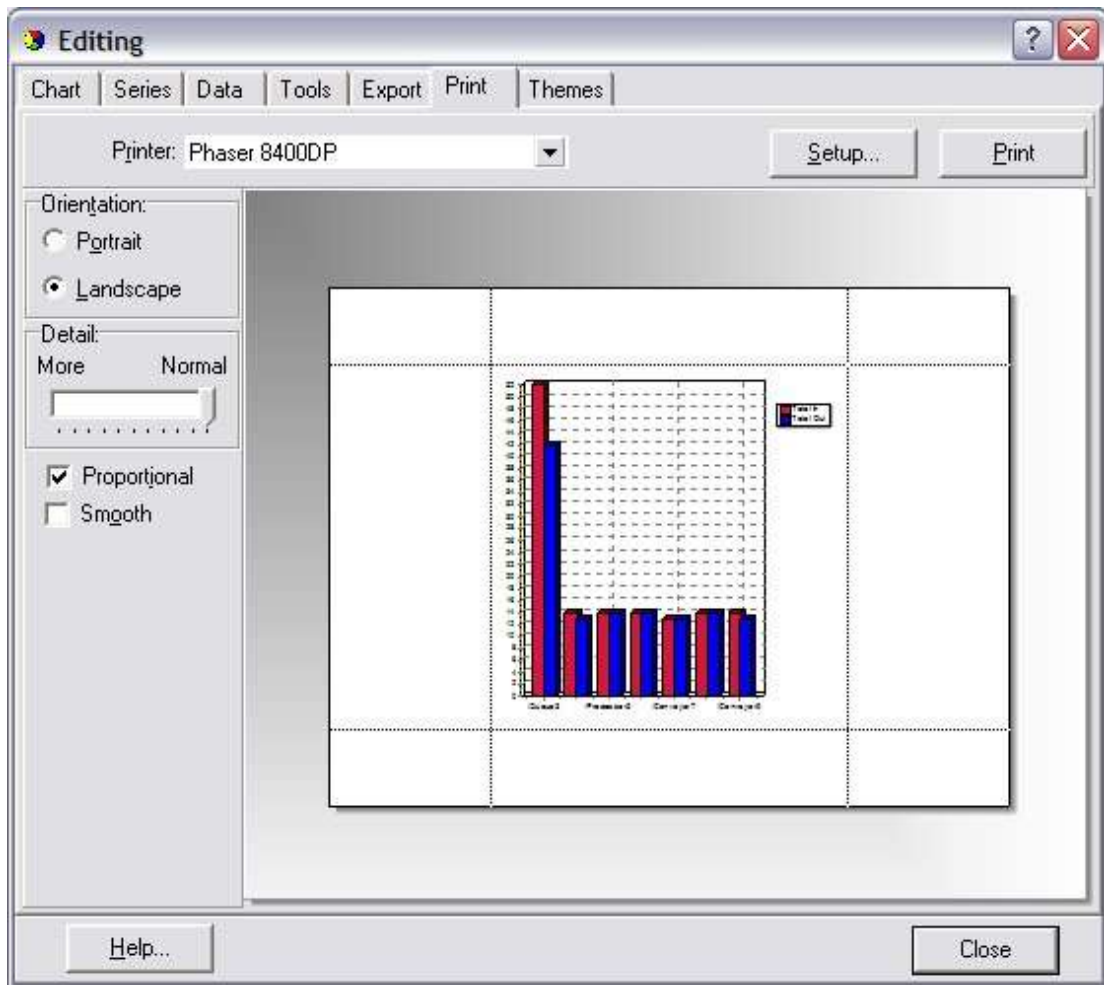


Figure 6 - Printing the current graph



# Flexsim Coding

## Writing Logic in Flexsim

In Flexsim, the full power of C++ is available to you, but as a Flexsim modeler you will only need to know a few commands to create very complex models.

### Where to get help

Whenever you need help with what commands to use and how to use them, you can refer to the Command Summary or the Command Hints window. These windows each allow you to quickly get the information you need.

### General Rules

Here are some general rules you will need to know when creating your own logic.

- language is case sensitive (A is **not** the same as a)
- no specific format is required (free use of spaces, tabs and line returns is encouraged)
- numbers are double precision floating point values unless otherwise specified.
- text strings are usually entered between quotes. "mytext"
- parenthesis follow a function call and commas separate the arguments of the function. moveobject(object1,object2);
- a function or command will always end with a semi-colon
- parenthesis can also be used freely to make associations in your math and logic statements.
- curly braces are used to define a block of statements.
- to comment out the rest of a line use //
- don't use spaces or special characters in name definitions (\_ is ok).
- named variables and explicit values can be interchanged in writing expressions.

### Math Operations

The following list show different math operations that can be performed on values.

Operation	Floating Point Example (=solution)	Integer Example (=solution)
+	1.6+4.2 (=5.8)	2+3 (=5)
-	5.8-4.2 (=1.6)	5-2 (=3)
*	1.2 * 2.4 (=2.88)	3*4 (=12)
/	6.0/4.0 (=1.5)	20/7 (=2)
% (integer mod)		34%7(=6)
sqrt()	sqrt(5.3) (=2.3)	
pow()	pow(3.0,2.2) (=11.2)	pow(3,2) (=9)
round()	round(5.6) (=6)	
frac()	frac(5.236) (=0.236)	
fabs()	fabs(-2.3) (=2.3)	
fmod() (floating point mod)	fmod(5.3,2) (=1.3)	

Be aware as you write your logic that, by default, all values in Flexsim are double precision floating point, so you will usually be using the operations as they apply to floating point numbers.

**Note:** By performing operations on floating point numbers, some precision may be lost.

**Note:** Be careful in using these operations while mixing integer types with floating point types, or with using just integer types. For example, the / operator will return an integer if both operators are integers. This may not be what you want to get out of the operation, in which case you will need to use floating point types instead of integer types. Note also that C++ will interpret the literal number 5 as an integer type. If you want it to interpret the number as a floating point type, enter 5.0 instead of just 5.

## Comparing Variables

The following table shows different operations for comparing two values or variables.

Operation	Example (solution)
> (greater than)	1.7>1.7 (false)
< (less than)	-1.7 < 1.5 (true)
>= (greater than or equal to)	45 >= 45 (true)
<= (less than or equal to)	45 <= 32 (false)
== (equal to)	45 == 45 (true)
!= (not equal to)	45 != 35 (true)
comparetext()	comparetext(getname(current), "Processor5")

**Warning:** The == operator can often cause problems if you are comparing two double precision floating point values, and one or both of those values have been calculated using math operations. When doing math operations, floating point values may lose some precision. Since the == operator will only return true if all 64 bits of each value are exactly the same, even a small precision loss will cause the == operator to return false. In such cases, you will want to instead verify that the two values are within a range of each other. For example: fabs(value1 - value2) < 0.000001, will return true if the two values are practically equal for all intents and purposes.

## Relating Variables

The following table shows different operations for relating several comparisons.

Operation	Example
&& (logical AND)	x>5 && y<10
(logical OR)	x==32    y>45
! (logical NOT)	!(x==32    y>45)
min()	min(x, y)
max()	max(x, y)

## Setting and Changing Variables

The following tables show ways of setting and changing variables.



Operation	Example
=	x = x + 2;
+=	x += 2; (same as x = x + 2)
-=	x -= 2; (same as x = x - 2)
*=	x *= 2; (same as x = x * 2)
/=	x /= 2; (same as x = x / 2)
++	x ++; (same as x = x + 1)
--	x --; (same as x = x - 1)

### Variable Types

In Flexsim, you will only need to use 4 types of variables. The following explains each of these types.

int	integer type
double	double precision floating point type
string	text string
treenode	reference to a Flexsim node or object
intarray	an array of integer types
doublearray	an array of double types
stringarray	an array of string types
treenodearray	an array of treenode types

For more information on how the treenode (or Flexsim node) type works, refer to the Flexsim tree structure.

### Declaring and Setting Variables

The following are some examples of how to declare and set variables.

```
int index = 1;
double weight = 175.8;
string category = "groceries?";
treenode nextobj = next(current);
```

### Declaring and Setting Array Variables

The following are examples of how to use array types.

```
intarray indexes = makearray(5); // makes an array with 5 elements
indexes[1] = 2; // in Flexsim, arrays are 1-based
indexes[2] = 3;
```

```
indexes[3] = 2;
indexes[4] = 6;
indexes[5] = 10;
```

```
doublearray weights = makearray(3);
fillarray(weights, 3.5, 6.7, 1.4); // fillarray is a quick way of setting the array values
```

```
stringarray fruits = makearray(2);
fruits[1] = "Orange";
fruits[2] = "Watermelon";
```

```
treenodearray operators = makearray(4);
operators[1] = centerobject(current, 1);
operators[2] = centerobject(current, 2);
operators[3] = centerobject(current, 3);
operators[4] = centerobject(current, 4);
```

## Executing Commands

Executing a command in Flexsim is made of following parts. First type command's name, followed by an open parenthesis. Then enter each parameter of the command, separating multiple parameters by commas. Each parameter can be a variable, an expression of variables, or even a command itself. Finish the command with a close parenthesis, and a semi-colon. For detailed information on the commands, their functionality and parameter lists, refer to the command summary. For a quick reference of the most used commands in Flexsim, refer to the section on basic modeling functions.

Syntax	Examples
commandname( parameter 1, parameter 2, parameter 3 ...);	coloryellow(current); setrank(item, 3 + 7); setitemtype(item, getlabelnum(current, "curitemtype"));

## Flow Constructs

The following are constructs which allow you to modify the flow of your code.

### Logical If Statement

The if statement allows you to execute one piece of code if an expression is true, and another piece of code if it is false. The else portion of the construct is optional.

Construct	Example
if (test expression) { code block } else { code block	if (content(item) == 2) { colorred(item); } else { colorblack(item);

}	}
---	---

### Logical While Loop

The while loop will continue to loop through its code block until the test expression becomes false.

Construct	Example
<pre>while (test expression) {     code block }</pre>	<pre>while (content(current) == 2) {     destroyobject(last(current)); }</pre>

### Logical For Loop

The for loop is like the while loop, except that it is used usually when you know exactly how many times to loop through the code block. The start expression is executed only once, to initialize the loop. The test expression is executed at the beginning of each loop and the loop will stop as soon as this expression is false, just like the while loop. The count expression is executed at the end of each loop, and typically increments some variable, signifying the end of one iteration.

Construct	Example
<pre>for (start expression;     test expression;     count expression) {     code block }</pre>	<pre>for (int index = 1;     index &lt;= content(current);     index++) {     colorblue(rank(current,index)); }</pre>

### Logical Switch Statement

The switch statement allows you to choose one piece of code to execute out of several possibilities, depending on a variable to switch on. The switch variable must be an integer type. The example below sets the color of items of type 1 to yellow, type 5 to red, and all other types to green.

Construct	Example
<pre>switch ( switchvariable ) {     case casenum:     {         code block         break;     }     default:     {         code block         break;     } }</pre>	<pre>int type = getitemtype(item); switch (type) {     case 1:     {         coloryellow(item);         break;     }     case 5:     {         colorred(item);         break;     }     default:</pre>

	<pre>{   colorgreen(item);   break; }</pre>
--	---

## Basic Modeling Functions and Logic Statements

Here we have provided a quick reference for commands that are used most often in Flexsim. For more detailed information on these commands, refer to the command summary.

### Object Referencing

The following commands and access variables are used in referencing objects in Flexsim.

#### current and item

- **current** - the current variable is a reference to the current resource object. It is often an access variable in pick lists.
- **item** - the item variable is a reference to the involved item for a trigger or function. It is often an access variable in pick lists.

#### Referencing commands

command(parameter list)	Explanation	Example
first(node)	This returns a reference to the first ranked object inside of the object passed	first(current)
last(node)	This returns a reference to last ranked object inside of the object passed	last(current)
rank(node,ranknum)	This returns a reference to the object at a given rank inside the object passed	rank(current,3)
inobject(object,portnum)	This returns a reference to the object connected to the input port number of the object passed	inobject(current,1)
outobject(object,portnum)	This returns a reference to the object connected to the	outobject(current,1)

	output port number of the object passed	
centerobject(object,portnum)	This returns a reference to the object connected to the center port number of the object passed	centerobject(current,1)
next(node)	This returns a reference to the next ranked object of the object passed	next(item)

### Object Attributes

command(parameter list)	Explanation
getname( object )	This returns the name of the object
setname( object, name )	This sets the name of the object
getitemtype( object )	This returns the itemtype value of the object
setitemtype( object, num)	This sets the itemtype value of the object
setcolor( object, red, green, blue )	This sets the color of the object
colored( object ) blue,green,white...	This sets the color of the object to red, blue, green, white, etc.
setobjectshapeindex ( object , indexnum )	This sets the 3D shape of the object
setobjecttextureindex ( object , indexnum )	This sets the 3D texture of the object
setobjectimageindex ( object , indexnum )	This sets the 2D texture of the object. This usually only applies if you are using the planar view.

### Object Spatial Attributes

command(parameter list)	Explanation
xloc( object ) yloc( object ) zloc( object )	These commands return the x,y, and z locations of the object
setloc( object, xnum, ynum, znum )	This sets the x, y, and z location of the object
xsize( object ) ysize( object ) zsize( object )	These commands return the x,y, and z size of the object
setsize( object, xnum, ynum, znum )	This sets the x, y, and z size of the object

xrot( object ) yrot( object ) zrot( object )	These commands return the x,y, and z rotation of the object
setrot( object, xdeg, ydeg, zdeg )	This sets the x, y, and z rotation of the object

### Object Statistics

command(parameter list)	Explanation
content( object )	This returns the current content of the object
getinput( object )	This returns the input statistic of the object
getoutput( object )	This returns the output statistic of the object
setstate( object, statenum )	This sets the current state of the object.
getstatenum( object )	This returns the current state value of the object
getstatestr( object )	This returns the current state of the object as a string
getrank( object )	This returns the rank of the object
setrank( object,ranknum )	This sets the rank of the object
getentrytime( object )	This returns the time the object entered the object it is currently in
getcreationtime( object )	This returns the time the object was created

### Object Labels

command(parameter list)	Explanation
getlabelnum( object, labelname ) getlabelnum( object, labelrank)	This returns the value of the object's label
setlabelnum( object, labelname , value ) setlabelnum( object, labelrank , value )	This sets the value of the object's label
getlabelstr( object, labelname )	This gets the string value of the object's label
setlabelstr( object, labelname , value ) setlabelstr( object, labelrank , value )	This sets the string value of the object's label
label( object, labelname ) label(object, labelrank)	This returns a reference to the label as a node. This command is often used if you have a label that is used as a table.

### Tables

<b>command(parameter list)</b>	<b>Explanation</b>
gettablenum( tablename / tablenode / tablerank, rownum, colnum )	This returns the value in the specified row and column of the table
settablenum( tablename / tablenode / tablerank, rownum, colnum, value)	This sets the value in the specified row and column of the table
gettablestr( tablename / tablenode / tablerank, rownum, colnum )	This returns the string value in the specified row and column of the table
settablestr( tablename / tablenode / tablerank, rownum, colnum, value)	This sets the string value in the specified row and column of the table
settablesize( tablename / tablenode / tablerank, rows, columns )	This sets the size of the table in rows and columns
gettablerows( tablename / tablenode / tablerank)	This returns the number of rows in the table
gettablecols( tablename / tablenode / tablerank)	This returns the number of columns in the table
clearglobaltable( tablename / tablenode / tablerank)	Sets all number values in the table to 0

### Object Control

<b>command(parameter list)</b>	<b>Explanation</b>
closeinput( object )	This closes the input of the object
openinput( object )	This re-opens the input of the object
closeoutput( object )	This closes the output of the object
openoutput( object )	This re-opens the output of the object
sendmessage( toobject, fromobject, parameter1, parameter2, parameter3 )	This causes the message trigger of the object to fire
senddelayedmessage( toobject, delaytime, fromobject, parameter1, parameter2, parameter3 )	This causes the message trigger of the object to fire after a certain delay time
stopobject( object, downstate )	This tells the object to stop whatever its operation is and go into the given state
resumeobject( object )	This allows the object to resume whatever its operation is
stopoutput( object )	This closes the output of the object, and accumulates stopoutput requests
resumeoutput( object )	This opens the output of the object once all stopoutput requests have been resumed



stopinput( object )	This closes the input of the object, and accumulates stopinput requests
resumeinput( object )	This opens the input of the object once all stopinput requests have been resumed
insertcopy( originalobject, containerobject )	This inserts a new copy of the object into the container
moveobject( object, containerobject )	This moves the object out of its current container into its new container

## Advanced Functions

### Object Variables

command(parameter list)	Explanation
getvarnum( object, "variablename" )	This returns the number value of the variable with the given name
setvarnum( object, "variablename" , value )	This sets the number value of the variable with the given name
getvarstr( object, "variablename" )	This returns the string value of the variable with the given name
setvarstr( object, "variablename" , string )	This sets the string value of the variable with the given name
getvarnode( object, "variablename" )	This returns a reference to the variable with the given name as a node

### TaskExecuter Control

For more information on controlling TaskExecuters, refer to the task sequence section.

### Prompts and Printouts

command(parameter list)	Explanation
pt( text string )	Prints text to the output console
pf( float value )	Prints a floating point value to the output console
pd( discrete value )	Prints an integer value to the output console
pr( )	Creates a new line in the output console
msg( "title" , "caption" )	Opens a simple Yes, No, Cancel dialog
userinput( targetnode, "prompt" )	Opens a dialog box where you can set the value of a node in the model
concat( string1, string2, etc. )	This returns the string concatenation of two or more strings

## More Advanced Functions

Here are more advanced functions that you might use. We do not provide their parameter lists here. For more information, refer to the command summary.

**Node commands** - node(), nodeadddata(), getdatatype(), nodetopath(), nodeinsertinto(), nodeinsertafter(), getnodename(), setnodename(), getnodenum(), getnodestr(), setnodenum(), setnodestr(), inc()

**Data changing commands** - stringtonum(), numtostring(), tonum(), tonode(), apchar()

**Node table commands** - setsize(), cellrc(), nrows(), ncols()

**Node table commands** - setsize(), cellrc(), nrows(), ncols()

**Model run commands** - cmdcompile(), resetmodel(), go(), stop()

**3D custom draw code commands** - drawtomodelscale(), drawtoobjectscale(), drawsphere(), drawcube(), drawcylinder(), drawcolumn(), drawdisk(), drawobject(), drawtext(), drawrectangle(), drawline(), spacerotate(), spacetranslate(), spacescale()

**Excel commands** - excellaunch(), excelopen(), excelsetsheet(), excelreadnum(), excelreadstr(), excelwritenum(), excelwritestr(), excelimportnode(), excelimporttable(), excelclose(), excelquit()

**ODBC commands** - dbopen(), dbclose(), dbsqlquery(), dbchangetable(), dbgetmetrics(), dbgetfieldname(), dbgetnumrows(), dbgetnumcols(), dbgettablecell(), dbsettablecell()

**Kinematics** - initkinematics(), addkinematic(), getkinematics(), updatekinematics(), printkinematics()

# 3D Media

## Importing 3D Media

Flexsim can import several types of 3D media. These include 3ds, wrl, dxf, and stl file types. You can import media using two methods. First, and most common, you can import media by selecting a 3D file from an object's visual properties page. Second, if there are some 3D media files that are not used by any model objects by default, but you want to change their 3D representation dynamically during the simulation run to use these 3D media files, you can use the media importer to import media files explicitly.

The following are some hints and suggestions for importing media correctly.

**Note on importing wrl files:** Flexsim will only import VRML version 1.0 shapes, not version 2.0.

**Note on importing stl files:** Flexsim will only import stl ascii files, not stl binary files.

### Using Shape Factors

Each media file that is imported has certain scaling and offset settings which may cause the 3D model that you import to not fit within the object's boundaries. In this case, edit the object's 3D shape factors from its visual properties window to fit the 3D shape within the object's yellow bounding box.

### Letting Color Show Through the 3D Object

Often, when you design your own 3D files or get them from the internet, they will have materials and colors predefined with the 3D model. If materials are defined on a 3D object that is imported into Flexsim, those materials will show through, instead of the object's color defined in Flexsim. If you want the object's Flexsim color to show through the 3D object, you will need to change the ordering of the 3D file accordingly. In the 3D file, the polygons whose Flexsim color shows through must be defined before any material is defined in the file. This can be tricky to do, and depends on the 3D modeling program that you are using.

### 3D Frames

You can animate an object using 3D frames. For any 3D file, you can specify frames of that 3d file by creating different 3d models, and saving them as <original file name>FRAME<frame number>.3ds. For example, the operator's primary 3d file is Operator.3ds. This is drawn if its frame is set to 0 using the command setframe(current, 0). Its other frames are defined in OperatorFRAME1.3ds, OperatorFRAME2.3ds, etc. If you call setframe(current, 3), then the operator will draw the 3D file specified in <original filename>FRAME3.3ds, etc.

### Level of Detail

You can specify several levels of detail for a given 3D file. This can significantly increase the speed of drawing the model. As you get further away from objects, they are drawn with less detail, thus increasing speed.

## Preparing a 3D File

The following steps should be performed when preparing a 3D file for Import:

### Reduce the Number of Polygons

3ds and wrl files typically include more information than is necessary. Removing excess polygons will improve the visual performance of your model. As a result, it will be easier to build and present the model.



### Textures vs. Polygons

The use of well made textures can help a modeler reduce the number of polygons required to make a realistic looking object. Below are some pictures of low polygon 3D files in Flexsim.



### Adjust the Scale

3D files are not necessarily drawn in feet or meters and may need to be rescaled to work appropriately in Flexsim. There are three ways to adjust the scale of a 3D file:

1. Appropriately scale the file in the 3D program.
2. Scale the visual tool or other object that the file is imported into.
3. Use an xds or xrl file.

If you are going to scale the visual tool or object that the file is imported into, then you may want to scale the 3D object to 1,1,1.

### Move Objects to the Origin

3D drawings are sometimes drawn using a specific coordinate system. This usually means that the objects are not located near the origin (0,0,0). Importing a 3ds or wrl file where the objects are not located at the origin will usually result in not being able to see the imported shapes. 3D objects will need to be moved to the origin or they will need to be adjusted using an xds or xrl file.

### XDS / XRL Files

xds / xrl files are used to make imported 3d shapes conform to the object they are imported into. xds / xrl files must have the same names as the objects they modify:

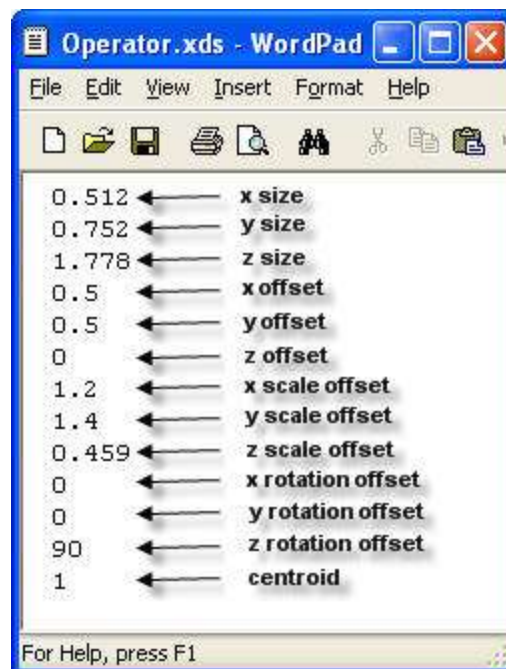
- crane.3ds – crane.xds
- crane.wrl – crane.xrl

### xds / xrl file info

Spatial values determine the end size of the 3d shape. Offset values are the values required to get the 3d shape zeroed out and sized to 1,1,1. The centroid value is 1 or 0 and determines if the object rotates around the center of the object or the top left corner.

### xds / xrl file makeup

The xds/xrl file is a text file made up of 13 values separated by carriage returns. You can edit this file using notepad or wordpad, as shown below.



### YDS / YRL Files

yds / yrl files are used to make imported 3d shapes look good. yds / yrl files must have the same names as the objects they modify:

- crane.3ds – crane.yds

- crane.wrl – crane.yrl

### yds / yrl file info

The yds/yrl file is a text file made up of 3 values separated by carriage returns. You can edit this file using notepad or wordpad, as shown below.



1. The luminous value determines whether the 3d shape will be affected by Flexsim's lighting.
2. The backfacecull value determines whether Flexsim will draw both sides of each polygon or not. If the imported shape looks backwards or odd, set this value to 1.
3. The creaseangle value is the angle at which Flexsim will stop smoothing the corners of the 3d shape.

### Importing a 3d File into Flexsim

To import a 3d file into Flexsim follow these steps:

1. Create a Visual Tool.
2. Select "Imported Shape" for the Visual Display.
3. Select the 3D file you want to import in the "Filename" field.
4. Set the minimum magnification (usually 0 unless you want the drawing to disappear at a certain magnification).
5. Set the maximum distance (usually 100000 unless you want the drawing to disappear when you move the view away from the drawing).

To change the shape of an existing object follow these steps:

1. Go to the Properties page of an object.
2. Select the 3D file you want to import in the "3D shape" field.



## Importing AutoCAD Drawings

The following steps should be performed when preparing an AutoCAD .dxf file for import:

1. **Remove all unnecessary information.** AutoCAD files typically include much information that is unnecessary to the simulation. Typically, all a simulation needs is a basic layout. Removing information that is extraneous to the simulation will make your model more clear and reduce the burden on your graphics card. As a result it will be easier to build and present the model. Remove any parts of the drawing that are not pertinent to the simulation study.
2. **Adjust the scale to Flexsim units.** AutoCAD files are often scaled in inches. Flexsim models are often scaled in feet or meters. It is important that the AutoCAD file be rescaled to work appropriately in Flexsim. For instance, to convert from an AutoCAD file in inches to a Flexsim model in feet, the scale factor will be 1/12. To determine how much to scale, follow these steps: Measure a known distance in AutoCAD ("\_dist"). Apply the following equation: scale factor = Flexsim distance / ACAD distance. To scale objects in AutoCAD follow these steps: Select the objects you want to scale. Type "\_scale" in the command prompt or select the scale command from the menus. Specify a reference point. Type in the calculated scale factor in the command prompt.
3. **Move objects to the origin.** AutoCAD drawings are usually drawn using a specific coordinate system. This usually means that the objects are not located near the origin (0,0,0). When a .dxf file is imported into Flexsim, it is positioned in Flexsim's coordinate system according to the .dxf's positioning. So if the origin point of your AutoCAD file is very far away from the actual drawing, when that .dxf file is imported into Flexsim, the layout will also be very far away from the model's origin position in Flexsim. This can be quite frustrating for the modeler. For this reason, move your AutoCAD objects to the origin. Do so by completing the following steps:
  1. Select the objects you want to move.
  2. Type "\_move" in the command prompt or select the move command from the menus.
  3. Specify a reference point.
  4. Type in the desired location of that point in the command prompt.
4. **Explode compound objects.** Flexsim can only import basic shapes, so it is important to explode any compound objects in the AutoCAD drawing into their basic shapes. To explode compound objects in AutoCAD follow these steps:
  1. Select the objects you want to explode.
  2. Type "\_explode" in the command prompt or select the explode command from the menus.
  3. Repeat the above steps until there are no objects that were exploded.

In Flexsim:

1. Drag a VisualTool object into the model. Open its Parameters window.
2. Select "Imported Shape" for the Visual Display.
3. Select the AutoCAD file you want to import in the "Filename" field.



4. Set the minimum magnification to 0 on the visual tool into which you are importing the media.
5. Set the maximum distance (usually 100000 unless you want the drawing to disappear when you move the view away from the drawing).

## Using Frames

Frames are different 3D shapes that are tied together by a common name.

file.3ds / fileFRAME1.3ds

If an object uses a 3D shape that has frames then the shape of that object displays can be changed by using the `setframe()` command.

- 0 is the base frame “file.3ds”
- 1 is the first frame “fileFRAME1.3ds”
- 2 is the second frame “fileFRAME2.3ds”
- N is the nth frame “fileFRAMEn.3ds”

Frames can be used to:

- Display Different states on a fixed resource.
- Display Different stages of a flowitems life.
  - Raw material
  - Intermediate Product
  - Finished Product
- Simplify the shape changing process anytime you will need to display different 3D shapes for one object.

### Preparing Frames

To prepare a group of 3D Shapes for use as Frames do the following:

1. Gather together two or more shapes that you want to use as frames of the same object.
2. Get the frames saved as the same file type (3ds, wrl).
3. Make sure that each frame uses or at least isn't disfigured by the base file's .xds file (each frame can have it's own .yds file).
4. Name the frame files appropriately. The first file is the “name.3ds” or “name.wrl”. Each successive file is “nameFRAME#.3ds” or “nameFRAME#.wrl”. For example: Queue.3ds, QueueFRAME1.3ds, QueueFRAME2.3ds, etc.

To import frames just follow the steps used in the Importing 3D Media section. Use the base object for the name of the file that you select. Then, in the simulation, use the `setframe()` and `getframe()` commands to set which frame you want to show. 0 is the base frame: file.3ds, 1 is the first frame: fileFRAME1.3ds, etc.

## Level Of Detail (LOD)

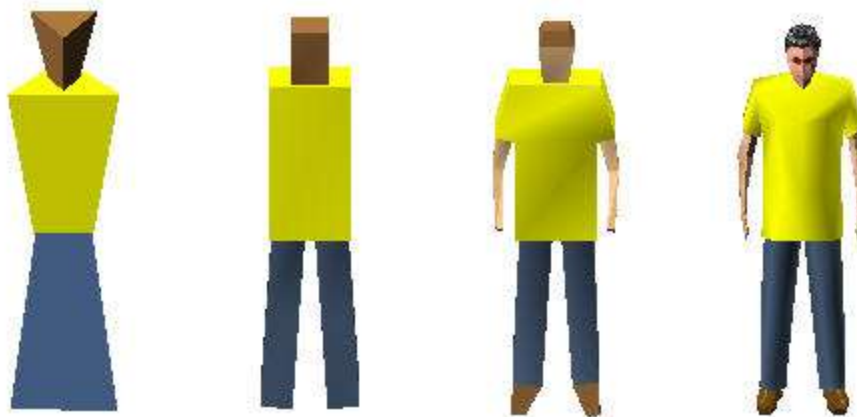
LOD files are 3D shapes that get progressively simpler and that are tied together by a common name:

file.3ds / fileLOD1.3ds

If an object uses a 3D shape that has LOD then the 3D shape that the object displays is dependent on how far that object is from the view's camera position.

- 0 is the base LOD “file.3ds”
- 1 is the 2nd LOD “fileLOD1.3ds”
- 2 is the 3rd LOD “fileLOD2.3ds”
- N is the nth LOD “fileLODn.3ds”

LOD Files are used to show higher level objects when the viewer is close up and lower level objects when the viewer is farther away. LOD improves the speed at which a model will display in the view window. LOD allows for the use of more polygon intensive 3D shapes by displaying only a few high polygon count shapes at any given time.



### Preparing LOD Files

To prepare a group of 3D Shapes for use as an LOD file do the following:

1. Gather together two or more shapes that you want to use as LODs of the same object.
2. Get the LOD files saved as the same file type (3ds, wrl).
3. Make sure that each LOD uses or at least isn't disfigured by the base file's .xds file (each frame can have it's own .yds file).
4. Name the LOD files appropriately. The first file is the “name.3ds” or “name.wrl”. Each successive file is “nameLOD#.3ds” or “nameLOD#.wrl”. For example, Queue.3ds, QueueLOD1.3ds, QueueLOD2.3ds, etc.

To import and LOD shape just follow the steps used in the section on Importing 3D Media. Use the base object for the name of the file that you select.

## LDS/LRL Files

lds / lrl files are used to determine the distances at which the different LOD's will be shown. lds / lrl files must have the same names as the objects they modify:

- crane.3ds – crane.lds
- crane.wrl – crane.lrl

### lds / lrl file info

The lrl/lds file is a text file made up of three numbers separated by new lines: The range perspective value, the range perspective mode, and the range ortho value.



The range perspective value is the distance value at which the LOD files will change in the perspective window. The range perspective mode value is either 1 or 0. 0 means linear (slightly faster). 1 means inverse distance (more coverage at farther distances). The range ortho value is the distance value at which the LOD files will change in the ortho window.

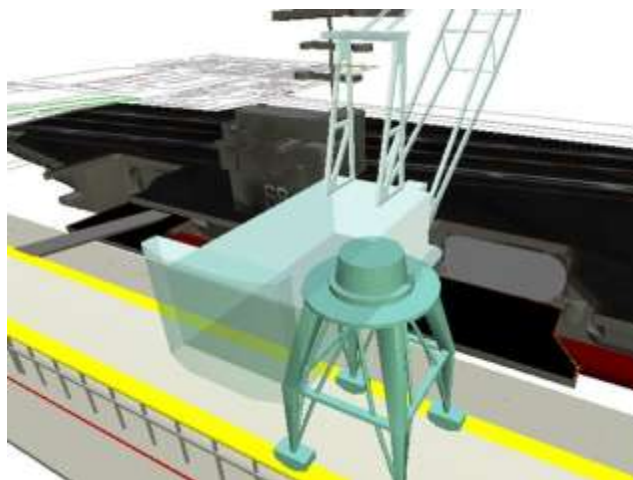
## Transparency

You can add transparency to your 3D objects by using .tmp and .tpg file types. .tmp and .tpg files are grey scale files that determine the transparency of a corresponding .bmp or .jpg file.

For the .tmp or .tpg files white is opaque and black is transparent with all of the shades of grey in between being semi-transparent. The name of the texture and transparent file must correspond to the original .bmp or .jpg file, and it must be in the same directory as well. For example, if the texture Flexsim.bmp is used, and there is also a Flexsim.tmp file in the same directory, then the .tmp file will be used to specify transparency in the texture.



**Note on object rank and transparency:** The rank of an object has a direct effect on the way the transparency of that object will look. All of the objects that are ranked higher in the model than the transparent object will not appear behind the transparent object.



model	Object
Tools	
Carrier Building	Object
Carrier Flight Deck	Object
Carrier Hull	Object
Crane Top	Object
Crane Bottom	Object
Pier 3D	Object
Yard ACAD	Object
Ramp A	Object
Ramp B	Object



# Miscellaneous Concepts

## Introduction to Flexsim Tree Structure

Flexsim is completely designed around the concept of a tree structure. All information in Flexsim is contained within the Flexsim tree, including the library objects, commands, and all model information. This tree hierarchy is made of individual nodes that link together and hold information.

### Nodes

A node is the building block of a Flexsim tree. All nodes have a text containing the name of the node. Nodes can simply be a container for other nodes, can be a keyword used to define an attribute of an object, or may have a data item.


The data item types which may be attached to a node are: number, string, object, or pointer. To attach data to a node, right-click on the node and go to the Insert menu option. You will see the four options to add data to a node. There are also shortcut keys for adding number, string(text), object, or pointer data. These are the keys N, T, O, and P. To add data to a node using a shortcut key, click on the node, then press the appropriate key. Nodes can also hold executable code. To make a node executable, first add string data to the node, and then toggle the node as either a C++ or a Flexscript node. To toggle a node as one of these types, right-click on the node and go to the Build menu.

The symbols for the different types of nodes are shown here:

Standard: 


Object: 

Attribute/Variable: 


Function (C++): 

Function (FlexScript): 

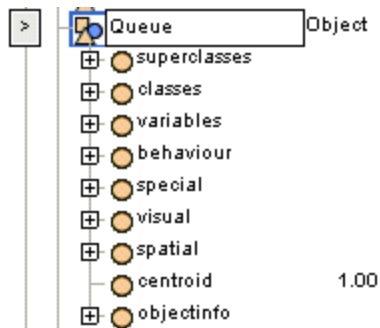
Nodes can be added and deleted from the tree. To delete a node, simply click the node and then hit the delete key. To insert a node, right-click on an existing node and choose Edit | Insert. This will add a new node immediately after the node that was clicked on. The shortcut for this operation is to hit the spacebar after first highlighting a node.

Nodes can also contain a sub list of nodes called the content branch. If a node contains sub nodes it can be expanded by pressing the  button. To insert a node into the content of an existing node, choose the option Edit | Insert Into, or hit the Enter key as a shortcut.

A node that has object data may contain a second sub list of nodes that are contained in a separate branch of the tree. This sub list of nodes is called the object attribute tree, and contains data that describes the properties of the object. A node containing object data may typically be referred to as an object node. When you

click on an object node you will see a greater than symbol  to the left of the node. Clicking on this button will open the object attribute tree branch.

The following picture shows an expanded object attribute tree for the Queue object in the library tree.



For nodes with object data, the attribute tree can contain many special attribute nodes. If a node is inside an object and has the name of a key attribute, it will have a special meaning to the object. The actual meaning of the attribute depends on what the attribute is and the object type. As an example, there are attributes for an object's position: 'spatialx', 'spatialy', 'spatialz'. The list of available attributes in Flexsim is found in attribute hints.

In addition to containing all model, library, and project information, the Flexsim tree also stores all windowing and interface information. All open windows, menus, toolbars and buttons have a corresponding representation in the Flexsim tree. We call these types of nodes view objects.

## General Organization Trees

Flexsim's root tree structure is split up into two parts. These are the Project Tree and the View Layout Tree.

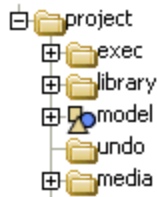
The project tree contains the Executive data, the Library, and the Model.

The view layout tree contains information on windows, editors, and other user interfaces. It also manages active windows.

## Project Tree:

To view the project tree, click on the main menu option: View | System Info | MAIN tree. The project tree holds projects at its most basic level. Each project holds the following crucial sub trees: exec, library, model, undo, media.





### Exec

This tree contains simulation executive data. This includes the simulation time, the eventlist, as well as other information with running a model.

### Library

The library of objects used by the model.

### Model

The simulation model.

### Undo

Holds undo history. A numerical value for this node is the limit on the number of undo steps. If there is no numerical data, undo will be disabled. Undo functionality may also be globally disabled.

### Media

Stores Images, 3D models and Sounds.

### View Layout Tree:

To view the project tree, click on the main menu option: View | System Info | VIEW tree. The view tree contains data for creating, storing, and using graphical user interfaces for objects.



## When To Compile Flexsim

In general, Flexsim will notify you when you need to compile. If you have set your global preferences to use flexscript code by default, then you should never have to compile unless you open a model that was built using C++. However, if you set your preferences to use C++ by default, the following actions will require a compile before running.

- Opening a project or session from the File menu.
- Opening a model that contains any C++ code.
- Creating a new object in the model. This can be done by dragging a new object from the library icon grid into an orthographic or perspective view, or into a planar view. This also can be done by duplicating the selected objects in the model, or creating an object from a user library.
- Adding tools through the Tools menu.
- Editing any picklists.
- Copying variables from a highlighted object to any set of objects.
- Writing explicit c++ code.

Whenever you perform one of the following operations, you may need to reset the model, but you should **NOT** need to compile the model.

- Editing a regular edit field like the name of an object, or the max content variable of a queue.
- Making connections between objects.
- Editing tables, lists, conveyor sections, etc.

## State List

Below is a list of state numbers and their respective macros. Whenever you write code that has to do with setting the state of objects, like using the stopobject() command or the utilize task, you can substitute these macros in for the number. Refer to the library objects for more information about what each state means to each object.

- 1 - STATE\_IDLE
- 2 - STATE\_PROCESSING
- 3 - STATE\_BUSY
- 4 - STATE\_BLOCKED
- 5 - STATE\_GENERATING
- 6 - STATE\_EMPTY
- 7 - STATE\_COLLECTING
- 8 - STATE\_RELEASING
- 9 - STATE\_WAITING\_FOR\_OPERATOR
- 10 - STATE\_WAITING\_FOR\_TRANSPORTER
- 11 - STATE\_BREAKDOWN
- 12 - STATE\_SCHEDULED\_DOWN
- 13 - STATE\_CONVEYING
- 14 - STATE\_TRAVEL\_EMPTY
- 15 - STATE\_TRAVEL\_LOADED
- 16 - STATE\_OFFSET\_TRAVEL\_EMPTY
- 17 - STATE\_OFFSET\_TRAVEL\_LOADED
- 18 - STATE\_LOADING
- 19 - STATE\_UNLOADING
- 20 - STATE\_DOWN
- 21 - STATE\_SETUP
- 22 - STATE\_UTILIZE
- 23 - STATE\_FULL
- 24 - STATE\_NOT\_EMPTY
- 25 - STATE\_FILLING
- 26 - STATE\_STARVED
- 27 - STATE\_MIXING
- 28 - STATE\_FLOWING
- 29 - STATE\_ALLOCATED\_IDLE
- 30 - STATE\_OFF\_SHIFT
- 31 - STATE\_CHANGE\_OVER
- 32 - STATE\_REPAIR
- 33 - STATE\_MAINTENANCE
- 34 - STATE\_LUNCH
- 35 - STATE\_ON\_BREAK
- 36 - STATE\_SUSPEND
- 37 - STATE\_AVAILABLE
- 38 - STATE\_PREPROCESSING
- 39 - STATE\_POSTPROCESSING
- 40 - STATE\_INSPECTING
- 41 - STATE\_OPERATING
- 42 - STATE\_STANDBY
- 43 - STATE\_PURGING

- 44 - STATE\_CLEANING
- 45 - STATE\_ACCELERATING
- 46 - STATE\_MAXSPEED
- 47 - STATE\_DECELERATING
- 48 - STATE\_STOPPED
- 49 - STATE\_WAITING
- 50 - STATE\_ACCUMULATING

## Kinematics

Kinematic functionality allows you to have a single object perform several travel operations simultaneously, each travel operation having its own acceleration, deceleration, start speed, end speed, and maximum speed properties. For example, an overhead crane usually has several motors that drive it. One motor drives the bridge along a railing, while another motor drives a trolley along the bridge, while another lifts the hook or grabber by a cable. Each of these motors may have their own acceleration, deceleration, and maximum speed properties. Having these different motors work simultaneously gives the motion of the crane a very dynamic behavior and look. Before kinematics were introduced, the simplest way to simulate this behavior was to have three different objects hierarchically ordered in the model tree, each object simulating one motion or kinematic. This, however, could often become tedious and unfriendly. Kinematic functionality attempts to fix this problem by allowing one object to do several motions or kinematics simultaneously. This help page defines the API for kinematic functionality. In the future, more functionality will be built on top of this API, such as a `tablekinematics` command that will automatically build kinematics for you by looking up values from a table, and a `TASKTYPE_KINEMATICS` task. But for now, this is the core API for kinematics.

To perform kinematic operations, you first call the `initkinematics` command. This initializes data for the kinematics, saving things like the start location and rotation of the object you want to apply motion to. Once you have initialized kinematics, you give subsequent travel/rotate operations to the object using the `addkinematic` command. For example, you can tell the object, starting in 5 seconds, to travel 10 units in the x direction, with a given acceleration, deceleration, and max speed. Then you can tell the object, starting in 7 seconds, to travel 10 units in the y direction, with a different acceleration, deceleration, and max speed. The effect of these two operations is that the object will start traveling in the x, then will start simultaneously accelerating in the y, following a parabolic curved path to the destination. Each of the individual operations is added using the `addkinematic` command. Then, when refreshing the view during the kinematic motion, you call `updatekinematics`, which calculates the current position and rotation of the object. All of this will be explained later. First, the `initkinematics` command.

For the `initkinematics` command, as well as all other kinematics commands, you must pass a reference to a blank node as the first parameter. This specifies where you want kinematic information to be stored, or, if you are getting information out of it, where kinematic information has been stored. The node needs to be an otherwise unused node, so that kinematic functionality can store data as needed. For modelers, this node will most likely be a label. Once kinematics have been initialized, the node will display the text "do not touch". This node should not be clicked on in a tree or table view while a kinematic operation is in process, or the kinematic data may be corrupted.

The `initkinematics` command is overloaded, so you can call `initkinematics` with two different parameter sets, depending on your situation. Parameters for these two overloads are as follows.

**void initkinematics(treenode datanode, treenode object [, int managerotation, int localcoords] )**

This parameter set assumes you have an object that you want to move, like a transporter. The first parameter, again, is a blank node kinematic data, either a label, attribute, or variable. The second parameter is the object that will do the moving. The command will save off the object's initial location and rotation. The optional parameter managerotation is either 1 or 0. If 1, the rotation of the object will be set according to the velocity of the object at any given time. By default, the object's positive x direction will always point in the direction of the object's current velocity. This would be used, for example, if you have a truck that you always want to point forward as it travels. If managerotation is passed as 0, then the object won't rotate unless given commands to rotate, which will be explained later. If you do not specify this parameter, then by default, managerotate is set to 0. The optional localcoords parameter specifies the orientation of subsequent travel command locations. For example, if my truck is rotated 45 degrees, and I want it to travel 5 units in the x direction, this can be interpreted in two different ways. Do I want it to travel 5 units in x according to the truck's coordinate system, or 5 units in x according to the model's coordinate system (or the coordinate system of the truck's container)? In the former case, the object would actually travel 3.5 units in the x direction and 3.5 units in the y direction according to the model's coordinate space. In the latter case, however, the object would travel the usual 5 in the x direction according to the model's coordinate space. The localcoords parameter specifies which coordinate system you want to use. If 1 is passed, the object's coordinate system will be used (the former case). Note that only the object's initial coordinate system will be used to calculate locations, not subsequent coordinate systems if the object rotates later on in the kinematics. If 0 is passed, the object's container's coordinate system will be used (the latter case). If you do not specify this parameter, the default is 0.

**void initkinematics(treenode datanode [, double x, double y, double z, double rx, double ry, double rz, int managerotation, int localcoords] )**

This parameter set allows you to explicitly pass initial locations and rotations, instead of referencing an object. Although you would probably more often use the other parameter set where you pass the object in, this parameter set gives you ultimate flexibility. Use this if you want to explicitly pass in the initial locations and rotations of the object, or if your location and rotation values don't necessarily represent real locations and rotations in your model. For example, you are simulating a robot arm, and there are several joints of the arm that move/rotation with different acceleration/deceleration/max speed values. The visualization of movements of the arm are not simulated with explicit Flexsim locations/rotations, but are done using your own draw commands and labels or variables. In this case, you don't want kinematics to be applied to straight rotations and locations, but rather to information that you are keeping on the object yourself. In such situations, a given set of kinematics does not need to be viewed as applied directly to x,y,z locations and x,y,z, rotations, but can rather be viewed as six separate kinematic motions, each along one axis. These six axes can represent whatever you want them to. For example, your robot has four joints, each with one rotation value. To have the four joints of the robot move using kinematics, you can have each joint simulate one axis in the kinematics. The x part of the kinematics applies to the rotation of joint 1, the y part applies to the rotation of joint 2, the z part applies to the rotation of joint 3, and the rx part applies to the rotation of joint 4. The other two parts of the kinematics, ry and rz, you don't worry about. You can initialize the kinematics with the start rotations of

each of your 4 joints, and then add kinematics that apply to each joint individually. Then, when you want to get the joints' current rotation values out later to draw the robot arm in motion, you can use the `getkinematics` command instead of the `updatekinematics` command to get the values explicitly, and not have them be applied to an object's location or rotation. These commands will be explained later.

Note that the `x`, `y`, `z`, `rx`, `ry`, and `rz` parameters are optional. If not passed into the command, default for each is 0. The `managerotation` and `localcoords` variables are the same as in the first `initkinematics` command.

**`void setkinematicsroffset(treenode datanode, double rx, double ry, double rz)`**

This command would only be used if `managerotation` is passed into `initkinematics` as 1. This allows you to set an initial rotation from which the rotation is managed. By default, the object's positive `x` direction will always point in the direction of the object's current velocity. In the case of a truck, you may want the truck, instead of always being rotated so that it is traveling forward, to travel backward. Here you could specify a rotation offset of (0,0,180).

**`double addkinematic(treenode datanode, double x, double y, double z, double targetspeed [, double acc, double dec, double startspeed, double endspeed, double starttime, int type ] )`**

This command adds a kinematic to the set of kinematics. The `x`, `y`, and `z` parameters make up an offset location or rotation. For example, the location (5,5,0) tells the kinematic to travel 5 in the `x` and 5 in the `y`. Note that these are offsets from the object's current position, and not absolute locations. The `targetspeed` parameter specifies the target speed for the travel operation. The other parameters are optional. `Acc` specifies the acceleration. `Dec` specifies the deceleration. `Startspeed` specifies the speed that the kinematic should start at. If this speed is higher than the target speed, then the object will start at the start speed and decelerate down to the target speed. `Endspeed` specifies the ending speed for the operation. If `endspeed` is greater than `targetspeed`, then at the end of the operation, the object will accelerate from the target speed to the end speed. The `starttime` is the start time of the kinematic, in simulation time, not as an offset from the current time. The `type` parameter specifies what type of kinematic it is. This value should either be `KINEMATIC_TRAVEL`, or `KINEMATIC_ROTATE`. If it is `KINEMATIC_TRAVEL`, the operation will be applied to the `x`, `y`, and `z` location values. If it is `KINEMATIC_ROTATE`, the operation will be applied to the `rx`, `ry`, and `rz` rotation values, and speeds are defined in degrees per unit of time, accelerations/decelerations in degrees per unit of time squared. The command returns the time that this kinematic operation will finish. If optional parameters are not passed into the command, the following defaults apply:

`acc`: 0 (or infinite acceleration)  
`dec`: 0 (or infinite deceleration)  
`startspeed`: 0  
`endspeed`: 0  
`starttime`: current time  
`type`: `KINEMATIC_TRAVEL`

**`void updatekinematics(treenode datanode, treenode object [, double updatetime])`**

This command should be called in the middle of the kinematic operation, usually on predraw or draw. It calculates and then sets the current location and rotation of the object, according to all kinematics that have been added and the current update time. The updatetime parameter is optional. If it is not passed, the current simulation time is used.

**double getkinematics(treenode datanode, int type [, int kinematicindex, double updatetime/traveldist])**

This command is used if you want to get explicit information on the kinematics. You can get information on the entire set of kinematics, or on each individual kinematic. Use this if your kinematics do not apply directly to object locations and rotations, or if you need this information in your logic. The type parameter specifies the type of information you want, and will be explained shortly. The kinematicindex parameter is optional, and specifies which individual kinematic you want to get information for. For example, if you add a kinematic to travel 5 units in the x direction as the second addkinematic command, you would pass a 2 as the kinematicindex parameter into the getkinematics command. If not passed, or if you pass a 0 value here, the default gets information for all kinematics together. The updatetime/traveldist parameter is optional. The meaning of this parameter depends on the type parameter you specify. Sometimes this parameter will not be used. Some of the time it represents the requested update time that you want to get information for. If not passed, the current time is used. In the case of a KINEMATIC\_ARRIVALTIME query, this parameter instead represents travel distance. The use of this parameter will be explained with each query type.

If you are getting information for all kinematics (kinematicindex is not specified or is 0), then you can pass the following values as the type parameter.

KINEMATIC\_X, KINEMATIC\_Y, KINEMATIC\_Z: return the x, y, and z locations that the object will be at for the given time.

KINEMATIC\_RX, KINEMATIC\_RY, KINEMATIC\_RZ: return the x, y, and z rotations that the object will be at for the given time. This will only work if rotation is not being managed by the kinematics, but rather you are managing it yourself.

KINEMATIC\_VX, KINEMATIC\_VY, KINEMATIC\_VZ: return the x, y, and z velocities that the object will be at for the given time.

KINEMATIC\_VRX, KINEMATIC\_VRY, KINEMATIC\_VRZ: return the x, y, and z rotational velocities that the object will be at for the given time.

KINEMATIC\_NR: Returns the number of kinematics that have been added. Here the updatetime parameter is not used.

KINEMATIC\_STARTTIME: This returns the lowest start time of all the kinematics. Here the updatetime parameter is not used.

KINEMATIC\_ENDTIME: This returns the highest end time of all the kinematics. Here the updatetime parameter is not used.



**KINEMATIC\_VELOCITY:** This returns a scalar value of the total velocity for the given time.

**KINEMATIC\_RVELOCITY:** This returns a scalar value of the total rotation velocity for the given time. This will only work if you are managing rotations yourself.

**KINEMATIC\_ENDDIST:** This returns the distance from the final destination location of all kinematics from the object's initial location. Here the updatetime parameter is not used.

**KINEMATIC\_ENDRDIST:** This returns the distance from the final destination rotational position of all kinematics from the object's initial rotational position. This will only work if you are managing rotations yourself. Here the updatetime parameter is not used.

**KINEMATIC\_TOTALDIST:** This returns the sum of the distances of all the added kinematics. This has a subtle difference from **KINEMATIC\_ENDDIST**. For example, if your first kinematic travels 10 in the x direction, and your second kinematic travels -10 in the x direction, then the enddist value will be 0, whereas the totaldist value will be 20. Here the updatetime parameter is not used.

**KINEMATIC\_TOTALRDIST:** This returns the sum of the rotational distances of all the added kinematics. This will only work if you are managing rotations yourself. Here the updatetime parameter is not used.

**KINEMATIC\_CUMULATIVEDIST:** This returns the cumulative travel distance of all added kinematics. This is different than enddist or totaldist; it calculates the distance of the possibly curved path that the object will follow during the entire kinematic operation. Here the updatetime parameter is not used.

**KINEMATIC\_CUMULATIVERDIST:** This returns the cumulative rotational travel distance of all added kinematics. This will only work if you are managing rotations yourself. Here the updatetime parameter is not used.

**KINEMATIC\_TOTALX, KINEMATIC\_TOTALY, KINEMATIC\_TOTALZ:** These return the sum of x, y, or z component of all added kinematics. Here the updatetime parameter is not used.

**KINEMATIC\_TOTALRX, KINEMATIC\_TOTALRY, KINEMATIC\_TOTALRZ:** These return the sum of rx, ry, or rz component of all added kinematics. This will only work if you are managing rotations yourself. Here the updatetime parameter is not used.

If you are getting information on an individual kinematic (by passing a kinematicindex parameter greater than 0), you can pass one of the following values as the type parameter.

**KINEMATIC\_X, KINEMATIC\_Y, KINEMATIC\_Z:** These return x, y, or z component of the current location of the specified kinematic at the given update time. For example, if you added a kinematic to travel 10 units in the x, starting at time 5, and you want to know the x location for this given kinematic at time 7, you can call `getkinematics(datanode, KINEMATIC_X, index, 7)` to get the x location for time 7.

**KINEMATIC\_RX, KINEMATIC\_RY, KINEMATIC\_RZ:** These return the rx, ry, or rz component of the current location of a rotational kinematic at the given update time.

**KINEMATIC\_VX, KINEMATIC\_VY, KINEMATIC\_VZ:** These return the x, y, or z component of the current velocity for the specified kinematic at the given update time.

**KINEMATIC\_VX, KINEMATIC\_VY, KINEMATIC\_VZ:** These return the x, y, or z component of the current rotational velocity for the specified kinematic at the given time if it is a rotational kinematic.

**KINEMATIC\_ENDTIME:** This returns the time that the specified kinematic will finish its operation. This is the same endtime that is returned from the `addkinematic` command. Here the `updatetime` parameter is not used.

**KINEMATIC\_STARTTIME:** This returns the time that the specified kinematic will start its operation. This is the same starttime that you specified when you added the kinematic. Here the `updatetime` parameter is not used.

**KINEMATIC\_ARRIVALTIME:** In this query, the `updatetime/traveldist` parameter is used as a requested travel distance for the given kinematic. This returns the time of arrival for a certain sub-distance of a given kinematic. For example, if I've added a kinematic that tells the object to travel 5 units in the x direction, but I want to know how long it will take him to travel just 3 of those 5 units, I can use this query, and pass 3 in as the `traveldist` parameter.

**KINEMATIC\_STARTSPEED:** This returns the start speed for the kinematic. This is the `startspeed` you specify in the `addkinematic` command. Here the `updatetime` parameter is not used.

**KINEMATIC\_ENDSPEED:** This returns the end speed for the kinematic. This is usually the `endspeed` that you specify in the `addkinematic` command, but may not be if the kinematic cannot decelerate/accelerate to your specified `endspeed` given the distance it has to travel. Here the `updatetime` parameter is not used.

**KINEMATIC\_ACC1:** This returns the acceleration value used to get from the start speed to the target speed. If the start speed is less than the target speed, then this value will be the acceleration value. Otherwise it will be the negative deceleration value. Here the `updatetime` parameter is not used.

**KINEMATIC\_ACC2:** This returns the acceleration value used to get from the target speed to the end speed. If the end speed is less than the target speed, then this will return the negative deceleration value. Otherwise it will return the acceleration value. Here the `updatetime` parameter is not used.

**KINEMATIC\_PEAKE SPEED:** This returns the peak speed or “reached speed” for the kinematic. This is usually the same as the target speed specified in the `addkinematic` command, but may not be if the kinematic cannot get to the target speed given the distance it has to travel. Here the `updatetime` parameter is not used.

**KINEMATIC\_ACC1TIME:** This returns the total time the kinematic will spend accelerating/decelarting from the start speed to the target speed. Here the `updatetime` parameter is not used.

**KINEMATIC\_PEAKTIME:** This returns the total time the kinematic will spend traveling at the peak speed. Here the `updatetime` parameter is not used.

**KINEMATIC\_ACC2TIME:** This returns the total time the kinematic will spend accelerating/decelerating from the target speed to the end speed. Here the `updatetime` parameter is not used.

**KINEMATIC\_TOTALDIST:** This returns the total distance for the kinematic operation.

**KINEMATIC\_TOTALRDIST:** This returns the total rotational distance for the kinematic operation if it is a rotational kinematic.

**KINEMATIC\_TOTALX, KINEMATIC\_TOTALLY, KINEMATIC\_TOTALZ:** These return the x, y, or z component of the total distance for the kinematic operation. These are the same values you passed into the `addkinematic` command. Here the `updatetime` parameter is not used.

**KINEMATIC\_TOTALRX, KINEMATIC\_TOTALRY, KINEMATIC\_TOTALRZ:** These return the x, y, or z component of the total rotational distance for the kinematic operation if it is a rotational kinematic. These are the same values you passed into the `addkinematic` command. Here the `updatetime` parameter is not used.

**KINEMATIC\_VELOCITY:** This returns the total velocity of the kinematic for the given time.

**KINEMATIC\_RVELOCITY:** This returns the total rotational velocity of the kinematic for the given time if it is a rotational kinematic.

**KINEMATIC\_TYPE:** This returns `KINEMATIC_TRAVEL` if the specified kinematic is a travel operation, and `KINEMATIC_ROTATE` if the kinematic is a rotate operation.

### **`void profilekinematics(treenode datanode [, int index ])`**

This command prints kinematic information to the output console. The index parameter is optional. If it is not passed, then information will be printed for all kinematics that have been added. If it is passed, then the index variable is an index of the added kinematic you want to print information for.

### **`void deactivatekinematics(treenode datanode)`**

This command tells the kinematic to not update locations when the `updatekinematics` command is called. Execute this on reset to free the object to move it around in the ortho view.

## Creating Custom Libraries

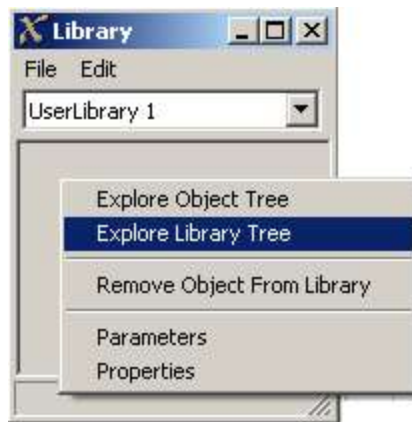
The custom library, or user library, mechanism provides you with flexibility in many areas. The most widely used functionality of the user libraries is to reuse customized objects in a model, but you are not confined to only that functionality. User libraries allow you to automatically install custom objects and data into models, install objects and data into the main project and view tree, and execute code when an object in the icon grid is dropped into the model. All of this functionality stems from two mechanisms within the user library functionality. First is the droppath and dropscrip mechanism, which allow you to customize what happens when an object is dropped into that model. Second is the automatic install mechanism, which allows you to install objects or data when the user does operations like creating a new model or opening a model. This topic will discuss these two mechanisms in detail.

### Dropscrip and Droppath

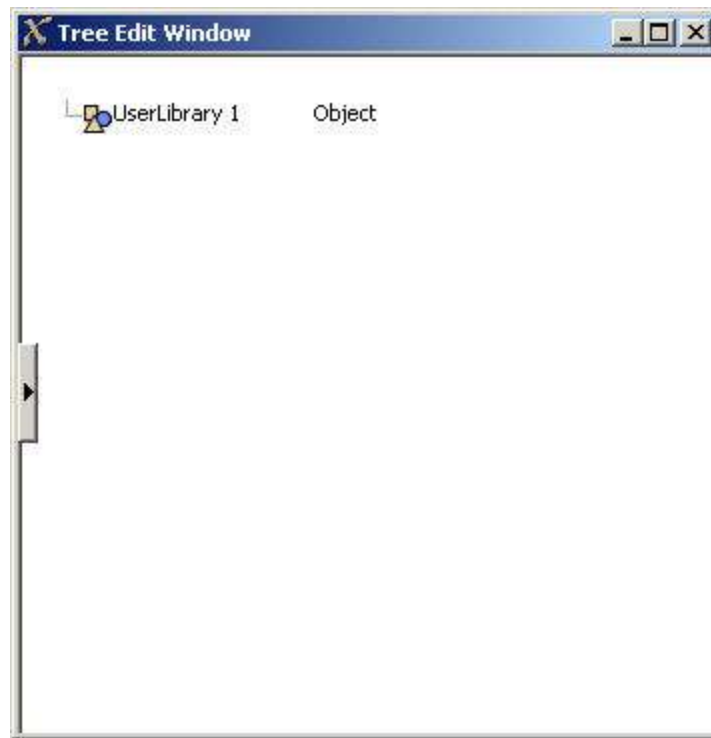
The droppath and dropscrip mechanism allows you to customize what happens when an object is dropped into the model. This is done by creating a custom object in the user library, and then adding either a "dropscrip" or a "droppath" attribute to the object and specifying the data for the dropscrip or droppath.


### Dropscrip

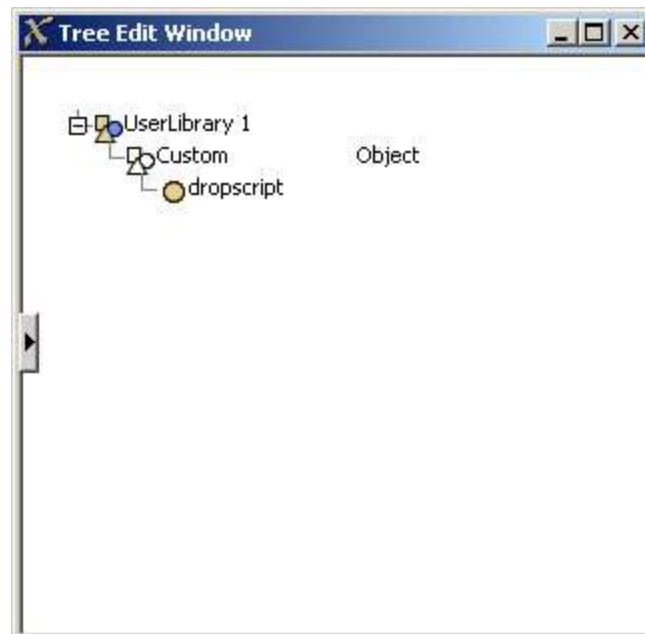
If a dropscrip attribute is added, then the dropscrip node will be executed as flexscript, and values will be passed into the flexscript function that give it information about the drop. Let's do an example. Create a new user library by going to File|New Library in the Library Icon Grid's window menu. Right-click into the icon grid explore the library as a tree.



You should see an empty tree.



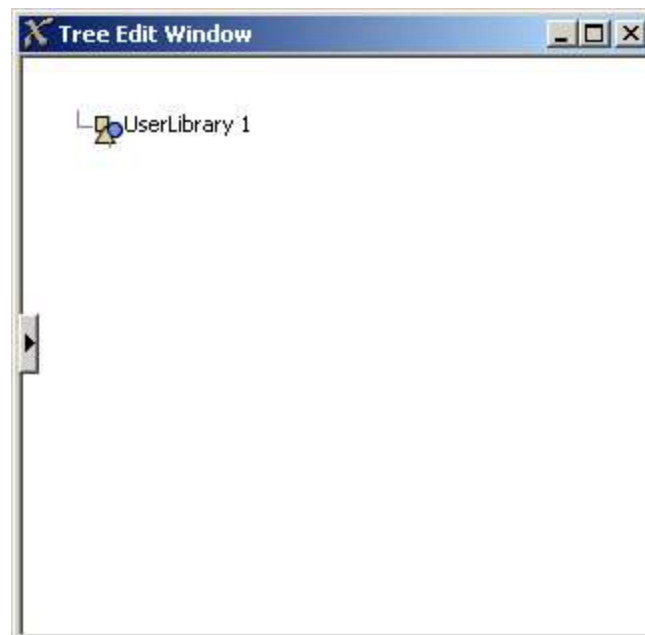
Insert a new object into the library by right clicking on it and selecting Edit|Insert Into, or by left-clicking on it and hitting the Enter key. Expand the user library tree so you can see the node you created. Give it the name "Custom". Now add object data to the node by right-clicking on it and selecting Insert|Add Object Data. Click on the  button to expand the object. Then insert an attribute node by clicking on the object node and hitting the Enter key. Give the attribute node the name "dropscript". The tree should appear as follows.



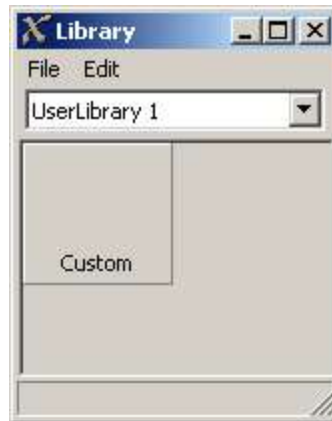
Add text to the dropscrip attribute by right-clicking on it and selecting Insert|Add String Data. Toggle the node as flexscript by right-clicking on it and selecting Build|Toggle as Flexscript. Give the attribute the following flexscript text:

```
msg("Object Dropped",
concat(
  "Onto Object: ", getname(parnode(1)),strascii(13),
  "X: ", numtostring(parval(2),2,2),strascii(13),
  "Y: ", numtostring(parval(3),2,2),strascii(13),
  "Z: ", numtostring(parval(4),2,2),strascii(13),
  "Onto View: ", getname(parnode(5))
)
);
```

Click off of the attribute node, then right-click it and select Build|Build Node Flexscript. Finally, add a picture attribute to the attribute tree of the object by right-clicking on the drop-script node and selecting Edit|Insert. Name the new node "picture". The libraries icon grid will only show the object in the grid if the object has a picture attribute. Usually this is a path to a bitmap file that represents the picture that is shown in the icon grid for that object. In our case we won't worry about that, but will just leave the picture attribute blank, which causes a blank icon to be shown in the grid with the object's name. The tree should appear as follows.



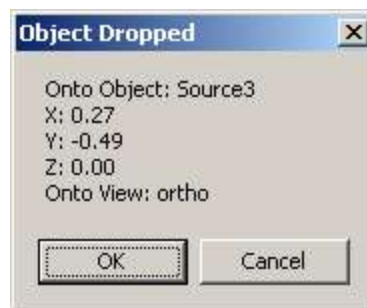
The library icon grid should appear as follows.



Now drag the Custom icon from the Library Icon Grid into the model. A message should appear.



This message is the code of the dropscript being executed. It shows the object that you dropped onto if there is one, an x,y and z location of the drop, and the view window that the object was dropped into. You can also drop it onto another object. Drag a regular library object into your model, then go back to the user library and drag the Custom object onto the object in the model. The following message should appear.



This time the message shows that it was dropped onto Source3. The x, y and z locations are now relative to the source object.

As the example shows, there are 5 access variables you can get within the dropscript function. They are:



parnode(1) : the object that was dropped onto. If the user dragged onto a blank area in the model, then this variable will be 0, or NULL. In such a case, you can usually assume that the user intends to drop it into the model. However, this is not always the case. For example, the user may drop the object into an ortho view open that is viewing a subspace of the model, like a VisualTool. To ensure a safe drop in this case, you should use the viewfocus of the view as the drop point (node(">viewfocus+", parnode(5))).

parval(2), parval(3), parval(4) : the x, y and z location of the drop, respectively. If the parnode(1) is NULL, then this is the location within the model space (or the view's viewfocus' space). If parnode(1) is not NULL, then this is the location within parnode(1).

parnode(5) : the view window onto which the object was dropped. Use node(">viewfocus+", parnode(5)) to get access to the viewfocus of the view.

If within the dropscrip function you would like to do a standard object drop, as if it were a regular user library object being dropped, then you can use dropuserlibraryobject(). This command executes the same functionality as when a regular object is dropped from a user library object. The first parameter is the object to drop, and the next five parameters are the onto object, x, y, z, and onto view, just like the dropscrip function itself. The command returns a reference to the object that was created. Your dropscrip function should also return a reference to the object you created. This allows Flexsim to do some extra work like checking to see if there were any c++ nodes created, and if so, set a flag to notify you that you need to compile the next time you run the model.

The dropscrip mechanism allows you ultimate flexibility with user libraries. You can use it in the standard way, for example to do a regular object drop but also execute some extra code to initialize the dropped object. You can also use it in more non-standard ways. Some ideas may be to use a drop scrip to add a curved or straight section to a conveyor belt, or to add levels or bays to a rack object. You could also use a dropscrip to add code to triggers of an object, or to set certain parameters of the object. You could add collision spheres to an object, or add a set of functionality to the model's tools folder. There are any number of possibilities.

## Droppath


The droppath mechanism is a method of indirection. It is a way that you can refer to another object that you want to be dropped when the icon is dropped into the model. This is usually used with automatic install functionality, but it can also be used with an icon in the icon grid. The method creating a droppath object is the same as for creating a dropscrip. You add an object to the library, and give it object data, but this time you give it an attribute named "droppath". You should not toggle the node as flexscrip, but should give text data to the droppath attribute, and specify in the text a path to another object that should be dropped. For example, you may give the droppath attribute the text: MAIN:/project/library/NetworkNode. This will effectively cause a network node to be created when you drag your object's icon into the model. You might ask why not just add a network node to the user library, or why not just make them go to the regular library and drop a network node from there. To the former question, if you plan on using this user library for a long time, you may want

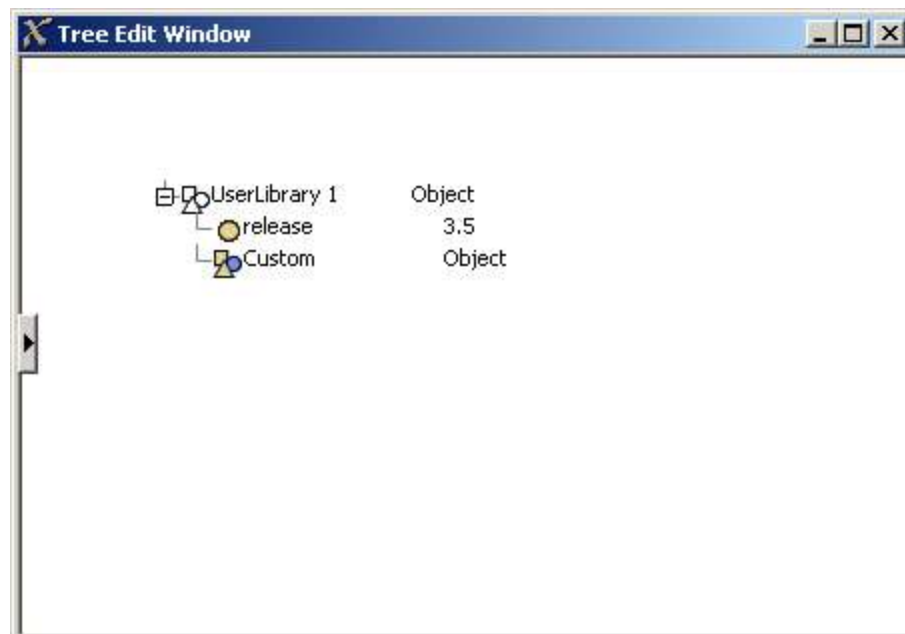
the network node to be compatible with future versions of Flexsim. You may want to drop whatever is the latest version of the network node, without having to update the user library with each new version. Also, why have redundant data in your user library if all of it is already there in the regular library? To the latter question, it may be useful to have the network node as a droppable icon in your library if the user uses the object often. It reduces the number of mouse clicks needed. Also, in the case of the network node object there are some issues with the OnClick event of the network node, which fires when you click on the object in the icon grid. This alternative works around that error.

The droppath text path is relative to the droppath node itself, so you can reference other objects within the library using normal path constructs like .., / and >.

## Automatic Install

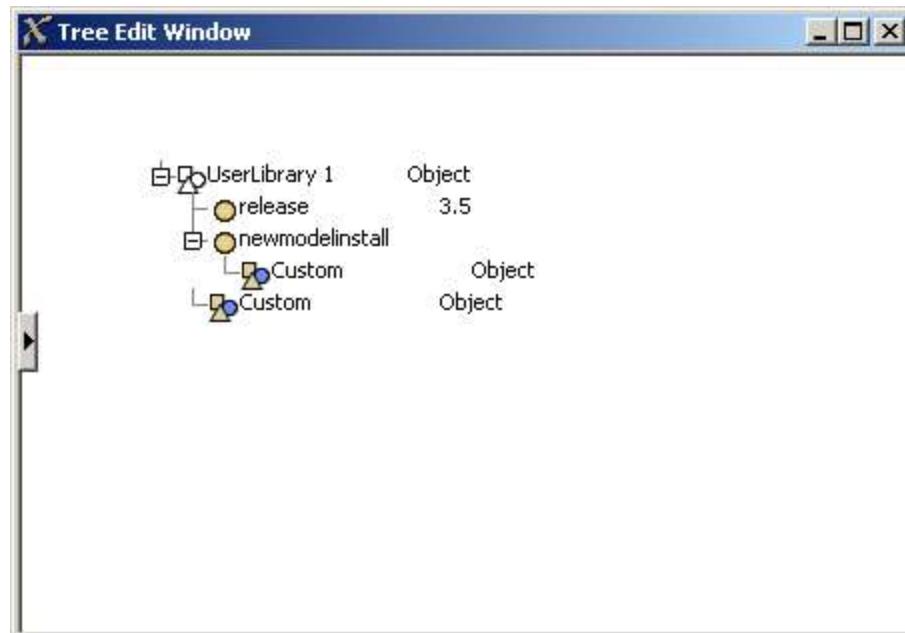
The automatic install mechanism allows the user library to install objects and functionality to the model when the user performs certain actions like creating a new model, or opening a model. The way you do this is by creating special folders within the library's attributes tree. Again, let's do another example. We will start from our previous example, and instead of having the object dropped when the user drags the icon, we will have it dropped when the user creates a new model.

Collapse the Custom object's attributes and expand the library's attributes tree by pressing the  button next to the library. The tree should appear as follows.



Insert a node after the release node by right-clicking on it and selecting Edit|Insert. Name the new node "newmodelinstall". Insert a node into the newmodelinstall node by right-clicking and selecting Edit|Insert Into. Expand the newmodelinstall node so you can see the new node. Then copy the Custom object and paste it into the node inside of the newmodelinstall node. Right-click on the Custom object and select

Edit|Copy. Then right-click on the node inside the newmodelinstall node and select Edit|Paste. The library should now appear as follows.



Now you can hit the "New" button on the toolbar of File|New Model, and the message will appear. The newmodelinstall folder acts a set of objects that will be "dropped" into the model when a new model is created. What I mean by "dropped" is that the same functionality is executed as when you actually drag an object from the icon grid. In the case of a dropscrip object, the script is execute as if the object were dropped at the point (0,0,0) in the model. If it were a regular object like a Source or Queue, then the object is created at (0,0,0) in the model. Please note that there are a few differences, though. With a dropscrip there is no view object to reference (parnode(5)). Also for a drop script, if the drop script function returns 1, then no views will be created by Flexsim. This allows your drop script to open a custom set of view windows when the new model is created.

There is a specific set of folder names that are valid in using the automatic install mechanism. They are listed as follows.

**newmodelinstall** : This folder is installed when the user creates a new model. It is also installed when the library is loaded.

**startupinstall** : This folder is installed when the library is loaded if the user has designated this library as a library to load during startup.

**loadinstall** : This folder is installed when the library is loaded explicitly by the user.

**openmodelinstall** : This folder is installed when the user opens an existing model. You may use this folder to check is the model contains the library's components, and if not, install them. You could also use it to update components in the model if those components are from an earlier version of the user library.

## Icon Grid Width and Height

You can also designate the size of icons in the user library's icon grid. Just give the library attributes named "cellwidth" and "cellheight". Each attribute should contain a number that is the width and height of one icon in the icon grid in pixels.

## View Attributes Reference

This document describes in detail the different GUI attributes you can use when creating custom GUIs. This describes attributes used for each view type and how those attributes affect the behavior of the control. Please read the topic on Graphical User Interfaces before referencing this topic.

### View Window Classes

In this topic, we will occasionally refer to Flexsim window classes versus Windows common controls. The difference between these two can be subtle but are nonetheless important. Flexsim uses the standard Microsoft Windows interface for its windowing system. Within that interface there is an ability to register custom window classes and define special functionality for those windows. There is also the ability to use Windows common controls, which are standard window classes that have already been defined by Windows like buttons, comboboxes, etc. Flexsim uses many of the common controls, but it also has created some of its own window classes. These two categories, and which Flexsim view types fit in which category, are listed below. I mention this fact because there are several view attributes in the "general" category that will only work on Flexsim registered controls. This will be mentioned under each attribute's description.

### Windows Common Controls

- static (or label)
- button
- radiobutton
- checkbox
- edit
- tracker
- combobox
- listbox
- tabcontrol
- statusbar

### Flexsim Registered Controls

- groupbox
- panel
- table
- graph
- ortho
- perspective
- tree
- icongrid
- planar
- dialog (this is the main window)

### Other Controls Used by Flexsim

- script (this uses the Scintilla text editor control)

There are also some controls available in the GUI builder's icon grid that are made up of a combination of one or more of the controls mentioned above. These controls will not be documented in this topic since they are simply a combination of functionality for controls that are documented here.

In this topic the terms view and control are used interchangeably. They basically describe a window control in Flexsim.

Many of the attributes described below can contain flexscript code that is executed. Unless otherwise stated, when adding such a flexscript attribute, you can choose whether you want to toggle the node as flexscript. If you toggle the node as flexscript, then Flexsim does not need to build the node's flexscript code when the function is executed. This results in a faster execution time, but also requires more memory to store data in the tree, and it causes Flexsim's global "Build all flexscript" function to take more time. As a rule of thumb, if the function is executed in "user" time, meaning it is an operation that is explicitly done by the user, like pressing a button or selecting a combobox option, Flexsim can build the flexscript fast enough so the user notices no difference, so you don't need to toggle the node flexscript. However, if it is an operation that is executed quite often, like a hotlinkx, or an OnDraw, then it might be wise to toggle the node as flexscript to increase refresh rates.

Please note that this does not document every single attribute possibility, although it should provide you with all the information you need for most situations.

## General Attributes

Below is a list of general attributes that can be added to almost every view type.

**alignrightmargin, alignbottommargin:** These attributes signal that the control's margin is "locked" to the right or bottom margin of the control's container window. They should contain number data, and the value represents the offset distance from the container window's right or bottom margin, in pixels, that the control's margin will be "locked" to. For example, if you give a button control an alignrightmargin attribute with a value of 10, then as you resize the window, the button will automatically resize so that its right margin is 10 pixels from the right edge of the window.

**alignrightposition, alignbottomposition:** These attributes signal that the control's position is "locked" to the right or bottom margin of the control's container window. They should contain number data, and the value represents the offset distance from the container window's right or bottom margin, in pixels, that the control's position will be "locked" to. For example, if you give a button control an alignrightposition attribute with a value of 100, then as you resize the window, the button's x position will automatically change so that its button's position (or left side) is 100 pixels from the right edge of the window.

**aligncenterx, aligncentery:** These attributes signal that the control should be center aligned with its container control. The attributes should contain number data, and their value represents an offset from the center position.

**grayed:** This attribute causes the control to be disabled, graying the control and disallowing the user from manipulating the control. It should contain number data; 1 means grayed, 0 means not grayed. Once a control has been initialized, the "grayed" state of the control will not change by simply changing the value of the grayed attribute. Because of this, and because the "grayed" state of a control is very dependent on certain parameters of objects, and can change during the life of the window, it is usually more practical to use the `windowgray()` command to change the "grayed" state of a control. This command does not require you to even have a grayed attribute, so the grayed attribute, for the most part, is unneeded.

**hidden:** This attribute designates that the control will be hidden from the user. For the same reasons as with the grayed attribute, this attribute is usually unneeded and can be replaced with the `windowshow()` command.

**coldlink:** This attribute is used to link a control with a value in the model. For example, an edit control with a coldlink to the max content of a queue will show the max content as a text in the edit control. It is a "cold" link because it is only refreshed when the window is opened, and only applied when the Apply or OK button is pressed. The coldlink attribute should contain text data that is a path to the node that holds the linked data. The path starts at the coldlink node itself. Refer to the Graphical User Interfaces topic for more information on the syntax of this path. You can use the `applylinks()` command to apply or refresh the coldlink. The first parameter is a node that is the start location for a recursive search. The second parameter is optional. The command will recursively search the window's tree structure and find any coldlinks (and hotlinks) and apply the coldlinks to the object's attributes. If the optional second parameter is 1, then instead of applying the coldlinks to the object, the `applylinks()` command will refresh the window's values as defined by the object's values.

**hotlink:** This attribute is used to link a control with a value in the model, just like the coldlink mentioned above. It is a "hot" link because the value shown in the control is continuously refreshed each time the window is repainted. Otherwise the hotlink is exactly the same as the coldlink.

**coldlinkname:** This attribute is just like the coldlink, except it links with the name of the node specified by the coldlink's path.

**hotlinkname:** This attribute is just like the hotlink, except it links with the name of the node specified by the coldlink's path.

**coldlinkx:** This attribute is like a coldlink, except that instead of holding text data with a path to the involved node, the coldlinkx holds flexscript code. The flexscript function should return a reference to the node that the view should link to. If 0 is returned, then nothing will be applied or refreshed for the control. Within the function there are 3 access variables. `c` is a reference to the control itself. `i` is a reference to the object focus of the view (the same as `node("@>objectfocus+",c)`). `eventdata` is either 1 or 0. If 0, then the coldlinks function is being executed in order to refresh the control according to the object's variable. If 1, then the coldlinkx is being executed in order to apply the value to the object's variable. Please note that when the coldlinkx function is called, the return value, or in other words the reference to the linked node,

is not remembered by the window. Each time the control needs to be refreshed or applied, the `coldlinkx` function is called again. This means that the `coldlinkx` function can actually be called many times throughout the life of the window.

**hotlinkx:** This attribute is just like the `coldlinkx`, except that it is refreshed every time the window is repainted.

**menupopup:** This attribute is only applicable for Flexsim registered controls. By adding this attribute you can define the menu that will appear when the user right-clicks on the control. The attribute should contain sub-nodes. Each sub-node is a menu option of the pop-up. The sub-node's name defines what the menu item's text will be. The sub-node should have text data. The text data defines a flexscript function that will be executed when the option is selected. Within the flexscript function there is one access variable, `c`. `c` is a reference to the menu option sub-node. Some commands that might be used within the flexscript function are listed below. For detailed information on each command, refer to the command documentation.

**ownerobject(c):** this returns a reference to the ownerobject of the menu option, or the view node itself.

**selectedobject():** this will return a reference to the highlighted (yellow) object of the view. For example, if the view is an ortho view, and the user right-clicks on an object in the ortho view and selects a pop-up menu option, the flexscript function can access the highlighted object with `selectedobject(ownerobject(c))`.

Standard views that use this attribute include the ortho and perspective windows, the library icon grid, and the table view in the labels tab of an object's properties window.

Right now there is no known way of dynamically customizing the popup menu based on what the user clicks on.

If there is no `menupopup` attribute specified for a Flexsim registered control, then the standard Flexsim pop-up menu will appear.

**tooltip:** This attribute defines a tip that will pop-up when the mouse hovers over the control for a certain amount of time. The attribute should have text data containing the text that should be shown.

**style:** This attribute is a special attribute that connects directly with windows control styles. When each control is created, it is given a default style, which is a 32-bit field that is passed to Windows where each bit represents a certain flag that affects the control's appearance or behavior. The style attribute can override the default style that Flexsim gives the control. Window styles are documented online on Microsoft Developer Network (MSDN). Go to [www.msdn.com](http://www.msdn.com) and search for Window styles. This will provide list of default window styles. There are also styles specific to each Windows common control. For example, you can search for Button Styles and it will show a list of all the styles you can give a button control.

The style attribute can have any number of sub-nodes. Each sub-node's name defines a windows style for that control, such as `WS_DISABLED` or `WS_BORDER`. The sub-node may contain optional number data. If the number data is 0, then that



will signal for Flexsim to set the bit flag low, or 0, in case the Flexsim default is for the bit flag to be high. If no number data is specified, then Flexsim will set that flag as a high bit.

An example of using the style attribute is to have a checkbox with the style BS\_PUSHBUTTON. This will cause the checkbox to instead look like a button that is depressed when checked. The ortho window's tool bar uses this style for radio buttons in its mode panel.

**exstyle:** This attribute is much like the style attribute, except it defines windows extended styles, which are styles not introduced until Windows 3.1 I believe. These styles begin with WS\_EX\_ instead of WS\_ and are also documented on MSDN.

**OnClick:** This attribute only applies to Flexsim registered controls. It is fired when the user clicks anywhere inside of the control. This includes when the user clicks the mouse and when the user releases the mouse. There are two access variables. c is a reference to the control node. i is a clickcode: 2 means left mouse button down, 3 mean left up, 4 mean right down, 5 means right up, and 1 means double-click. Some commands that you might use within the OnClick are: cursorinfo(), selectedobject().

**OnMouseButtonDown:** This attribute only applies to Flexsim registered controls. It is fired when the user presses the left mouse button in the view. There is one access variable, namely c, which is a reference to the control node.

**OnMouseButtonUp:** This attribute only applies to Flexsim registered controls. It is fired when the user releases the left mouse button in the view. There is one access variable, namely c, which is a reference to the control node.

**OnKeyDown:** This attribute only applies to Flexsim registered controls. It is fired when the control has keyboard focus and the user presses a key on the keyboard. There is one access variable, namely c, which is a reference to the control node. You can also query what key went down using lastkeydown(), or query whether any key is down with iskeydown().

**OnKeyUp:** This attribute only applies to Flexsim registered controls. It is fired when the control has keyboard focus and the user presses a key on the keyboard. There is one access variable, namely c, which is a reference to the control node. You can also query what key went down using lastkeydown(), or query whether any key is down with iskeydown().

**OnMouseWheel:** This attribute only applies to Flexsim registered controls. It is fired when the control has keyboard focus and the user scrolls the mouse wheel. The control also needs to have an OnMouseWheelDelta attribute with number data. When the user scrolls the mouse wheel, Flexsim will set the value of the OnMouseWheelDelta value according to how much the user has scrolled, and then will call the OnMouseWheel function. Within the function, c access the OnMouseWheel attribute itself.

**OnMouseWheelDelta:** This attribute is used as described in the OnMouseWheel attribute above.

**OnDrop:** This attribute only applies to Flexsim registered controls. It is fired when the user drags an object from an icon grid and drops it on the view. The attribute should have text data containing flexscript code that fires when the object is dropped. Within the function, you have access to the object that was dragged with `dropnodefrom()`, and the object that it was dropped onto with `dropnodeto()`.

## Dialog

The following attributes can be added to a dialog view type, or the view node for the main window.

**menucustom:** This attribute acts much like the `menupopup` attribute, except the menu will appear at the top of the window instead of appearing when the user right-clicks in the view.

**menuview:** This attribute allows you to have the standard Flexsim menu appear at the top of the window. The attribute should have number data and be set to 1 to have the menu appear.

**objectfocus:** This attribute will specify a path to the object that a given window instance will point to. The `createview()` command contains a parameter that specifies the objectfocus, the objectfocus attribute will be filled with that parameter when the window is opened.

**OnOpen:** This attribute allows you to specify flexscript functionality that will fire when the window is opened. It is fired when the window is initially opened, as well as when the window is restored after a compile, as well as when the window is "redirected" to point to a new object if the user switches the window to point to another object. You may use this trigger to initialize settings in controls that may not have a coldlink.

**OnPreOpen:** This attribute allows you to execute functionality before the window is created. The `OnPreOpen` is fired after the tree structure for the window is created, but before the window itself is initialize. Unlike `OnOpen`, it is not executed after a compile or when the window is redirected. You could use the `OnPreOpen` to modify the structure of the window before it is opened, such as adding or removing tab windows, or adding or removing any controls from the tree structure of the window.

**OnClose:** This attribute contains text data with flexscript code that will be executed when the window is closed.

**palettewindow:** This attribute will cause the window's title bar to be smaller and look more like a tool window. The window will also always appear on top of other windows. The attribute needs no data.

**windowtitle:** This attribute defines the title of the window. It should have text data defining the title.

## Static

The static (or label) control is a control that simply shows text. If no special attributes are given to the control, then the name of the control will be shown as its text. Below are attributes that can be used with the static control.

**coldlink,hotlink,coldlinkx,hotlinkx,coldlinkname,hotlinkname:** The coldlink and hotlink attributes allow you to have the text be dynamic based on an attribute of the object. They work as documented above and define the text of the static control. Because the text of the static control is not editable, the links are not applied as they would be for other controls. You can also use `setviewtext()` and `getviewtext()` to explicitly get and set the text of the control.

**bitmap:** The bitmap attribute causes the static control to show a bitmap instead of text. The attribute should have text data that defines a path to the bitmap file, starting at the Flexsim main directory. The file must be a .bmp file. You can also specify within the bitmap file certain areas as "transparent", meaning the standard background color of the view will show through. To do this, the bitmap must be created in index color mode 24 bits per pixel, and the color that Windows will designate as transparent is the color R:192 G:192 B:192.

## Edit

The edit control shows text that can be edited by the user. If no special attributes are given to the control, then it will show the name of the control as its text. Below are attributes that can be used with the edit control.

**coldlink,hotlink,coldlinkx,hotlinkx,coldlinkname,hotlinkname:** The coldlink and hotlink attributes are most common with this type of control. They connect the text of the edit control with the value of an attribute or variable node of the object. They work as documented above and define what will appear in the text of the control. If the linked node contains number data, then the value will be copied into the control's text with the precision that the user specified in the main Edit menu. When the link is applied, the value in the edit control's text field is copied back to the linked node. You can also use `setviewtext()` and `getviewtext()` to explicitly get and set the text of the control.

**style:** One style that may be useful for the edit control is the `ES_READONLY` style. This causes the edit's text area to be gray and uneditable, but with the depressed border unlike a static, signifying a value that is feedback to the user but not an input from the user.

## Checkbox

The checkbox control is a button that has two states, check or unchecked. Below are attributes that can be used with the edit control. A box with a check in it appears, with text to the right of the box. The text of the checkbox is defined by the name of the control node. Below are attributes that can be used with the checkbox control.

**coldlink,hotlink,coldlinkx,hotlinkx:** The coldlink and hotlink attributes link the state of the checkbox to a value in an attribute of the object. The node that the cold/hotlinks link to should hold number data and have a value of 1 or 0. 1 will cause the box to be checked, 0 will cause the box to be unchecked. You can also explicitly check the box with `setchecked()` or get whether it is checked with `getchecked()` (1 means checked, 0 means unchecked).

**itemcurrent:** The itemcurrent attribute is actually added to the checkbox when an instance is created. Usually it signifies whether the box is checked, but it may not always. Use getchecked() to get the check state of the box.

**style:** A style that may be useful with the check box is the BS\_PUSHLIKE style. There is also a style for having the box be to the right of the text.

**bitmap:** The bitmap attribute causes the checkbox to have a bitmap next to it instead of text. It should be created the same as documented for the static view type.

## Radiobutton

The radiobutton control is much like a check box in that it can have two states, but it also has some extra functionality where only one radiobutton can be checked at one time within a container of radiobuttons. It allows the user to choose one of several options. Below are attributes that can be used with the radiobutton control.

**coldlink,hotlink,coldlinkx,hotlinkx:** The coldlink and hotlink attributes link the state of the radiobutton to a value in an attribute of the object. The node that the cold/hotlinks link to should hold number data and have a value of 1 or 0. 1 will cause the button to be checked, 0 will cause the button to be unchecked. You can also explicitly check the box with setchecked() or get whether it is checked with getchecked() (1 means checked, 0 means unchecked). Radio buttons present a problem because in order to have them work using just cold/hotlinks, you would need to connect each one to an individual node's value. Hence you would need 5 nodes in the object's attributes with only one having the value of 1 and the rest having 0. What users often want instead of this is to have a set of radiobuttons that reflect one value. For example, if a node has value 1, then radio button 1 should be checked; for value 2, radio button 2 should be check, and so forth. In order to do this you would need to write your own code in a coldlinkx or in the OnOpen of the window. Alternatively, you can use a combobox control. The combobox control fulfills the same functional requirement of choosing one of several options, but packages it all into one control that can be linked to a value on one node.

**itemcurrent:** Again, the itemcurrent attribute is added to the radiobutton automatically when an instance is created. Usually it signifies whether the box is checked, but it may not always. Use getchecked() to get the check state of the box.

**style:** Again, it may be useful to use the BS\_PUSHLIKE style here.

**bitmap:** The bitmap attribute causes the radiobutton to have a bitmap next to it instead of text. It should be created the same as documented for the static view type.

## Button

The button control is a push-able button. Below are attributes that can be used in a button control.

**apply:** If the apply attribute is added to the button, then Flexsim will call applylinks() on the button's owner view when the button is pressed. The attribute needs no data.

**close:** If the close attribute is added to the button, then Flexsim will close the button's owner view when the button is pressed.

**bitmap:** the bitmap attribute specifies a bitmap to be shown on the button. It should contain text data specifying a file path to the bitmap, starting at the Flexsim main directory. If the bitmap attribute is present then the name of the button will not be shown.

**coldlink, hotlink, coldlinkx, hotlinkx:** The coldlink attributes can be added to the button. They work as documented above and designate what will be shown as the button's text.

**OnPress:** The OnPress attribute specifies code that will be executed when the button is pressed. It should have text data containing flexscript code. Within the code, c accesses the button view.

## Combobox

A combobox is a box containing a dropdown list from which you can select options. Below are listed some of the attributes you can use with comboboxes.

**items:** The items attribute is a required attribute. It should contain subnodes. Each subnode represents an option in the combobox. The subnode's name specifies the text that will be shown in the option. You can manually add subnodes and give them names if your combobox options do not change. Often you will want the options in the combobox to be dynamic. You can do this by writing code in an OnOpen or some other trigger that populates the items attribute dynamically. Access the items attribute with the items() attribute command, clear the contents of it, then add nodes into the items attribute with nodeinsertinto(), then set their names with setname(). Once you've populated the list, set the itemcurrent attribute's value to the rank of the option that you would like the combobox to reference by default, then call comborefresh() to refresh the combobox windows options according to the new items list.

**itemcurrent:** The itemcurrent attribute is also a required attribute. It should have number data, and it specifies which attribute is currently selected.

**OnSelect:** The OnSelect attribute is a trigger that fires when the user selects an option in the combobox. The attribute should have text data with flexscript code specifying what to do when the user selects the option. Within this trigger you will want to access the value of the itemcurrent attribute to find out which option the user selected.

**picklist:** The picklist attribute allows you to redirect where the list of options is located. You may want to do this if you use several comboboxes in different GUIs, but all of the comboboxes use the same list of options. The picklist attribute should have text data which is a path to the pick list, starting at the picklist attribute. The pick list that is referred to should be structured just like the items attribute, with subnodes whose names designate the option text. When the window is opened, the referenced pick list will be copied into the items attribute of the combobox.

**pickitem:** The pickitem attribute causes the combobox to be linked to a number value on the object. The attribute does not need any data. If you give the combobox this attribute, then you will also need to give the combobox a coldlink attribute. For example, if you give the combobox a pickitem attribute and a coldlink with the text: @>objectfocus+>variables/arrivalmode, then this will link the combobox with the arrivalmode variable on the object. If the user chooses the second option in the combobox, then the arrivalmode variable will be given a value of 2. A third option selected results in a value of 3, and so forth. As an example, the Source's parameters page uses this attribute to tie the "Arrival Style" combobox to the Source's arrivalmode variable.

**pickcopydataonly:** The pickcopydataonly attribute causes the combobox to execute special functionality that allows for the implementation of Flexsim code pick lists. Although it could be used for other purposes, the main purpose is for code pick lists. In simple terms, the attribute causes the text of a given items subnode to be copied to a linked variable on the object. If you add the pickcopydataonly attribute, you will need to also add a pickprimary attribute and a picklist attribute. The pickprimary attribute acts much like a coldlink attribute. It links the combobox to an attribute of the object. When the combobox is initialized, the template code for the node referred to by pickprimary is copied into the combobox's items attribute as the first, or default, option. Then the referenced picklist is copied into the items attribute as additional options for the combobox. When the links are applied, then the text of the currently selected item is copied back to the node referenced by pickprimary. Note here that the subnodes of the items attribute now will have text data, and whatever item is picked, its text data is copied to the pickprimary reference. The existence of the pickcopydataonly attribute also effects some additional features with regards to the picklist attribute. The picklist attribute can contain subnodes that refer to other pick lists. If present, the subnodes should contain text data that is a path to a pick list, just like the picklist attribute itself. These pick lists will be appended on to the options of the combobox. Thus you can have the combobox provide a concatenated list of options from several different locations. The need for this feature came up because often different pick options may share a general list of options, but then have a unique list as well. For example, a setup time pick list may have all of the same options as the cycle time pick list, but it also needs a "Setup time if itemtype changes" option. So this feature allows the setup time pick list to mainly use the cycle time's "shared" pick list, but then add an additional option specific to a setup time. Another feature that is available when the pickcopydataonly attribute is used is with using code headers. If the main pick list node that is referred to by the picklist attribute contains text data, then that text data will be appended to the front of any of the options. This allows for a common header section for all options, and lets the options. For example, a header section may contain the code: "treenode current = ownerobject(c);". This code will be appended in front of the actual code for a given option. This method is used for all standard pick list comboboxes in flexsim, so you can find examples of this quite easily. You can also give the combobox a picklistheader attribute with text data designating the header code to use instead of what is found on the referenced pick list node.

**picklistheader:** This attribute designates a code header as explained in the pickcopydataonly attribute description above. It should have text data with code that represents the code header.

**pickprimary:** This attribute is used with pickcopydataonly, as explained above.

## Listbox

The listbox is similar to the combobox, except the options are not displayed in a window that drops down. Instead, then list is displayed in the window itself. Below are attributes that can be used with the listbox control.

**items:** The items attribute is required on the listbox. It represents the list of options in the list box. It acts just like the items attribute in the combobox.

**itemcurrent:** The itemcurrent attribute is also required. It also acts just like the itemcurrent attribute of the combobox.

**OnSelect:** The OnSelect attribute is a trigger. It acts just like the OnSelect attribute of the combobox.

## Script

The script control uses the Scintilla text editor. This is a flexible code editor that provides Flexsim with many code editing features. To learn more about the Scintilla text editor, go to [www.scintilla.org](http://www.scintilla.org). Below are attributes that you can add to a script control.

**coldlink:** The coldlink attribute links the text of the script control to the text of a node on the object. It works as documented in the general section above.

**noformat:** The noformat attribute causes the script control to be unformatted, meaning no code highlighting, line numbers, or folding will be done to the text of the control.

You can explicitly get the text of the control with `getviewtext()`, and set it with `setviewtext()`. You can also use a script control to display template code using the `codetotemplate()` command, which fills the template text of the control based on the code of a node, and the `templatetocode()` command, which modifies the code on a node based on the changes that the user has made to the template text. If you use the script control to show template code, you should give the control a noformat attribute.

## Table

The table control is a window that allows the user to see and edit a table or list of data. Below are attributes you can use with the table control.

**viewfocus:** The viewfocus attribute is required for the table control. It specifies what the table will "point at". It is like the objectfocus attribute. It should have text data that is a path to the node that contains the table information. There are two options for the structure of the table node itself. It can be a list, meaning it contains a number of subnodes. If the table node is a list, then the table control will show just one column of data, and the text in each entry is the data, either text or number, of each of the subnodes in the list. The row headers of the table are defined by the names of each subnode. The other option is to have the table node be an actual table. In this case

the table node contains a set of subnodes. Each subnode is one row in the table, and it contains subnodes, each being an individual entry in the table. The number of columns that will be shown in the table is defined by the number of subnodes of the first row. Row headers again are defined by the names of each row node. Column headers are defined by the names of the subnodes of the first row node. Please note that with a table, it is a direct view into the table data. This means that any changes made in an entry in the table will be applied as soon as you click off of the entry. This is different than using coldlink in other controls, where changes are only made when the user hits the Apply button. This is why the table control uses the viewfocus attribute, because it is a direct view into the data of a model, just like an ortho or perspective control.

**dataentry:** If you add a dataentry attribute to a table control, then this will allow the user to quickly traverse the table entries using the Tab and Enter keys as well as the arrow keys. This attribute will usually be used. The only time you may not want it is if you may want the user's Tab, Enter and arrow keystrokes to be captured within the entry itself, for example if a table entry may contain multi-line code.

**noformat:** Like the script control, the table control uses the Scintilla text editor. By adding the noformat attribute, Scintilla features like code highlighting, line numbers, folding, etc. are disabled. Usually this attribute should be present as it makes quick table editing easier.

**cellwidth:** This attribute specifies the default column width, in pixels, of the table. The attribute should have number data with the value in pixels. The cellwidth attribute can also contain subnodes. Each subnode should have number data that defines the column width of an individual column. The first subnode defines the column width of the row header column, the second subnode defines the first column's width, and so forth.

**cellheight:** This attribute specifies the height, in pixels, of each row in the table. Unlike table columns, all table rows must be the same.

**drawlines:** The drawlines attribute allows you to customize how lines between columns and rows are drawn. The attribute should have number data with a value between 0 and 3. A value of 1 will cause no separation lines to be drawn at all. A value of 1 causes both column and row separation lines to be drawn. A value of 2 causes only column separation lines to be drawn. A value of 3 causes only row separation lines to be drawn.

**list:** The list attribute designates the table as a list of node names. The attribute needs no data. The table will have only one column (no row headers column). In the single column, only the names of each subnode of the table will be shown.

## Groupbox

The groupbox is a control that groups several other controls together with a heading and a border around it. There are no special attributes for the groupbox.

## Panel



The panel control is like a groupbox but with a different border. You can give the panel a sunken border or no border at all. Below are attributes that can be used with the panel control.

**beveltype:** the beveltype attribute specifies what the border of the panel should look like. The attribute should have number data with a value between 0 and 2. A value of 0 will cause no border to be drawn. A value of 1 causes a one pixel sunken border to be drawn. A value of 2 causes a 2 pixel border to be drawn.

**bitmap:** You can give the panel a bitmap. To do this, add the bitmap attribute, give it the path to the bitmap file (like buttons\up\_arrow.bmp), then give the panel a viewfocus attribute with the following text: "..>bitmap". This will cause the bitmap to be shown on the panel.

**color:** You can also have the panel show a certain color. To do this, give the panel a color attribute as well as a viewfocus attribute. If the color you want to display resides on the object itself, then the text of the viewfocus should be something like: "@>objectfocus+>visual/color". If you would like to store the color on the view itself, then give the color attribute three subnodes, each with a number value between 0 and 1 representing the red, green and blue values respectively.

**splitterx, splitter:** The splitterx and splitter attributes designate the panel as a container for two resizable subcontrols. You should use one or the other but not both attributes. The attribute does not need data. To set up a splitter panel, add then the attribute, then add two subcontrols to the panel by dropping them inside the panel. The way the panel works is, if the mouse clicks and drags on any portion of the panel that is not part of a subcontrol, then the subcontrols will be resized. Thus you want to have the sub-controls take up the entire area of the panel except for a small "resizer" bar in the middle of the panel. As an example, drag a panel into your GUI and then give it a size of 200 x 200. Then drag two buttons into the panel. Give the first button a location of (0,0) and a size of (100,200). Give the second button a location of (105, 0) and a size of (95,200). Then give the panel a splitterx attribute. Then press F5 to go into preview mode. Notice the 5 pixel wide bar in the middle. Click it and drag the mouse right or left. Notice that the buttons will be resized as you drag. This example is not a very practical example, but the panel can be used with any controls. For a more useful example, look at the picklist code editor, which uses a splitter panel to show the template code and actual code simultaneously (VIEW:/standardviews/picklistedit).

## Tracker

The tracker control is a control that allows you set numeric values by visually dragging a locator along a trackbar. The shape factors GUI and the model speed tracker at the bottom of the main Flexsim window each use a tracker control. Below are attributes that can be used with the tracker.

**coldlink, hotlink, coldlinkx, hotlinkx:** These attributes link the tracker to a value on a node. They work as documented in the general section above.

**itemcurrent:** The itemcurrent attribute is a required attribute. Its value holds the current value that has been selected for the tracker.

**rangemin, rangemax:** The rangemin and rangemax attributes specify the min and max values of the tracker. They should have number data specifying the min and max values respectively.

**rangeexp:** This attribute tells the tracker to exponentially increase the value as the locator is dragged to the right, instead of linearly. It should have number data specifying the factor to exponentially increase by. The run speed control at the bottom of the Flexsim window uses this attribute.

**OnSelect:** This attribute allows you to execute code when the user drags the locator to a given position. The attribute should have text data with flexscript code specifying what to do. Within the function, c will access the tracker control. To get the currently selected value, access the value in the itemcurrent attribute.

## Tabcontrol

The tabcontrol is a control that contains sub-controls that are each a tab page. The add a tab page, drag a control into the tab. Below are attributes that can be used with the tabcontrol.

**pagelist:** The pagelist attribute allows you to have an external source for the pages of the tab control. The attribute should have text data specifying a path to a node that contains subnodes. Each of the subnodes should contain text data that is a path to the page control. The pagelist attribute is used quite often in object Parameters windows.

**itemcurrent:** The itemcurrent attribute is required. It allows you to know which tab is currently selected. It should contain number data whose value will be set whenever the user chooses a tab.

**OnSelect:** The OnSelect attribute lets you execute flexscript code when a tab page is selected. The attribute should have text data that will be fired when the user selects a tab. Within the code, c accesses the tabcontrol view. Use get(itemcurrent(c)) to get the currently selected page.

## Statusbar

The statusbar control adds a status bar to the bottom of another window. You can set the text of the statusbar with setviewtext(). Below are attributes you can use with the status bar:

**coldlink, hotlink, coldlinkx, hotlinkx:** These attributes link the text of the status bar to the value on a node. They work as documented above.

## Icongrid

The icongrid control provides a drag-drop mechanism for the user. The library icon grid is an icongrid control. The GUI builder also uses icongrids to drag-drop controls and attributes into the GUI. Below are attributes that can be used with the icongrid.

**viewfocus:** The viewfocus attribute is a required attribute. It should have text data that contains a path to the view focus of the icon grid. The icongrid will display each

subnode of the view focus as a drag-able rectangle. In order for a given sub-node to be displayed, it must have object data, and it must have a picture attribute. If not, the object will not be shown at all in the icongrid. The picture attribute of the object may contain text data that specifies a path to a bitmap file, starting in the Flexsim main directory. If the path is valid, then the picture will be shown in the object's rectangle. Otherwise, only the name of the object will be shown.

**viewwindowsource:** This attribute is required if you want the icongrid to be draggable. It should have number data with the value of 1, meaning yes you want the user to be able to drag objects from the icongrid onto other views.

**cellwidth:** This attribute allows you to define the width, in pixels, of each rectangle in the icongrid. It should have number data containing the desired width.

**cellheight:** This attribute allows you to define the height, in pixels, of each rectangle in the icongrid. It should have number data containing the desired height.

**displaygroup:** This attribute lets you display just a sub-group of the view focus. The attribute should have text data specifying a name for the sub-group, like "Manufacturing". Then, each object in the view focus should also have a displaygroup attribute with text. If the displaygroup of the object matches the displaygroup of the icongrid, then the object will be shown in the icongrid. To change the group that you want shown, just change the text of the icongrid's displaygroup attribute, then call `repaintview()` on the icongrid.

**viewhidealllabels:** This attribute causes the objects' names not to be shown, but only their pictures. The attribute should have number data with a value of 1.

**viewbackgroundcolor:** If this attribute is present, then the icongrid will not be shown as a set of raised buttons, but instead will just paint the pictures and names on top of a background you specify. The attribute should have 3 sub-nodes, each with number data between 0 and 1 corresponding to the red, green and blue components respectively.

**depresshighlighted:** If this attribute is present, then the view's highlighted object will have a sunken border instead of a raised border. The attribute does not need any data. To access the view's highlighted object, use the `selectedobject()` command.

**picturealignleft:** If this attribute is present, then each object's icon will be shown to the left of the object's name, instead of above it and in the center. The attribute should have number data specifying the width, in pixels, to make available to the left of the object's name for the object's picture.

**OnDrag:** This attribute allows you to execute code when an object is dragged from the icongrid onto another view. The attribute should have text data with flexscript code. Within the function, `c` gets access to the icongrid, `i` gets access to the view on which it was dropped, `dropx()`, `dropy()`, and `dropz()` get the drop location if the view is a ortho, perspective, or planar view, `dropnodefrom()` gets access to the object that was dragged, and `dropnodeto()` gets access to the object it was dropped onto if one

exists. Please note that if no OnDrag attribute exists, then a copy of the object will be made in the dropped view's focus or in droptodet() if it exists.

## Tree

The tree view into flexsim's tree. Below are attributes that can be used with the tree control. For the most part, necessary attributes are added automatically for you. This will document only the attributes that you will need to know about.

**viewfocus:** The viewfocus specifies the focus of the tree view. It should have text data with a path to the node that will be the head of the tree.

**cellwidth:** The cellwidth attribute lets you specify how wide, in pixels, each node's name will be shown.

**noformat:** If this attribute is present, then when text data is edited it will not be formatted for code.

**viewpointx:** This attribute is required and has number data specifying the x viewpoint, in pixels, of the tree view.

**viewpointy:** This attribute is required and has number data specifying the y viewpoint, in pixels, of the tree view.

## Graph

The graph control lets you show data in a graph format. It can be presented as a pie chart, a bar graph, a line graph, a histogram, or a scatter plot. The graph is updated on the fly, so you can have real-time model data be shown as the model runs. Below are attributes that can be used with the graph.

**viewfocus:** The viewfocus attribute is required. It should have text data containing a path to a node that contains data for the graph. The structure of the data depends on the type of graph. If the graph is a pie chart, then the view focus node should have text data that specifies the name of the chart. If the graph is a line graph, scatter plot, or bar graph, then again the viewfocus node should have text data that specifies the title of the chart, and the viewfocus should have sub-nodes. Every odd-numbered sub-node will represent a point on the x axis, and every even-numbered sub-node represent its corresponding y value. If the graph is a histogram, then the viewfocus node should have three sub-nodes, each having number data. The first sub-node should be the minimum value of the histogram range. The second sub-node should be the maximum value of the histogram range. The third sub-node should be the number of "buckets" or divisions in the histogram.

**graphgrid:** If this attribute is present, then a grid will be shown as a background for the graph. The width of the grid does not correlate to any set value range in the graph, but is rather for comparative purposes, for example to compare the height of two bars in the graph. The attribute does not need any data.

**graphtitle:** If this attribute is present, then the grid will show a title for the graph. This is usually the text data on the viewfocus node except in the case of a histogram,

where it is the text data on the graphhistodata focus. The attribute does not need any data.

**graphaxes:** If this attribute is present, then the grid will show the min and max values of the x and y range for the graph. The attribute does not need any data.

## Histogram Attributes

To create a histogram, you should add the following attributes. For an example of the structure needed for a histogram, view the tree of any Flexsim object and go to the tree at >stats/staytime/stats\_staytimehisto.

**graphhistodata:** If this attribute is present, then the graph will show a histogram. The attribute should have text data containing a path to a node that contains "bucket" sub-nodes. The number of sub-nodes should be the number specified by the viewfocus plus 2. The first sub-node will be designated as the "underflow" node, where any values less than the minimum range value of the histogram will be added to this node. The last node is "overflow" for values that are greater than the histogram's maximum range. Every other sub-node represents the histogram value for the "bucket" for the sub-node's corresponding interval.

**graphhistointervaldata:** You can also allow Flexsim to calculate a confidence interval on the mean of the histogram data by adding this attribute. This attribute should contain text data with a path to a node with 5 sub-nodes. The node itself should have number data that is either 1 or 0. If 1, the confidence interval will be shown on the graph. The first three sub-nodes are used by Flexsim and should have number data. The 4th sub-node should be the percent confidence, and the 5th sub-node should have number data with the value 1, telling Flexsim to automatically calculate the interval.

**graphbars:** This attribute signifies that the graph will be shown as bars. The attribute needs no data.

## Pie Chart Attributes

To create a pie chart, add the following attributes.

**graphpie:** This attribute signifies that the graph will be shown as a pie chart. The attribute needs no data.

**graphpiedata:** This attribute should have text data that contains a path to a node with sub-nodes. Each sub-node's name will be shown in the pie chart's legend, and should have number data that is the value for that item in the pie chart. The sum of all sub-nodes of the focus node will represent a full 360 degree circle, and each sub-node's value represents a chunk of the pie.

## Line Chart, Bar Chart, Scatter Plot Attributes

If the graph is not a pie chart or histogram (and it has a graphxy attribute), then the sub-nodes of the focus node will be interpreted as paired values to be plotted on the x and y axes of the chart. Odd-numbered sub-nodes represent x values and even-

numbered sub-nodes represent y values. Use the following attributes to define how those values will be drawn. The attributes can be combined as needed.

**graphxy:** Add this attribute to make each sub-node pair be interpreted as an x/y value. If this attribute is not present, then each sub-node (not sub-node pair) will be interpreted as a y value, and the points will be evenly distributed across the x-axis, the first point with the x value 1, the second with x-value 2, and so forth. You may want to leave this attribute out if you want to display a bar chart where the bars are evenly distributed across the x axis.

**graphlines:** Add this attribute to have a line drawn between consecutive points in the graph. The attribute needs no data.

**graphpoints:** Add this attribute to have a dot drawn at each point in the graph. The attribute needs no data.

**graphbars:** Add this attribute to have a bar drawn from 0 to the y value at each point in the graph. The attribute needs no data.

**graphstep:** If this attribute and the graphlines attribute are present, then each line will be drawn first horizontally and then vertically to the next point, creating a stair-step effect instead of diagonal lines.

**gridx, gridy:** If the graphgrid attribute is present, then you can also add a gridx and/or gridy attribute, which specifies the x/y grid interval.

**graphscatterdata:** This attributes causes the structure of the data to be redefined. The attribute should have two sub-nodes, each containing text data defining a path to a container node. The first sub-node's focus node contains a list of "x" data points, and the second sub-node's focus node contains a list of "y" data points. Each point on the graph consists of an x-value from the first focus node paired with a y-value from the second focus node. If this attribute is present, then the viewfocus attribute will be ignored.

**dataseries:** The dataseries node is actually not an attribute on the graph control. Instead, it should be placed in the same container as the focus node (>viewfocus+/.). If the graph control sees that this node is present, then it will split the graph up into several uniquely colored data series. This allows you to have a line graph with multiple colored lines, or a bar graph with multiple colors and a legend. The dataseries node should have sub-nodes, each representing one data series. The name of each sub-node will be shown in the graph's legend. Each sub-node should also contain number data that signifies how many points are in that data series. The value of the sub-node specifies an "up-to" point in the data. Take for example a line chart with 20 points. Normally the graph will simply draw a red line between each point in the graph. However, let's say you want to split the 20 points into 3 categories. The first 7 points represent the content graph for Object A, the next 10 points represent the content graph for Object B, and the last 3 points represent the content graph for Object C. To split these data series, add a node called dataseries to the content graph's container node. Then add three sub-nodes, named Object A, Object B, and Object C. Then give each sub-node the respective values: 7,

17, 20. This designates Object A's series as points 1-7, Object B's as points 8-17, and Object C's as points 18-20. The graph should now show three uniquely colored lines, as well as a legend.

## Ortho/Perspective

The ortho/perspective view type is a 3D view. The GUI builder's ortho and perspective views include many default attributes that are not documented in detail here, but you can experiment with those attributes from within the GUI builder, as well as look at the standard ortho view and its settings window to see what attributes do what. We document below attributes essential to an understanding of the ortho and perspective views.

**viewfocus:** This attribute defines what the ortho view is "looking at". It should contain text data with a path to the node containing the objects to be viewed.

**viewprojectiontype:** The viewprojectiontype attribute specifies whether the view is an ortho or a perspective view. It has number data, 1 signifying ortho, 0 signifying perspective.

**viewpointx,viewpointy,viewpointz,viewpointrx,viewpointry,viewpointrz:** These attributes specify the focal point and viewing angle of the view.

**viewmagnification:** This attribute is only used with ortho views, and specifies the magnification or "zoom" of the view.

**viewpointradius:** This attribute is only used with perspective views, and specifies the camera's distance from the focal point.

**OnDropNode:** This attribute should contain text data and defines flexscript code that will be executed when the user holds a key down while click-dragging from one object drawn in the view to another.

## Planar

The planar view type is a 2D view. The GUI builder's planar view includes many default attributes that are not documented in detail here, but you can experiment with those attributes from within the GUI builder, as well as look at the standard planar view and its settings window to see what attributes do what. We document below attributes essential to an understanding of the ortho and perspective views.

**viewfocus:** This attribute defines what the planar view is "looking at". It should contain text data with a path to the node containing the objects to be viewed.

**viewprojectiontype:** The viewprojectiontype attribute specifies whether the view is an ortho or a perspective view. It has number data, 1 signifying ortho, 0 signifying perspective.

**viewpointx,viewpointy,viewpointz:** These attributes specify the focal point of the view.

**viewmagnification:** This attribute specifies the magnification or "zoom" of the view.

**OnDropNode:** This attribute should contain text data and defines flexscript code that will be executed when the user holds a key down while click-dragging from one object drawn in the view to another.



# Index

## 3

3D File .....	668
3D Media .....	667

## A

ASRSvehicle .....	337
ASRSvehicle Page.....	221
Attribute Hints.....	203
Attributes .....	701
AutoCAD .....	672
AVI Maker.....	430

## B

Basic Modeling Functions .....	661
BasicFR.....	339
BasicFR Page .....	222
BasicTE.....	340
BasicTE Page .....	226
Bay .....	546
Break To Requirement.....	557
Breakdown/Repair Trigger .....	536
Build Menu .....	189

## C

C++.....	655
Code.....	655
Collision Page .....	230
Collision Trigger .....	527
Combiner.....	341
Combiner Page .....	233
Command Hints .....	204
Comparison Chart.....	620
compile.....	682
Completion Hints .....	205
Conveyor Page .....	234
ConveyorLayout Page .....	238
Coordinated Task Sequences .....	583
Crane.....	348
Crane Page .....	240
Creation Trigger .....	526
Custom Built Task Sequences .....	577
Custom Draw Code.....	531
Custom Libraries.....	693
Cycle Time .....	543

## D

Database Table View .....	217
Database Tables .....	648
Description .....	9
Dispatcher.....	349
Dispatcher Page.....	242
Down Dispatcher.....	544

## E

Edit Highlighted Object Toolbar .....	180
Edit Menu .....	186
Elevator .....	350
Elevator Page.....	243
Entry Trigger .....	523
Events .....	439
Excel.....	489
Excel Import .....	469
Excel Interface .....	432
Execute Menu .....	190
Exit Trigger .....	523
Experiment .....	560, 565

## F

Fall Through Mark Triggers .....	551
Financial Report .....	629
Find Objects Toolbar.....	181
Find Replace .....	206
FixedResource Page .....	244
Flexscript.....	655
Flexsim Concepts .....	134
Flexsim Toolbar.....	198
Flow Page .....	245
Flow Rate .....	545
Flowitem Bin.....	433
Flowitem Gantt Chart .....	645
Flowitem Page .....	248
FlowNode .....	356
FlowNode Page.....	249
Fluid Objects .....	412
FluidBlender .....	290
FluidBlenderRecipe.....	292
FluidGenerator .....	293
FluidLevelDisplay .....	295
FluidMixer.....	297
FluidMixerSteps .....	299
FluidPipe .....	419
FluidPipeLayout .....	303
FluidProcessor.....	421
FluidSplitter .....	307
FluidSplitterPercents.....	309
FluidTank.....	424
FluidTankMarks.....	312
FluidTerminator .....	313
FluidTicker.....	314
FluidToItem .....	315
Frames.....	674

<b>G</b>	
Gantt Chart.....	635, 645
General Properties Page .....	325
Global Tables .....	434
Global Task Sequences.....	441
Global Time Tables .....	436
Global User Events .....	439
Global Variables .....	444
Graphical User Interfaces .....	446
Graphs.....	649
Groups.....	182
<b>H</b>	
Handle Collision .....	527
Help Menu .....	196
<b>I</b>	
Import .....	489
Import Media Files .....	463
Imported Shape.....	397
Importing 3D Media.....	667
Importing AutoCAD Drawings.....	672
Initial_Product .....	317
Inter-Arrivaltime Usage .....	542
ItemToFluid .....	428
<b>K</b>	
Keyboard .....	159
Kinematics.....	685
<b>L</b>	
Labels Tab Page .....	327
Layouts .....	179
Level .....	547
Level Of Detail.....	675
Libraries.....	693
Library .....	335
Light Source Editor .....	175
Load Time .....	539
Load Trigger .....	522
Logic.....	655
<b>M</b>	
Media.....	667
MergeSort .....	358
MergeSort Page .....	250
Message Trigger .....	524
Minimum Dwell Time.....	540
Minimum Staytime .....	540
Mode Toolbar .....	176
Model Description .....	9
Model Layouts Toolbar .....	179
Model Startup Code .....	464
MTBF.....	465
MTTR .....	465
MultiProcessor .....	361
MultiProcessor Page.....	252
<b>N</b>	
NetworkNode .....	256
NetworkNode Connection Page .....	254
Networks .....	178
Node Entry Trigger.....	528
<b>O</b>	
Object Comparison Chart .....	620
Object Gantt Chart .....	635
Object Trigger Functions .....	288
On Continue/On Arrival.....	528
OnChange Trigger .....	530
OnCover Trigger .....	529
OnDraw Trigger.....	531
OnEmpty .....	533
OnFull.....	533
OnResourceAvailable .....	537
OnUncover Trigger .....	529
Operator .....	371
Operators Page.....	257
Optimizer.....	473
Ortho Popup Menu.....	173
Orthographic View Window .....	143
<b>P</b>	
Pass To.....	558
Performance Measure .....	564
Perspective View Window .....	143
Photo Eyes.....	259
Pick Lists.....	519
Pick Operator .....	544, 548
PlaceOffsetTask.....	589
Planar View .....	150, 166
Plot .....	625
Preferences Window.....	207
Presentation Builder.....	478
Process Dispatcher.....	548
Process Finish.....	525
Process Finish Trigger.....	525
Process Time.....	543
Processor.....	373
Processor Page.....	260
ProcessTimes Page.....	261
Pull From Port.....	549
Pull Requirement .....	550
<b>Q</b>	
Queue.....	375
Queue Page .....	262
Queue Strategy .....	559
<b>R</b>	
Rack.....	377
Rack Page.....	264

Rack Size Table.....	266	Task Sequence .....	573
Receive .....	351	Task Sequence Preempting .....	581
Receive From Port.....	549	Task Sequences .....	441, 577, 579, 583
Recorder.....	379	Task Types.....	587
Replication .....	561, 566	TaskExecutor Page.....	281
Reports.....	210	Text Code.....	571
Request Transport From.....	555	Text Display .....	570
Reservoir .....	380	ticker.....	320
Reservoir Page .....	275	Time Picklist.....	542
Reset Trigger .....	534	Time Picklists .....	538
Rise Through Mark Triggers .....	551	Time Plot .....	625
Robot.....	381	Time Tables .....	436
Robot Page .....	276	Tools Introduction .....	429
Run Panel .....	200	Tools Menu .....	193
<b>S</b>		Trace Debugger .....	215
Sample Models .....	209	Traffic Control Network Node Page..	283
Scenario .....	562, 567	Traffic Control Page.....	284
Schedule .....	436	TrafficControl.....	392
Script Editor.....	481	Transparency .....	677
Send Requirement .....	552	Transport Dispatcher .....	555
Send To Port .....	553	Transporter.....	396
Separator.....	382	Transporter Page .....	287
Separator Page .....	277	Travel Networks Toolbar .....	178
Setup Time.....	541	Tree Browse Dialog .....	216
Simulation Experiment Control .....	482	Tree Structure .....	679
Single Object Chart.....	639	Tree Window .....	151
Single Table Export.....	488	<b>U</b>	
Single Table Import.....	489	Unload Time.....	539
Sink .....	384	Unload Trigger .....	522
Sink Page .....	278	User Commands .....	494
Sky Box Editor.....	490	User Events.....	439
Source.....	385	UserLibraries.....	153
Source Page .....	279	<b>V</b>	
Speed .....	556	Variables .....	444
Split/Unpack Quantity .....	554	View Attributes Reference .....	701
State List .....	683	Views Menu.....	168, 187
State Report .....	618	Visio Import .....	496
Statistics.....	210	<b>W</b>	
Statistics Menu.....	191	Warmup.....	563
Statistics Properties Page.....	329	Watch List.....	518
Summary Report.....	616	Welcome To Flexsim .....	1
<b>T</b>		What's New .....	3
Table Configurator .....	492	Writing Code .....	655
Table Editor .....	218	<b>X</b>	
Table View .....	217	X Value .....	568
Table Y Val.....	569	<b>Y</b>	
Tables.....	434	Y Value .....	569