USER MANUAL

FlexSim 2017 Update 1

Date Published
10 APR 2017

Table of Contents

FixedResources

- Concepts
- BasicFR
- Combiner
- MultiProcessor
- Processor
- Queue
- Rack
- Separator
- Sink
- Source

TaskExecuters

- Key Concepts
- Interface
- Basic Concepts Tutorial
- Concepts
- Advanced Concepts Tutorial
- ASRSvehicle
- BasicTE
- Crane
- Dispatcher
- Elevator
- Operator

Distribution Chooser

- About the Distribution Chooser
- Using the Distribution Chooser
- Exponential
- Duniform
- Normal
- Triangular
- Uniform

Robot

- Overview
- Moving Flowitems
- Motion Paths
- Event List
- Event Log
- Transporter
- Excel Interface

Travel Networks

- NetworkNode
- Concepts
- TrafficControl
- Reference

Flowitem Bin

- Global Tables
- Global Variables

Visual

- Overview

Graphical User Interfaces

- Example

About Process Flow

- Why Use Process Flow? o Overview of Process Flow
- Navigating in the Process Flow Window o Running a Simulation o Types of Process Flows

Building a Process Flow

- Introduction o Overview of Objects o Adding and Connecting Activities o Moving, Resizing, Deleting Activities
- Editing Activity Properties o Common Properties

Linking Process Flow to 3D Models

- Linking Process Flow to 3D Models
- Event-Listening Activities
- Event Types o General Properties o Labels

Changing Process Flow Visuals

- Changing Process Flow Visuals
- Using Process Flow Themes o Troubleshooting a Process Flow

- Introduction
- Creating and Using Charts
- Chart Properties
- Process Flow Statistics
- About Zones
- Sub Process Flows
- Task Sequences
- Process Flow Instances
- Process Flow Variables
- User Libraries
- Coordination
- Preemption

Activities

- Inter-Arrival Source o Schedule Source o Event-Triggered Source

Basic

- Assign Labels
- Delay
- Custom Code
- Decide
- Batch
- Organizing Batches
- Releasing Batches
- Batch Statistics
- Wait for Event
- Create Tokens
- Sink o Run Sub Flow o Start o Finish o Change Visual o Run Animation o Create Object o Move Object o Destroy Object o Travel o Load o Unload o Delay o Travel to Loc o Custom Task o Create Task Sequence o Dispatch Task Sequence o Split o Join o Synchronize o Save Token Context o Release Token o Restore Token Context

Shared Assets o About Shared Assets

List

- List
- Push to List
- Pull from List

Resource

- Resource
- Acquire Resource
- Release Resource

Zone

Zone

- Zone
- Zone Reference
- Zone Properties Window
- Enter Zone
- Exit Zone

Display Objects

o Text o Arrow o Image o Flow Chart

Conveyors

- About Conveyors
- Introduction to Objects
- Adding Conveyors
- Moving, Resizing, and Reversing
- Connecting Conveyors

Flow Control

o Introduction o Sorting o Merging and Slug Building o Gapping o Power and Free Systems

Pick List Behaviors

o Introduction o The Condition Field o Send Item o Stop/Resume
o Area Restriction o Movement o Set Conveyor Speed

The Conveyor System Tool

o Introduction to the Conveyor System Tool o Accessing Type Properties o Conveyor System Settings o Managing Conveyor System Types o Conveyor Type Settings
o Transfer Type Settings o Decision Point Type Settings o Photo Eye Types Settings o Station Type Settings o Entry Transfer Types Settings o Exit Transfer Type Settings

Objects

o Straight and Curved Conveyors o Join Conveyors o Decision Point o Photo Eye o Station o Motor o Merge Controller o Transfers

AStar

- Concepts
- Example

- Reference

AGV

AGV Network Properties

o AGV Types o Way Points o Control Point Connections o Accumulation Types o Deallocation Types o Visual o General

Objects

o AGV o Path o Control Point o Control Area

- Pick-List Actions

Welcome to the FlexSim User Manual

If this is your first time using FlexSim, try some of the following suggestions:

1. Go to FlexSim's YouTube Channel and watch how to build your very first model.

2. Read the Getting Started section of this User Manual and follow along by building the model described.

3. Work through the Tutorials described here in this User Manual (only the first two can be completed with the Trial Version).

Have some fun by building your own model from scratch!

Feel free to contact our technical support staff if you have any questions while evaluating the software. FlexSim Technical Support can be reached Monday - Friday, 8:00 am - 5:00 pm MST. You may call 801224-6914, e-mail your questions using a web form at http://www.flexsim.com/support/, or post your questions on our worldwide FlexSim Community Forum (http://www.flexsim.com/community/forum/). We hope you enjoy learning how FlexSim can help you optimize your flow processes.

What's New in FlexSim

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

FlexSim 17.1.0

- Changed FlexSim to store strings using UTF-8 encoding.
- Added support for Oculus Touch controllers.
- Implemented OpenVR for HTC Vive compatibility.
- Improved the shadow frustum calculation in VR so that shadows look better.
- Added a global preference for changing the resolution of the shadow map.
- Added support for nested queries in SQL.
- Added ROW_NUMBER as a SQL keyword.
- Implemented F2 and Esc functionality in tables.
- Updated the table view and labels tab to be more robust with different datatypes.
- Added Table(name) method for referencing Global Tables.
- Added more FlexScript classes, including List, AGV, Conveyor, and TrackedVariable.
- Added more properties and methods to existing FlexScript classes, including Object stats and additional string methods.
- Improved scientific notation for literals in FlexScript.

- Added a start value to tracked variables.
- All tracked variables in the model now reset on model reset.
- Changed itemtype references to referencing an item's type label instead.
- Improved the Person flowitem's 3D shape.
- Added repeating events to time tables.
- Added a short description field to user commands.
- Made the gantt charts and time charts scroll with a fixed time window.
- Removed the global table Clear on Reset checkbox and replaced it with a reset trigger.
- Added new visualization options for the Rack.
- Added duplicate buttons to the Excel Interface window.
- Added a duplicate option to the Toolbox's context menu.
- Taskbar now shows experimenter/optimizer status and runtime based upon stop time.

- Disabled deleting objects while the model is running.
- Fixed an issue with the undo history when pasting over nodes with pointer data.
- Fixed issues with using the ternary operator after properties.
- Fixed an issue with writing to Access databases with read-only fields.
- Included fixes listed in 17.0.4 below.

Backwards Compatibility Note: the following changes may slightly change the way updated models behave.

- Removed the FlexSim WebServer application from the default installation and developed a new WebServer application using Node.js that streams the 3D view much faster. The new WebServer can be downloaded through FlexSim's Online Content.
- Existing models will continue to work with itemtype, but new models should be built using a type label instead of the itemtype attribute and commands.

Process Flow

- Added an Assign Released Resource(s) To field on the Release Resource activity.
- Added functionality to allow you to Ctrl+Drag activities in an activity block.
- Added a right-click menu option and Alt+Click to open the Token View.
- Added a sampler to the assign labels Label Name field so you can sample other activities or tokens to get label names.
- Added a right-click menu option to open multiple views of a Process Flow.
- Added an Assign To property to the Create Tokens activity.
- Added a Make Array label aggregation type to the Batch.
- Added Center in View buttons for fields with pointer data.
- Added a name property to the Token FlexScript class.

- Fixed a bug with duplicating Process Flows using the Toolbox.

Backwards Compatibility Note: the following changes may slightly change the way updated models behave.

- Updated the Pull From List activity so it does not assign a null label when nothing was pulled or the token was released early (manually). Previously, if you used a Max Wait Timer or Max Idle Timer (or some other mechanism) to release a token from a Pull from List prematurely, the label specified in the Assign To field would be created with a value of NULL. Now, the label will not be created. This may break other models that are checking to see if the label value exists. For example, saying objectexists(token.pulled) will throw an exception if the pulled label is never created. This can be easily remedied by changing the

code to objectexists(token.pulled?). The ? will cause the value returned to be nullvar when the pulled label does not exist.
- Universal Edit fields are now more strict when accessing labels on a token. Previously, typing token.labelThatDoesNotExist would happily return NULL and move on. Now the Universal Edit will throw an exception if the label doesn't exist. This does not include Universal Edit fields that assert labels, for example the Assign To fields.

Conveyor

- Fixed an issue with aborting transports on a missed pick at an exit transfer.

FlexSim 17.0.4

- Added better error message information on certain exceptions.
- Fixed an issue with the [] operator throwing an exception when the node doesn't exist.
- Fixed some autocomplete exceptions in the code editor view.
- Fixed a bug with autocomplete when identifiers have an underscore.
- Fixed a memory leak in the Table class.
- Fixed an issue with casting from a class type to a new object of the same class type.
- Fixed hanging issue with function-like macros that are instantiated with the same parameters as those passed.
- Fixed a bug with inObjects and outObjects not working properly with internal container connections.
- Improved the error message when executing a single-line script command that doesn't return a value.
- Fixed a bug with the menus not correctly graying in certain situations.
- Improved the error handling of watch variable evaluation.
- Fixed a bug with the query() command when using the debugger.
- Fixed an exception in the debugger.
- Fixed an issue with the ++ operator on label properties.
- Fixed a bug with the code headers for the TrafficControl triggers.
- Fixed a bug with the Passing drop-down list on the Network Properties window.
- Fixed a bug with the Command Helper view not updating correctly in certain cases.
- Fixed a crashing issue with a sampler in the Video Recorder window.
- Fixed an issue with the Processor's Use Operator error message not appearing correctly.
- Fixed a bug with tools deleting themselves on reset if created during a model run.
- Fixed a bug with the initial positioning of curved network edge spline points.
- Fixed an issue with the Script Console losing text when undocking a neighboring tab.
- Network Nodes now highlight when hovered over in the 3D view.
- Fixed a UI issue with units in dashboard chart Properties windows.
- Fixed an issue with the Custom Chart collecting data beyond its max time interval.
- Fixed Statistical Distributon popups not showing the submenu correctly.
- Fixed a bug with transporting batches of flowitems not working properly in certain situations.
- Fixed an issue with certain GUI Builder menu options not working properly.
- Added a global preference for disabling auto-insert parentheses and quotes.

Process Flow

- Fixed a bug with assigning a label to a task sequence with the Assign Labels activity.
- Fixed a bug with adding Process Flow objects to user libraries.
- Fixed a bug with the Wait For Event properties window.
- Fixed a bug with tying an object with a TE/FR Process Flow to a Resource with a count greater than one.
- Updated the Conditional picklists so they no longer evaluate both the true and false statement.
- Fixed an issue on the List with having backorders with no pullers.
- Fixed a bug with copy/pasting Zones.
- Fixed a bug with chart time windows.

Conveyor

- Fixed an issue with decision points and photo eyes not snapping to the conveyor in certain situations.
- Fixed a bug with transfer statistics.
- Fixed some drawing issues with invalid conveyors.
- Fixed an issue with dragging multiple selected conveyors and decision points.
- Fixed a bug with conveyor virtual length.
- Fixed a bug in the accumulation gap calculation algorithm.

AGV

- Fixed curved paths being draggable when path draw mode was set to clickable only.

FlexSim 17.0.3 (March 7, 2017)

- Fixed a bug when compiling with Visual Studio 2015.
- Fixed a bug with querying list partitions by their index.
- Fixed a crashing issue with large arrays in tables.
- Fixed an issue with fractions in the convert() command.
- Fixed some issues with autocomplete and code highlighting.
- Fixed bugs with certain FlexScript operations not working correctly with the Table's [][] operator.
- Fixed adding lists from a user library.
- Fixed an issue with exporting dashboards to html.
- Fixed a bug in the Export Results to CSV File pick option.
- Fixed some UI bugs in the Excel Interface window.
- Fixed an issue with combined dashboard stats used as a performance measure.
- Updated user libraries so that replaced time tables, mtbf/mttr, and lists retain their members and contents.
- Fixed some bugs in the legacy conveyor's normals and texture coordinates.
- Fixed a bug with saving/loading multi-dimensional arrays.
- Fixed a crashing bug with accessing certain arrays out-of-bounds.

Process Flow

- Added a message when trying to sample a Global List from a Push/Pull from List activity.
- Fixed an issue that caused quick properties to sometimes update incorrectly.
- Fixed an issue that caused tokens to not move through activities in certain circumstances.
- Fixed the Assign Labels name fields so you can type in text that is the name of an object.
- Fixed the edit fields so they properly handle all of the Token's properties.
- Fixed the Token's instance property so it returns the correct object.
- Added right-click options for assigning number or string data to Schedule Source label columns.

Conveyor

- Added an error message if the target fill percent is too large for slug build conveyors.

AGV

- Fixed a routing error where AGVs couldn't find a path in certain situations.
- Fixed a bug with waypoints when using agredirect() to switch directions.
- Fixed a bug with AGVs running over each other on an accumulating path after a preempt.
- Fixed an exception in agvinfo(agv, AGV_ACCUM_AHEAD_AGV).
- Fixed an issue with adding cp connections, then removing them and adding a different type of connection.

FlexSim 17.0.2 (February 2, 2017)

- Fixed a memory leak that caused the views to crash unexpectedly.
- Fixed assigning a table value directly to a property.
- Fixed a bug with parenthesis matching when using string literals containing parentheses.
- Fixed a bug with global table Clear on Reset not working properly in certain cases.
- Fixed default datatype value of settablesize() in C++.
- Fixed a bug with drawtext() default parameters in FlexScript.
- Fixed Travel to Location pick option.
- Fixed an issue with setting item state incorrectly when using lists in Send To Port.
- Included fixes listed in 16.0.9 below.

Conveyor

- Fixed a crash that could happen when transporting items between an exit and entry transfer.

AGV

- Fixed an exception when listening to OnAGVAllocationFailed.
- Fixed an issue involving AGVs with trailers not routing correctly in certain cases.

FlexSim 17.0.1 (January 19, 2017)

- Fixed the updatekinematics() command.
- Fixed a memory leak in the Table class.
- Fixed a crash with executing certain switch cases, such as the Date/Time Text display.
- Fixed a bug with the tonode(Variant) overload.
- Fixed an issue with setting treenode properties in certain cases.
- Fixed an issue with setting label properties on a Variant in certain cases.
- Fixed some bugs with certain executetablecell() overloads.
- Fixed the lognormalmeanstdev() command parameters.
- Fixed the function_n() command definition.
- Fixed a crash when updating some user libraries.
- Fixed a bug with Navigator connections when adding TaskExecuters with the Experimenter.
- Fixed a bug with autocomplete for current labels within code editors.
- Updated the Event List view so it displays message parameters.
- Fixed an issue with the Experimenter's Performance Measure charts again not displaying correctly when bad data was passed in.
- Fixed issues with module dependencies not being asserted correctly.
- Fixed some picklists to use Objects instead of treenodes where applicable.
- Fixed the Precision field for Bar Charts.
- Fixed an issue with flowitems without windowtitle attributes.
- Fixed a bug in the Set Object Color pick option.
- Fixed a bug in the Travel to a Home Location pick option.
- Fixed event definitions for the Rack (Place In Bay/Level and Min Dwell Time).
- Included fixes listed in 16.0.8 below.

Process Flow

- Fixed the State property for the Delay activity.
- Fixed the code headers for all the task sequence activity properties.
- Fixed the Sub Flow Instance Creation option from Quick Properties.
- Made the add button for attaching objects to instanced ProcessFlows more lenient.
- Fixed edit fields to properly display the code for labels with spaces in the name.
- Updated edit fields to more properly handle typing in token labels as opposed to code.

FlexSim 17.0.0 (December 12, 2016)

- Improved FlexScript performance by compiling and executing it as machine code instead of interpreted bytecode.
- Added new syntax to FlexScript for accessing objects' methods and properties, including dynamic label access.
- Improved OpenGL compatibility by removing many deprecated function calls and adding support for the OpenGL Core Profile.
- Added a mechanism for loading DWG data into the tree using the Model Background object.

- Added a new Snap to Background setting on the 3D view for snapping to points loaded from a DWG file.
- Upgraded the licensing system to Flexnet Publisher 2016 R1 (11.14).
- Changed the network licensing system so that you can open multiple FlexSim instances using only one license seat.
- Added a Quick Library popup to Dashboard views.
- Added Travel to Location pick option to OnResourceAvailable.
- Added Export Results to CSV pick option to the End of Experiment trigger.
- Moved the Rank buttons in the General Properties panel of Quick Properties to the Node Properties panel.

- Fixed a crashing bug with settablesize() on tables with bundle data.
- Fixed some issues with auto-complete not displaying correctly.

Process Flow

- Added functionality for snapping activities into the middle of a block.
- Improved the Activities window, including renaming activities, better filtering options, and locating label references.
- Added a Billboard setting for Text objects.
- Added a right-click Edit Activity Visuals option.
- Added an option to the Create Object activity for positioning an object at another object's location without moving into that object.
- Added a Preserve Global Position checkbox to the Move Object activity.
- Updated the Schedule Source table so that it can add labels.
- Updated the Release Token activity to allow numbers and strings.
- Added the ability to pass an object reference into the Resource(s) to Release property of the Release activity.

Backwards Compatibility Note: the following changes may slightly change the way updated models behave.

- The new version of Flexnet Publisher requires network license servers to be upgraded. They need to use the latest vendor daemon and update the Flexnet Licensing Service. More information can be found in the license upgrade instructions at https://www.flexsim.com/ftp/LicenseServer/
- Updated the Separator's order of events (executing OnEntry before Setup Time) to be consistent with the Processor.
- Users of the mesh api should update usage of GL_QUADS to use GL_TRIANGLES instead. GL_QUADS is deprecated and will not work when using the OpenGL Core Profile.
- Since the new FlexScript parser compiles to machine code, the order in which parameters are evaluated has changed to be aligned with the x86/x64 calling convention. This means models containing code where parameter evaluation order is important may have changed results. For example, the following code will have different results:

  ```
  myusercommand(duniform(1, 5), duniform(1, 10))
  ```

  In the x86/x64 calling convention, parameters are evaluated from last to first. In this case, the duniform(1, 10) call will be called first. Since this call changes the state of random stream 0, changing the order of parameter evaluation changes the result.

- The new parser has a stricter grammar for the == and != comparison operators. The types of the operands must be the same or related. For example, the following code will now give compile errors because it is comparing unrelated types:

  ```
  double x = 0;
  treenode y = model(); if
  (x == y) { }
  ```

- This version introduces a new Array type, which is an array of variants, enabling a more feature rich array usage. In doing this, we are deprecating the old array types of doublearray, intarray, stringarray, and treenodearray. Specifically, the old array types are now just aliases for the standard Array type. This means that you can now, technically, put a string into a doublearray and vice versa, because they are all just Arrays. Hence we encourage you to just use Array in your code instead of the old array types.

  This change has also introduced a problem regarding the Variant type. In previous versions, the Variant could hold each of the four types of arrays, and it had a type value associated with each type, which you could get with the getvartype() command, comparing that value to one of VAR_TYPE_INTARRAY, VAR_TYPE_DOUBLEARRAY, VAR_TYPE_STRINGARRAY, or VAR_TYPE_TREENODEARRAY. Now, however, since we've merged all of those types into one, all of those values would theoretically be the same value, introducing issues if you had code that switches on getvartype(), or in some cases if you had a series of if/else compares on that value. Depending on the specific nature of that code, it would be hard to predict exactly how that code would behave going forward. Thus, we have decided to get rid of those old macros for each array type. Now there is just the macro VAR_TYPE_ARRAY.

  If you have existing code that uses the older macros, you will get compile errors when you open your model in version 17.0. We do this specifically so that you will be notified of code that needs to be updated. There are also several pick list options in the process flow module that use these older macros. Version 17 includes update scripts that will hopefully update all of those pick options in existing models to use the new VAR_TYPE_ARRAY macro instead of the old macros.

- With the new FlexScript parser, there are now some differences with how the parser compares a null variant to 0. In the old parser, the following expressions applied:

  ```
  (nullvar == 0) is false
  (nullvar <= 0) is true
  (nullvar >= 0) is true
  ```

  In the new parser, the following expressions apply:

  ```
  (nullvar == 0) is false
  (nullvar <= 0) is false
  (nullvar >= 0) is false
  ```

  This brings the <= and >= operators inline with the == operator. However, old code will evaluate differently now. The following expressions apply in both the new and old parsers:

  ```
  (nullvar < 1) is true
  (nullvar > -1) is true
  ```

- The param() command will now return nullvar if the parameter number is greater than the number of parameters passed to the function (previously it returned 0).

FlexSim 16.2.2 (November 8, 2016)

- Included fixes listed in 16.0.7 below.

Process Flow

- Fixed a bug where all of a process flow's connections were sometimes being deleted.
- Fixed the TE Delay activity so it properly shows the delay progress in the view. • Changed ProcessFlow charts to always store the tracked variable's full history.

AGV

- Made multiple AGV process flows share the same AGV work list.
- Made the AGV path finder more adaptive on 90-degree turns.
- Fixed a bug where sometimes AGVs would flip orientation incorrectly at certain transfer points.

FlexSim 16.2.1 (October 12, 2016)

- Fixed a bug with the Font dialog not opening in Global Preferences.
- Fixed an exception with using JOIN ON without a WHERE clause in SQL queries.
- Fixed a bug with the number of child processes spawned by the Optimizer.
- Fixed the Presentation Builder auto-keyframe button.
- Fixed a bug with the Model Limit view staying open in loaded models.
- Fixed a bug with undoing in the Global Table Quick Properties panel.
- Fixed a bug in the date-based Time Table UI.
- Fixed undo in the 3D View's Quick Library popup.
- Fixed a bug with priority fields not accepting negative numbers.
- Included fixes listed in 16.0.6 below.

Process Flow

- Fixed a bug with not being able to click in the ProcessFlow view while debugging.
- Fixed bugs with selecting tokens in stacked blocks under a Batch activity.
- Fixed an issue with the text edit view not sizing correctly at certain zoom levels.
- Fixed an issue with Send To Front and Send To Back not working with containers.
- Fixed a bug with preemption in the Synchronize activity.
- Fixed a bug with Zone partitions sometimes not working properly when using the query() command.
- Fixed template instance objects resizing based upon the model units.
- Improved draw speed when drawing many tokens within a single activity.
- Fixed the Token(s) field on the Release Token activity to allow code.
- Fixed a bug with arrows not correctly following the theme color.

Conveyor

- Fixed a bug with the Conveyor Station drawing its red stopped squares incorrectly.

AGV

- Fixed some bugs with AGV orientation.
- Fixed a bug with AGVs sometimes stopping erroneously at a control point and not starting again.
- Fixed some issues with continuation from non-zero end speed when deceleration would take you across a control point.
- Fixed some bugs with the AGV template process flow.
- Fixed an issue with clicking control points.
- Fixed an issue with the navigator not rebuilding its routing information after deleting a path.
- Fixed a bug with copying paths in visual tools.

FlexSim 16.2.0 (September 5, 2016)

- Added support for STEP and IGES 3D shapes.
- Added a Quick Library popup when you double-click on empty space or A/S Connect to empty space in the 3D view.
- Improved object rotation manipulation in the 3D view.
- Added sorting to toolbox for top level items.
- Added the ability to use the query() command with partitioned lists, using ListName.$1 or ListName.PartitionIDStr syntax.
- Added support for the ON clause in SQL queries using the query() command.
- Added support for the ARRAY_AGG() aggregation function in SQL queries using the query() command, in the manner of PostgreSQL.
- Added an "Allow Multiple Pushes Before Back Order Fulfillment" setting to lists.
- Added a "Reevaluate All Values On Push" setting to lists.
- Added a Date Based feature to gantt charts.
- Updated the auto-numbering mechanism when naming new objects.
- Updated file browse dialogs to remember the last used directory.
- Changed the default directory for opening and saving user libraries to be within documentsdir().
- Added a colors tab to the tracked variable dashboard chart.
- Updated the Presentation Builder interface to make the timeline more useable.
- Updated the Animation Creator interface.
- Updated the Stop Time interface.
- Added range and type validation to many edit fields.
- Overhauled the FlexSim Web Server interface.
- Added functionality to the Webserver to restrict access to certain models using Windows Authentication.

- Fixed a bug with the window docking system that caused sizing to not work properly sometimes.
- Fixed TimeTable Excel Import bug not working with Daily/Weekly Repeat table.
- Fixed the Visual Studio project files so they don't give you the linker errors when you compile flexsimcontent.dll with Visual Studio.
- Fixed a bug with cell heights and widths when inserting and deleting table rows or columns.
- Fixed line charts continuing to collect data beyond their collect time.

- Fixed the query() command so that it can be called within List fields.
- Included fixes listed in 16.0.5 below.

Process Flow

- Improved Process Flow view performance.
- Added Split, Join, and Synchronize activities.

- Added right-click options for re-ranking activities in a block and removing an activity from a block.
- Added tooltips to activities that show you the activity type.
- Added copy and paste options to the Process Flow right-click menu.
- Populated field names of lists for SQL drop-down menu options.
- Updated Text editing interface to be more user friendly.
- Added an "Assign Event Object To" option to the Wait For Event and Event-Triggered Source.

- Fixed a bug with Zone Partitions not being able to be graphed.
- Fixed an issue with connectors losing their ranks when clicking and dragging them.
- Fixed the List Back Order Reevaluation Events so you can listen to activity events.
- Fixed the Experimenter duplicating TEs/FRs attached to Process Flow objects not properly duplicating the Process Flow instance.
- Fixed Sub Flows that have Resources pointing to 3D objects in the model not properly creating/destroying copies of the 3D object.
- Fixed Resources in Fixed Resource and Task Executer Process Flows losing their references when being updated by a user library.
- Fixed a bug with the Pull from List where releasing the token and leaving the back order on the list would not give you all of the pulled results when using the 'Assign To' option.
- Fixed a bug with capturing ProcessFlow views at resolutions larger than the window size.

AGV

- Added new agvinfo() command options.
- Updated paths so that they can be contained within visual tools.

Backwards Compatibility Note: the following changes may slightly change the way updated models behave.

- Fixed issues with routing through path junctions where 3 or more paths connect.

Conveyor

- Fixed an issue with manually moving a waiting item off a conveyor with a restart delay.

FlexSim 16.1.2 (August 5, 2016)

Process Flow

- Fixed the Acquire Resource receiving a reference to the back order when acquiring using a query.

FlexSim 16.1.1 (August 2, 2016)

- Included fixes listed in 16.0.3 below.
- Fixed color and sizing issues with recording Dashboards using the Video Recorder.

-
- Added a volume scale option to the FluidConveyor to adjust the visual display.

Backwards Compatibility Note: the following changes may slightly change the way updated models behave.

- Fixed several issues with the FluidConveyor.

Conveyor

- Fixed an issue with manually moving a waiting item off a conveyor with a restart delay.

Process Flow

- Fixed a bug with Zone Partitions not being able to be graphed.
- Fixed an issue with connectors losing their ranks when clicking and dragging them.
- Fixed the List Back Order Reevaluation Events so you can listen to activity events.
- Fixed the Experimenter duplicating TEs/FRs attached to Process Flow objects not properly duplicating the Process Flow instance.
- Fixed Sub Flows that have Resources pointing to 3D objects in the model not properly creating/destroying copies of the 3D object.
- Fixed a bug with capturing ProcessFlow views at resolutions larger than the window size.

FlexSim 16.1.0 (June 22, 2016)

- Added group commands (groupaddmember(), groupmember(), groupnummembers(), groupremovemember()).
- Global Lists will now update their Initial Content on reset when connected to a Group if the group's members change.
- Added support for the UPDATE clause in SQL.
- Added support for the RAND expression in SQL (uses stream 0 and will always be the same for a given query/row selection combo).
- Improvements to speed and memory usage when performing SQL inner joins.
- Tracked variables with type Categorical can now store arbitrary states.
- Added a Kinetic Level tracked variable type (e.g. Battery Level).
- You can now hold down the Alt key while clicking and dragging in the 3D view to ignore all objects.
- You can now resize objects while maintain their aspect ratio by pressing both the left and right mouse buttons down on a sizer arrow.
- Updated dashboard charts to support x-axis scaling.
- Added OnStop and OnResume events to 3D objects for use with event listening objects (Wait for Event and Event-Triggered Source in Process Flow).
- Video Recorder can now record Dashboards.
- Time Tables, MTBF/MTTR and User Events can now be disabled. Experiment variables can be set to enable/disable these objects.

- Fixed an issue with the VideoRecorder not loading properly.
- Fixed issues with windows and popups not opening on the correct monitor when using a multiple monitor setup.
-

- Fixed Fluid Conveyor throwing exceptions on reset.

Backwards Compatibility Note: the following changes may slightly change the way updated models behave.

Updated to OpenGL 3.1. Deprecated OpenGL functions, such as glBegin(), glEnd(), glVertex(), glNormal(), and glTexCoord(), may no longer work correctly on some graphics cards. You should instead use the mesh api.

- Changed the way that fixed resources (except the Combiner) receive items when they are being transported in. Previously, when they were notified that an item was being transported in, they would close their inputs and create an event to receive the next item. This would cause problems if the upstream objects had multiple items to send because their routing strategy would work differently when transporting vs not transporting. We have fixed this so that routing strategies will work the same when transporting as when not transporting. This may change the way that old models work because it changes the events that are created, and, obviously, it fixes routing strategy logic.
- menumain(), menubelow() and getviewmenu() have changed. Instead of returning a double, they now return a var. Any calls to these commands will need to be updated to use var or the value returned will be 0.

  double myMenu = getviewmenu(); Changes
  to:
  var myMenu = getviewmenu();
- Previously the Fluid Conveyor would stop conveying if no material was moved into it (ie inputs were closed). This was changed so that the Fluid Conveyor continues to convey the material in it whether any additional material is added or not. Set the conveyor speed to 0 in order to keep the same functionality that was in previous versions.

Conveyor

- Added Ports to the Entry and Exit Transfers.
- Added a Restart Delay setting to Conveyor Types.

- Fixed the Join Conveyors Tool when connecting conveyors with a custom conveyor type.
- Various bug fixes and improvements specifically with regards to items moving between conveyors and transfers.

Backwards Compatibility Note: the following changes may slightly change the way updated models behave.

- Changed the way stopping a non-accumulating conveyor works when propogating stops to straddled conveyors. When a conveyor is stopped, it will act the same as if an item on the conveyor were stopped.

Process Flow

- Added Preemption activities (Save Token Context, Release Token and Restore Token Context).
- Added gettokens() and getbatch() commands.
- Added Templates for Fixed Resource and Task Executer Process Flows.
- Added a Token Data to Preserve option to the Sink and Finish activities. In the Tokens window you can view Dead tokens and explore their data.
- Token Trace History is now stored as a label on the token.
- Added a Speed Type and Repeat Type to the Run Animation activity.
- A lot of improvements to the Zone.

-
- Updated the evaluation of the Return Value from the Finish activity so that executesubflow() can get a return value from multiple Finish activities.
- Added Activity Profiles to the token trace histories.

Fixed a bug causing the Wait for Event to evaluate its Max Wait Timer even if the token left the activity.

-

- Fixed issues with the releasetoken(), createtoken(), and releasebatch() commands working differently when running vs stepping.

Backwards Compatibility Note: the following changes may slightly change the way updated models behave.

- Changed the way that a token manually released (preempted) from waiting in a task sequence activity manages the task that it's waiting on. Now the task will be removed, and the task executer preempted if currently working on it. This may change older models that release tokens in task sequence activities.

AGV

- Added Event info to AGVs and Control Points for use with event listening objects (Wait for Event and Event-Triggered Source in Process Flow).
- Added AGV templates to Process Flow.

- Fixed bug with way points not firing their OnArrival for redirected AGVs.
- Fixed issue with arrival waypoints not firing correctly with non-zero end speed travel tasks.

Backwards Compatibility Note: the following changes may slightly change the way updated models behave.

- Fixed the order by which AGV trailers are attached to their AGV. This will affect old model trailer ordering.

FlexSim 16.0.9

- Fixed issues with table commands not working in C++ when the table has bundle data.
- Fixed more bugs in time-scaled animations in various cases.

FlexSim 16.0.8 (January 18, 2017)

- Fixed an issue with pushing an array to a list with static label fields.
- Fixed a crash when a bundle used by a chart is destroyed.
- Fixed a bug in time-scaled animations that was causing them to not update properly in some cases.
- Fixed images in static text not showing when aligned to center.
- Fixed a bug with the By Time of Day picklist on Send To Port.
- Fixed several By Percentage picklists so that they work properly with 0%.
- Fixed Global Variables being linked to Dashboard Radio Buttons.
- Fixed an issue with selecting weeks in the Date Based Time Table.
- Fixed module object context help options.
- Fixed an issue with the Experimenter updating state values when a run is finished.
- Fixed the event log not ignoring module events when you disabled them.
- Fixed copying and pasting presentation slides.

-
-

Conveyor

Fixed some issues with floating point precision errors.
Fixed photo eyes not clearing when height > 0.
Fixed an issue with an entry transfer's pull strategy pulling from a list.

- Fixed a bug in the Exit Area pickoption.

Process Flow

- Fixed an issue where some charts didn't collect data if they weren't visible.
- Fixed the Zone so you can view its tokens in the global process flow view of instanced process flows.
- Updated some charts so they show the correct times when using a warmup time.
- Fixed a bug with the batch when releasing multiple batches due to overflow when the batch quantities differed from the current batch and the new batches.
- Fixed an issue with viewtofile() on the Process Flow view.

AGV

- Fixed an issue with adding and removing different types of control point connections.

AStar

- Fixed the distancetotravel() command for AStar.
- Fixed issues with creating and drawing barriers in models set to units other than meters.

FlexSim 16.0.7 (November 8, 2016)

- Fixed a bug with clearglobaltable() not always clearing the entire table.
- Fixed bugs with addtablerow(), addtablecol(), movetablerow(), and movetablecol() on bundle tables.
- Fixed a UI bug with large Partition IDs displaying incorrectly.
- Fixed a UI bug with the user command code editor's default header.
- Fixed a bug with animation durations when deleting animation clips.

Conveyor

- Fixed conveyorsenditem() so that it works with Station objects.

Process Flow

- Fixed the Create Object activity so creating something on an AGV control point sets the location to the control point.

-

-
-
- Fixed referencing process flow variables so it throws an exception if the variable doesn't exist.
- Fixed the Run Animation Wait for Complete check box to work with bone animations.

FlexSim 16.0.6 (October 12, 2016)

- Fixed an exception when calling updatelocations() on TaskExecuterFlowitems directly under the model node.
  Fixed a bug with multisorttable() not working correctly with string data.
  Fixed a bug with querying lists using the $ parameter syntax.
  Fixed a bug with state charts not properly accounting for the minimum collect time with the new stats structure.
- Fixed an intermittent exception in gettenetnode() in the Travel to Home Location pick option.
- Fixed an intermittent exception when pasting network nodes.
- Fixed a bug with eventget(num, 4) throwing exceptions.
- Fixed a bug with gettablecols(string) returning inconsistent values.
- Fixed issues with the Optimizer not exporting string variables correctly.
- Fixed a bug with the Optimizer not correctly running multiple replications with certain configurations.
- Fixed a bug with Copy Visuals and Shape Factors not copying the shape factors correctly.
- Fixed a UI bug in the MultiProcessor's Properties window that was renaming the first step.
- Fixed the State Pie Chart so it correctly shows Totals and Averages of combined groups.
- Fixed a bug in the State Pie Chart with "Combine All into One Pie" not showing the correct average totals.
- Fixed some UI issues with sampling global tables.
- Fixed the On Process Finish trigger for the MultiProcessor so Process Flow activities can listen to it.

Conveyor

- Fixed a bug in the Entry Transfer's Pull Requirement code header that caused pull not to work correctly.
- Fixed a UI bug with the Station's Use Operator fields.
- Fixed the z-height of conveyors created in a container using click-click creation.

Process Flow

- Greatly improved model run performance by removing some unnecessary destroyeventsofobject() function calls in the Sink activity.
- Fixed issues with being able to wait for events on Tracked Variables and Fields.
- Fixed the sampler in pick options in the Create Object activity properties.
- Fixed a UI bug with the Visually Trace checkbox.
- Fixed the token Labels view so you can select and copy label names.

-

-
-

AStar

- TaskExecuters that are travelling on the AStar Network will now properly respond to stopobject() and resumeobject() calls and to preempting task sequences.

FlexSim 16.0.5 (September 5, 2016)

- Fixed sin() command not displaying in the Command Helper.
- Fixed a bug when shift-selecting entire rows and columns of global tables.
- Fixed C++ Variant comparisons where the primitive is on the left side.
- Fixed the return values for CURRENT_MINUTE, CURRENT_SECOND, and CURRENT_MILLISECOND in getmodelunit().
- Fixed a bug with SQL parsing of FlexScript functions returning doubles in 32-bit FlexSim.
- Fixed some pin buttons in the Stats window.
- Fixed a bug in a Pull From List pick option.
  Fixed Crane and Robot speeds not scaling with model units.
  Fixed a bug with the From/To Lookup Table popup not putting quotes around table names. Fixed a bug with exporting Custom Chart data as CSV sometimes not working.
- Fixed some UI issues with the Edit Name box for User Libraries.
- Fixed an exception in the Set Label trigger popup.

Conveyor

- Fixed header for OnMessage trigger of Photo Eyes.
- Fixed decision point rotation in a rotated container.
- Fixed a bug with calculating the time that items will split off from each other when an ahead item speeds up.
- Fixed a rare crashing issue with accumulated items.
- Fixed an issue with resuming nonaccumulating conveyors that weren't stopped.

Process Flow

- Fixed a UI bug with the process flow variable panel in Quick Properties when opening a model.
- Fixed some instance references from showing just the name to now showing the path to better distinguish objects in containers.
- Fixed an extra line being added in the token Shared Assets list when requesting a Zone.
- Fixed the Quick Library so you can collapse sections.

AGV

- Fixed issue with click-creating AGV control points before creating paths.
- Fixed issue with calling stopobject() on an agv when it has arrived at a control point at (near) zero speed.

-

-
-

FlexSim 16.0.4 (August 5, 2016)

Process Flow

- Fixed the Acquire Resource receiving a reference to the back order when acquiring using a query.

FlexSim 16.0.3 (August 2, 2016)

- Fixed a performance issue with settablenum() when referencing a table by rank.
- Fixed a bug in exporttable() not exporting the correct number of rows.
- Fixed a bug in query() with $iter(1) in certain cases when inner-joining tables.
- Fixed a rare crashing issue with the WRL importer.
- Fixed some glitches with editing Chinese characters in code edit fields.
- Fixed an issue with Chinese characters in some Dashboard charts.
- Added a flag to listpull() to fix an issue involving backorders when using Unique Pullers or Unique Values.
- Fixed a bug with custom state names in the State Chart.
- Fixed an infinite loop caused by using long object names with some Dashboard charts.
  Updated some picklist options to use more consistent default terms.
  Fixed an issue with Code Snippet popups not resizing vertically. Fixed
  a bug with adding all user commands to a user library.
- Fixed a bug with some startup and load install options not working properly with user libraries.
- Fixed a UI bug with the Excel progress bar.
- Fixed a UI bug with certain spin boxes not updating their text when holding the mouse on their arrow buttons.
- Fixed a bug with Excel Import starting row and starting column not always working correctly.
- Fixed an HTML Frame Load Error for user accounts with Unicode characters by changing programdatadir() to again use ProgramData instead of AppData.
- Fixed the convert() command to handle years after 2035.
- Fixed a bug with the experimenter displaying blank results due to invalid data.
- Fixed a bug with the experimenter not properly recording dashboard state charts with collect times defined.
- Fixed an issue with pushing an array of values to a list when there are back orders waiting.

Process Flow

- Fixed a bug with Pull from List not setting the pulled label when nothing was pulled.
- Added conveyor pick options to the Custom Code activity.
- Fixed a timing issue with the Resource at time 0 that sometimes allowed too many tokens to allocate it.
- Fixed displaying Chinese characters in Text objects.
- Fixed a bug with Zones throwing SQL errors when also using the query() command.
- Fixed an infinite loop in the auto-naming of Split connectors.

-

-
-
- Fixed an issue with certain time-weighted charts not calculating the average correctly.

AStar

- Fixed a repeatability issue with path finding in large models.

FlexSim 16.0.2 (June 28, 2016)

- Fixed a bug with nodetopath() sometimes returning a path that starts with / when it should start with >.
- Fixed a bug with exporting tables with bundle data as csv.
- Fixed an issue with reading current bound-statistic values.
- Fixed a bug with weight objectives in OptQuest.
- Fixed a bug showprogressbar() not updating the text properly.
- Fixed an issue with watch variables not working right with some arrays.
- Fixed an issue with popups closing when you hover over a tab when using the sampler.
- Fixed an exception with drawing connections and hidden objects in the 3D view.
- Fixed a bug with 'station' not being correctly passed into the Load Time and Unload Time triggers on Task Executers.
- Changed the number precision model setting so that it behaves in a more consistent manner.
- Fixed a UI bug with the Display Current State Only checkbox.
- Fixed Help Manual's Open in Browser button.
- Fixed a bug with the Code Profiler closing when undocked from a tab pane.
- Fixed a bug with opening the Properties window for some shapes in the Animator.
- Fixed an issue with using multiple replications with OptQuest.
- Fixed a UI bug in the Optimizer view.
- Fixed an issue with operators getting stuck when preempted while finishing an unload task.
  Fixed a bug with the Combiner components list sometimes not updating.

-

-
-

  Fixed the Time Table Dated Based GUI so it works with all model time units.
  Fixed some exceptions being thrown with dashboard state charts when setting an object to a state that is not listed on the state chart.

Conveyor

- Fixed a UI bug with applying Decision Point and Photoeye triggers.
- Fixed a bug with an operator picking up an item when it has transfered partially to the next conveyor.
- Fixed an issue with an item exiting firing the accum stopped cleared event of upstream items.
- Fixed some errors with clearing photo eyes on item exit.
- Fixed a bug where items behind a side transfer were not notified of a speed driver change at a side transfer.
- Fixed an issue that caused exceptions when items are removed while straddling multiple conveyors.
- Fixed a binding issue with the OnSlugStart event.
- Fixed a bug where re-routing was not working on a conveyor with a virtual length.

Process Flow

- Fixed some exceptions in Push to List.
- Fixed a Push to List bug where puller is not assigned to a label if the pushed value is not a token.
- Fixed the Tokens popup throwing exceptions when the token you're viewing leaves.
- Fixed a bug causing the Wait for Event to evaluate its Max Wait Timer even if the token left the activity.
- Fixed some issues in the releasetoken(), createtoken(), and releasebatch() commands so that they work the same when running or stepping.
- Changed Request and Require fields to not require integers.
- Fixed some bar chart color bugs.
- Fixed a precision issue in the Batch activity.
- Fixed a UI bug with ProcessFlow chart Display Names.

AGV

- Fixed a bug with way points not firing their OnArrival for redirected AGVs.
- Fixed a bug with queryDistance().
- Fixed an issue with arrival waypoints not firing correctly with non-zero end speed travel tasks.

FlexSim 16.0.1 (April 1, 2016)

- Fixed a performance bug with stats_staytime() that was causing models to run slowly.
- Fixed a bug with Excel export not working for certain columns.
- Fixed some memory leaks in drawcylinder(), drawcolumn(), drawdisk(), and drawsphere().
- Fixed an issue with dll nodes sometimes not binding properly.
- Fixed bugs with excelcreatesheet() and excelsaveas().
- Fixed a bug with old-structure stateprofiles not resetting properly.

-
- Fixed a bug with group objects sometimes being added to charts twice.
- Fixed a bug with 3D stats not displaying correctly on some objects.
- Fixed dashboard buttons not saving their text color.
- Fixed a bug where the hover text for combined stats on Stat charts said average for total values.

Conveyor

- Fixed a bug with decision points' current content stats.
- Fixed a bug where items were sometimes not routed to the correct location.
- Fixed a bug where exit transfers sometimes sent items to the wrong port when multiple items were released.

Process Flow

- Fixed some statistics bugs in the Zone.
- Fixed some crashing issues when "Assign To" is given invalid data.
- Fixed a bug with Batch Label Aggregation to keep strings/arrays/etc. with the Last Value and First Value options.
- Fixed issues with the Run Animation not allowing you to pass in an array of objects.
- Fixed an issue with using the wrong instance when pushing/pulling from a List in another Process Flow.
- Improved 3D mouse performance in the ProcessFlow view.

AGV

- Made the addMember method a dll function so Process Flow can connect travelers to any control point using the Create Object activity.
- You can now correctly get an AGV's current control point when it finishes OnPreArrival.
- Fixed an issue with AGV start speeds incorrectly being set to 0 in certain circumstances.

FlexSim 16.0.0 (March 14, 2016)

- Added options to Global Preferences to define which libraries are visible in the Drag-Drop Library and what order they appear in.
- Restored the Media Files window (Main Menu > View > Media Files).
- Added the ability to assign unique random streams to objects in the model using the command getstream().
- Added a Network Navigator Properties window. (Right-click a NetworkNode to access.)
- Added a Track Number Field Totals option to the List for stats collection.
- Added a 'Values Only' data distinction option to the Excel Importer. This data distinction reads number data and string data from excel cells without regard to the cell's formatting (i.e. dates and times are imported as Excel time values).
- Implemented the excelrangeread() command to import a range of cells using the 'Values Only' data distinction.
- Added the command convert() for converting model/Excel dates and times.
- You can now use keyboard shortcuts for debugging. Next Line -> F10, Step Into -> F11, Continue -> F5, Stop Simulation Clock -> Shift+F5

- Updated object statistics to use Tracked Variables for collecting Content, Input, Output, and Staytime statistics. NOTE: These new statistics are only available for new objects. See the Object Statistics backwards compatbility note below for more information.
- Made the sql IN clause work if you compare it against a value that is an array.
- Added trackedvariable() and inittrackedvariable() commands. This allows tracked variables to be created on objects as labels, etc.
- Tracked Variables can now be accessed using the commands get(), set(), inc(), setnodenum(), getnodenum(), getlabel(), setlabel().
  Tracked Variables now have a Type which can be Level, Cumulative, Time Series, or Categorical (see inittrackedvariable() documentation).
- Tracked Variables can now store profiles (similar to state profiles, the profile stores data on the amount of time the tracked variable was at each given value).
- Improved OpenGL compatibility with older graphics cards and Nvidia Optimus cards.
- Fixed an issue with copy and paste in the 3D view not maintaining the z position of the copied object.
- Fixed the Load From File button in the Edit Selected Objects window to call the firecreateevent on loaded objects.
- Fixed an issue with the dashboard Send To Back not always working.
- Fixed an issue with the Experimenter Default Reset Scenario combobox.
- Fixed dashboard snapping.
- Fixed issues with pasting a node onto another node of a different data type.
- Fixed an issue with FlexSim sometimes crashing when a dashboard widget was deleted.
- Fixed a memory leak in the query() command.

Backwards Compatibility Note: the following changes may slightly change the way updated models behave.

- Updated the Combiner so that items can properly exit through port 0 (such as using a Process Flow Load activity).
- Fixed a bug in the Separator state management that was causing it to be idle when it should be blocked.
- Changed the max bones per mesh for bone-animated shapes from 60 to 42.
- Object Statistics: We have changed the default statistics structure for objects in the library. This will not change the structure of existing models, but will now be different for new objects that are created. Specifically, we have made the standard object statistics--namely input, output, content, staytime, and state--use tracked variables instead of the many different object attributes they used previously. This enables us to add new capabilities, such as a content-vs-time graph that exactly reflects the object content changes instead of an interval-based time graph (use the Tracked Variable vs Time chart or the pin button in the Quick Properties Statistics panel), and the ability for Process Flow Wait For Event activites to listen to object content, input, output changes, etc.

  Current FlexSim features will work for both the old and new statistics structures. If you have model or module code that is accessing statistical data directly (such as using commands like get(stats_contentavg(object)) or get(rank(state_profile(object), STATE_IDLE))), this code will continue to work on objects that have the older structure, but it will not work for new objects added to the model. If you want this access to work with objects having the new structure, you should use the getstat() command to get the desired data. The getstat() command will work on both the old structure and the new structure. Below is a list of examples of old statistical access commands vs new getstat() based access.

  OLD: get(stats_contentavg(object))
  NEW: getstat(object, "Content", STAT_AVERAGE)

- (Note here that the getstat() command will return the weighted average that includes the time that the object has been at its current content, whereas with the old method, you had to add that on manually).

  OLD: get(stats_staytimemax(object))
  NEW: getstat(object, "Staytime", STAT_MAX)

  OLD: get(rank(state_profile(object), STATE_IDLE))
  NEW: getstat(object, "State", STAT_TIME_AT_VALUE, 0, STATE_IDLE)
  (Note here that the getstat() command will actually give you the total time in the idle state. This includes the time in its current state if the object is currently idle. With the old method you always had to manually add that in, i.e. get(rank(...)) + (getstatenum(object) == STATE_IDLE ? time() - get(state_since(object)) : 0);

  OLD: getstatenum(object)
  NEW: getstatenum(object)
  The getstatenum() command works on both structures, so you don't need to change it. Alternatively, you can use getstat(object, "State", STAT_CURRENT, 0) (0 is the state profile)

  OLD: getinput(object)
  NEW: getinput(object)
  The getinput() and getoutput() commands work on both structures, so you don't need to change them. Alternatively, you can use getstat(object, "Input", STAT_CURRENT)

Conveyor

- Added Pull and Send To functionality to conveyors. This is done by connecting multiple objects to a single entry/exit transfer.
- Added reset position functionality to conveyors.
- Added additional events to conveyor objects that can be used in connection with the Wait for Event and Event-Triggered Source activites in Process Flow.
- You can now explicitly remove connections from an entry/exit transfer without it being deleted (using the right-click menu). This is useful when sending flowitems to a list where connections are not required.
- Straight and curved conveyors now have an optional Virtual Length.
- Added Priority and Preempt fields to the Exit Transfer.
- Added a Station object. The Station works much like a Decision Point but also allows you to specify a processing time and optionally call operators.
- Added States to conveyor objects.
- Added a CONV_INFO_GET_ACTIVE_ITEM option to the conveyorinfo() command to get a decision point or photo eye's covering item.
- Fixed decision points and photo eyes not saving to user libraries when adding a conveyor.
- Fixed conveyor widths not updating when changing conveyor types.
- Fixed issues not being able to snap decision points to the beginning or end of conveyors.
- Fixed the conveyor type so new conveyors use the default conveyor type rather than Custom.
- Fixed the name numbering of decision points and photo eyes.
- Fixed the Release Area pick option in decision points.
- Various other bug fixes and improvements to conveyors.

Process Flow

- Added a label filter option for tokens in Quick Properties.
- Process Flow View now supports a 3D mouse.
- Updated activity snapping to allow activities to be snapped to the top of other activities.
- Updated the Release Resource activity to allow a reference to a Resource Shared Asset to be passed into the Resources to Release property.
- Blocks of activities and display objects can now be saved in User Libraries.
- Added aligning functionality to the Process Flow view.
- Added snap-to-grid functionality to the Process Flow view.
- Speed improvements to drawing.
- Assign To properties on the Create Object, Push To List, Pull From List and Create Task Sequence can now either overwrite the assigned label's data or insert the data at the front (creating an array).
  When tying the Resource shared asset to a Group, Dispatcher or Array, the Count property now allows you to utilize any number of those objects. This allows you or the experimenter to easily change the number of available resources without objects being added/removed from the model.
- Added a 'Do not remove entries from List' option to the Pull From List activity allowing you to utilize the Request and Require fields without removing entries.
- Tracked Variables can now be added as labels on objects. This allows you to track the history of its values as well as tie it to a Wait for Event activity as a value change listener.
- Fixed issues with the Finish/Sink activity not properly deallocating shared assets if they still had children in the model.
- Additional bug fixes and overall improvements.
- Fixed exceptions sometimes being thrown when dragging connectors around.
- Fixed the Priority field in the Create Task Sequence activity.
- Fixed some issues with listening to the OnStateChange of objects.
- Fixed an issue that caused large numbers of TE/FR instances to take a long time to reset.
- Fixed an issue with the Resource when using a Local type with each instance having a unique resource reference.

Backwards Compatibility Note: the following changes may slightly change the way updated models behave.

- The Assign Labels activity used to allow you to pass in a treenodearray or a Group so that it assigned labels to multiple objects at once. Now the Assign Labels only accepts one treenode at a time. Passing in a treenodearray will cause only the first entry in the array to have labels assigned to it. To update any old model utilizing this functionality, loop through each object in the array using the Object from Label Array pick option.

AGV

- Updates to AGV travel tasks and preempting.
- Some minor bug fixes.

Backwards Compatibility Note: the following changes may slightly change the way updated models behave.

- AGVs will now read the end speed of the travel task and use it to determine when to finish the task. This means that if you have an existing model where you defined a non-zero end speed of a travel task, older versions would have ignored this, but now the end speed will be simulated properly. Please see the user manual topic on AGV Types for more information on how the end speed is simulated.

-
- We changed the AGV preemption mechanism so that:
    1. When you preempt an AGV, if you do not immediately give it a new travel task, it will decelerate to stop after the preemption occurs. In previous versions the preemption mechanism stopped the AGV immediately.
    2. If you preempt an AGV and then immediately give it a new travel task, it will start the new task at the speed it was at before it was preempted. In older versions, a preemption would cause the AGV to immediately stop and then start from speed 0.

FlexSim 7.7.4 (January 28, 2016)

- Fixed an exception in the VideoRecorder.
- Fixed a bug with list back orders table sometimes not drawing.
- Fixed a bug with the BasicConveyor recycling itemtype nodes.
- Fixed a bug with coordinated task sequences.
- Fixed a bug with compiling using Visual Studio 2015.
- Fixed a bug with the Ticker component list table.
- Fixed an issue with getvarnum() returning treenodes when it shouldn't.
- Fixed an exception when debugging uninitialized treenode variables.
- Fixed a scaling issue with changing the operator's shape with different model units.
- Fixed a bug with some vrml textures not displaying correctly.
- Fixed bugs with certain configurations of drawcylinder(), drawcolumn(), drawcube(), and drawsphere().

Conveyor

- Fixed exceptions that were thrown when an item straddling multiple conveyors exits through an exit transfer and it is still taking the speed of the upstream (sending) conveyor.

Process Flow

- Fixed the Resource Queue Strategy.
- Fixed a bug where FlexSim would crash when opening some models.
- Fixed issues with using the Create Object activity to TaskExecuter Flowitems on Network Nodes that were inside of Visual Tools.

FlexSim 7.7.2 (January 8, 2016)

- Fixed Elevators, Cranes, and ASRS objects not resizing correctly for different model units.
- Fixed an issue with the List Entries debugging tool not allowing you to sample objects.
- Fixed the default header for the OnStateChange trigger.
- Fixed an issue with displaying variants in the local/watch variables windows of the debugger.
- Fixed a bug with drawtomodelscale() and drawtoobjectscale() commands.
- Improved the performance of XML loading and saving.
- Fixed the gettablestr() command when a Global Table is set to use a bundle.

- Fixed a bug with using global lists in the query() command.
- Fixed bugs that caused child experimenter processes to die unexpectedly.
- Fixed a bug where code windows would sometimes not show the text when multiple windows were open together.
- Fixed a bug with multisorttable() not working correctly with descending column sorts.
- Improvements to quick properties for faster updates and better scroll management.

- Added an Update Locations picklist to Enter/Exit triggers.

Backwards Compatibility Note: the following changes may slightly change the way updated models behave.

- Fixed drawtext()'s rotation parameters so that they match FlexSim object rotations.

AGV

- Fixed issues with agvaddlistener and cpaddlistener not interpreting object references correctly.

Conveyor

- Fixed a smooth transfer interpolation bug when an item is straddling three conveyors.
- Fixed an issue where an item merging onto a conveyor doesn't merge properly if it is blocked by an item that is diverting from the conveyor.
- Made various fixes for undo functionality.
- Fixed an issue with speed drivers not properly calculating accumulation block points in specific scenarios.

Process Flow

- Fixed an issue with trace histories not displaying in FR/TE Process Flows.
- Fixed an issue with viewing List back orders for instanced Process Flows.
- Fixed the Wait for Event and Event Triggered Source to properly pass in the eventRank.
- Fixed an issue with containers sometimes not moving activities inside of them.
- Fixed an issue with entries on a List not properly being removed when a token was released manually from the Push to List activity.
- Fixed issues with using Process Flow Variables in the Experimenter.
- Fixed issues with saving Process Flow dashboards and using Process Flow dashboards as PFMs in the Experimenter.
- Fixed an issue with connectors not being renamed properly.
- Tokens are now destroyed when sitting in Sink/Finish activities once all of their shared assets are deallocated.
- Improvements to the Universal Edit fields (you can now select a portion of text in the field and use the sampler or select a label from the drop down).
- Improved zooming in the Process Flow view.
- Improvements to quick properties for faster updates and better scroll management.
- You can now manually release a token from a Run Sub Flow activity using the releasetoken() command.
- Updated the Custom Task Sequence activity to use an edit field for the task type to allow for userdefined custom tasks.
- Update the Delay Task Activity to display the delay time on the token.
- Updated the Process Flow dll to allow user-defined modules to extend Process Flow with custom activities.
- Added OnEntering, OnExiting, OnManualRelease, OnAssetAllocated, and OnAssetDeallocated events to the token.
- Added additional Array picklists for the Assign Labels activity.

FlexSim 7.7 (November 23, 2015)

- Added a Process Flow module for handling complex logic using flowcharting.
- FlexScript now has a var type which can be any of the current variable types (number, string, treenode, intarray, doublearray, stringarray, treenodearray).
- Bone Animations - Animator can now import 3D models with bone animation data and set those to FlexSim animations.
- Added Global List objects to the Toolbox (see User Manual > Modeling Tools > Global List).
- Global Tables can now store their data in a bundle rather than in nodes.

- Arrays (int, double, string and treenode) can be stored on nodes such as labels.
- Arrays are now viewable/editable as tables.
- Labels can also store pointers to other objects using the setlabel() command.
- Deprecated parnode(), parval() and parstr() and replaced with param().
- Added an arraysize() command.
- Added support for Oculus Rift.
- Added support for using Xbox 360 Controllers.
- Added a global preference for displaying the 3D view's frames per second.
- Added a Truck TaskExecuter Flowitem.
- Added a Custom Gantt Chart dashboard.
- Dashboards now have the ability to scroll.
- Added a precision property to dashboard stat widgets.
- Added a spinner object to the dashboard model input objects.
- Updated dashboard charts to rebuild their data if pointing to a group and the number of group members changes during the model run.
- Dashboard charts can now be copied and pasted.
- Removed the AVI Maker and replaced it with the new Video Recorder.
- Added an "Available at Simulation Start" option to TaskExecuters that causes the OnResourceAvailable trigger to fire at time 0.
- Changed many Quick Properties panels to apply to all selected objects.
- Improved the Rotated Selected and Flip Selected so you can click in the 3D view and select a rotation axis.
- Added a Model Layout option to the Experimenter Variables.
- Improved the Experimenter web viewer export so view can be customized.
- Task Sequences now have the ability to store their own labels.
- Improvements to FlexSim's display for non-standard DPI settings.
- Added a panel to Quick Properties that allows for saving high resolution images of the 3D view (this can also be done through the viewtofile() command).
- Updated TrackedVariables to be a simpledatatype object instead of a node with a bundle.
- Typing shift+enter in a table cell creates a new line rather than moving to the next cell.
- Changed the UI for flowitem packing methods so it makes more sense.
- Added a flowitem packing method - Layer Stacking By Item Size
- Added On State Change triggers to the BasicTE and BasicFR trigger pages.
- Got rid of the usage of FRLOAD and FRUNLOAD. USe LOAD and UNLOAD instead.
- Updated the updatelocations command so that if you pass an item in as the object parameter, it calls updatelocations on the parent object of the item.
- A lot of general improvements to drawing performance.
- Improvements to how model data is saved to remove differences in the xml files.
- Added a page to the User Manual about model repeatability (FlexSim Concepts > Model Repeatability).
- Deprecated the getmodelunitnum() and getmodelunitstr() command and replaced it with getmodelunit().
- Deprecated Global Task Sequences. User Process Flow instead.
- Improved User Manual search results to display the most relevant results first.
- Improvements to the License Activation window.
- The middle mouse button can now be used to pan in the 3D view and scrolling with the mouse wheel now zooms relative to the mouse location.
- Added statisticaltime() command.
- Added rangemin and rangemax attributes to an Edit field to define a range of valid values that can be entered (options have also been added to the Dashboard).
- Added all FlexScript commands as sql scalar functions.

- Adding support for the HAVING clause in the query command.
- Added the == operator to the sql parser.
- Added an option to the Excel Importer to allow you to import cells formatted with Dates/Times as numbers when using the Automatic Data Distinction setting.
- Changed Copy to also put the path to the highlighted object onto the clipboard and removed Copy Path to Clipboard from the tree's right-click menu.
- Improvements to the Excel Importer's Automatic Data Distinction option. It now correctly determines if a cell has a number in it but is stored as text. In this case, instead of importing a numeric value of 0, it will import the text as a string.
- Fixed Time Plots in the dashboard so they clear their data after the warmup time is complete.
- Fixed a bug with the visual tool inner connections not surviving copy/paste correctly in certain cases.
- Fixed the visual tool so it can have multiple internal connections go through each other.
- Fixed the Elevator so the delivery platform properly lines up with wherever the item is being dropped off at.
- Fix for Task Executers so the endspeed works for offset travel.
- Fixed fly paths to run in real seconds when not running with the simulation time.

Backwards Compatibility Note: the following changes may slightly change the way updated models behave.

- Fixed resource item state variables are now stored with a custom data structure instead of being stored directly in the tree. This means that if you previously accessed an item's fixed resource state directly through the tree instead of using the commands getitemstate(), getitemsendto(), getitemvar(), etc. your model will need to be updated to use those commands.
- Previously, drawfont() would only change the font for one call of drawtext(). Now, calling drawfont() will change the font and leave it that way until it is changed again.
- Fixed the 50-flowitem receive limit on the combiner, processor, and rack. This will break old models if your model is dependent on this bug.
- Null Variants no longer equate with 0 (NULL). We have added the "nullvar" keyword that equates with null Variants. This could affect models that use getlabel() on a label that does not exist. For more information, see getlabel()'s documentation.
- The following commands now take/return Variants instead of doubles:
    - o nodefunction() o
    sendmessage() o
    function_s() o
    function_n() o
    userfunction_s() o
    userfunction_n() o
    nodefunctionlisten() o
    delayednodefunction() o
    addbundleentry() o
    applicationcommand()

    The implications for these changes are as follows:

    nodefunction(x, tonum(current)); // BAD CODE nodefunction(x,
    current); // GOOD CODE

    The problem here is that if you cast it into a number, and on the other side someone wants a node, the Variant thinks it's a number so it won't give back a valid node.

Additionally, all FlexScript and c++ code nodes now return a Variant instead of a double. This also has implications.

return tonum(centerobject(current, 1)); // BAD CODE return
centerobject(current, 1); // GOOD CODE

Here again, if you put something into a number, and then on the other side it wants a node, the variant realizes it's holding a number and won't give back a valid node.
We have implemented an update script that will search the model for this type of code and will update the code to not use tonum(). The update script logs any changes made, putting the log in the node MODEL:/Tools/CodeUpdateLog.

The update script will make the following changes:

11. return tonum(...); changes to:
    return (...);
12. For each of the above mentioned commands, the update will search the model for calls to this command, and will update the parameters as follows:
    nodefunction(theNode, tonum(...), tonum(...));
    changes to: nodefunction(theNode, (...), (...));

However, the update script will NOT catch the following:

double myVal = tonum(...); return
myVal;

or,
double myVal = tonum(...);
nodefunction(theNode, myVal);

In saying this, please realize that leaving number casts in these areas may continue to work properly, particularly if you the user control both sides of the pass-off, i.e. you have implemented both the caller side and the callee side. For example:

Caller side:
nodefunction(theNode, tonum(theObj));
Callee side:
treenode myVal = parnode(1);

This example should actually work properly. We have deprecated parnode(), parstr(), and parval() and replaced them with a single param() command that returns a type-safe Variant. However, the deprecated commands will essentially try their best to take whatever was passed in and put it into whatever you want. So parnode() will see a number, and try to get it into a treenode. This is the same for returns:

Callee side:
return tonum(myObj); Caller
side:

treenode myObj = tonode(nodefunction(theNode, ...));

Again, this should work, because tonode() and tonum() will essentially try to take whatever is there and put it into a number/node.
Where this will break is where one side is using the type-safe method while the other is not.

Caller side:
nodefunction(theNode, tonum(theObj));
Callee side:
treenode myVal = param(1); // WILL NOT WORK BECAUSE param(1) IS A NUMBER NOT A NODE

On FlexSim's side we have moved all (or most) of our code over to using the type-safe method. Thus any calls that interact between FlexSim code and your code should use the type-safe method. We hope that the proactive update script will resolve this for 99.99% of cases.

Conveyor

- Fixed a bug with exceptions throwing when an item is picked by an operator while straddling two conveyors.
- Fixed bug that was caused when you have skewed rollers and the item size is less than half the width of the conveyor and you go through a merging side transfer.
- Fixed a bug with side transfers when the item is already moving to an inline transfer.
- Fixes to rotate selected and flip selected for Conveyors.
- Bug fixes on drawing conveyors.
- Fixed entry/exit transfer connecting with network nodes.
- Added listenable events for Conveyors, Photo Eyes, and Decision Points, allowing Process Flow to listen for conveyor events.
- Fixed an issue with transfers being deleted when you click on a visual tool
- Fixed photo eye draw with offset angle.
- Added list push/pull pick options for Decision Points and Photo Eyes.

Process Flow

- Released a new Process Flow module. See Why Use Process Flow for more information.

FlexSim 7.5.4 (April 24, 2015)

- Fixed an issue with installing flowitems from user libraries.
- Fixed an issue with the undo history in the Global Variables window.
- Added a Delete Path option to the context menu of Network Node paths.
- Fixed the Match Whole Word option in the Tree Search panel.
- Fixed an issue with task sequences to allow for creating more than 65,535 tasks.
- Fixed an issue with visual tools as containers storing correct stats when a warmup time is used.

Backwards Compatibility Note: the following changes may slightly change the way updated models behave.

- Fixed a bug with "Do not travel offset and block space on networks" where task executers started at their endspeed even after a load/unload or after an extended period of time.
- Fixed a bug with TEs jumping to their max speed when leaving a network to do offset travel.

Conveyor

- Fixed a bug that was causing exceptions when you copy/paste conveyors from one FlexSim instance to another.
- Fixed a bug with sending to the correct exit transfer when there are multiple transfers at the end of a conveyor.
- Fixed an issue where a conveyor transfer's inline speed rule was not being followed when transferring from a previously blocked non-accumulating conveyor.
- Fixed a bug causing exceptions when you reset the model in certain power and free models.
- Fixed a bug where accumulation stop events weren't being created at the right times (bad calculation of gap close point).
- Fixed issues with calculating the accumulation split time.
- Fixes for acceleration/deceleration, accumulation stops and speed changes.
- Fixed bugs with connecting from an object inside a visual tool to a conveyor.

- Fixed bugs with the kinematics and finish time not working on side transfers.
- Fixed a bug that created two output port connections from a Decision Point to an exit transfer.
- Fixed issue with accumulation split events being created at time FLT_MAX.
- Fixed issues with place offset onto an entry transfer
- Fixed an issue that sometimes caused items to move beyond their conveyor and ignore other items when doing an inline merge that accumulates to the merge point.
- Fixed exceptions throwing when building conveyor models with code.
- Fixed issues with adding conveyor types and transfer types to models with scaled units.
- Fixed an issue with stopped decision points not being resumed on reset.

FlexSim 7.5.2 (March 4, 2015)

- Several fixes to the conveyor module (See the conveyor module's release notes topic).
- Updated images for the custom user toolbar items.
- Fixed an issue with docking 3 tool windows on the left or right side (the third would docked above the others).
- Fixed issues with the help manual throwing exceptions when there was no internet connection.
- Made it so the Reports and Stats export can export all object types, including module classes.
- Made a time graph only update its stats at the timed intervals (not on repaint), so custom stats don't get called repeatedly.
- Fixed the DWG importer to properly show the layer colors.
- Fixed an issue with the model floor not drawing properly.

Conveyor

- Added an optional "force" parameter to conveyorresumeitem(), which override the stop/resume counting mechanism and force the item to resume.
- Fixed issues caused by conveyors with invalid spatial properties messing up snapping and causing other conveyors to disappear when you drag them.
- Fixed an issue with not being able to connect network nodes to entry/exit transfers
- Fixed an issue with doing an inline divert at a point where items accumulate
- Fixed an issue with items still being rotated when they leave the conveyor system
- Fixed an issue with transfers from faster to slower non-accumulating sections and the item accumulation at the transfer point.
- Fixed a bug where items get off-center over time in a conveyor system.
- Fixed an issue with double-clicking on conveyor transfers.
- Fixed an issue with feeding items into a Combiner.
- Fixed a UI glitch where if you drop a Decision Point or Photo Eye onto a conveyor, it is placed inside the conveyor in the tree.
- Fixed a UI bug with defining the location of an Exit/Entry Transfer on a curved conveyor.
- Fixed an issue with slug building on an accumulating section where other things are done on the section such as stopping/starting items manually.
- Added off-network routing logic for exit-to-entry transfers directly between two conveyors, so a user can dispatch from one conveyor to another that's not connected to it, using conveyorsenditem().
- Added a color property to Conveyor Types. If you defined them manually on the conveyors before, you'll need to redefine them on the conveyor types.

-
- Added an optional delay time for releasing and exiting a restricted area. This can be used to define variable gapping.
- Added Area Restriction examples to the documentation.
- Added a Conveyor Commands topic to the user manual, which links to the Conveyor command group in the command documentation.
- Added a message trigger to Photo Eye and Decision Point types.
  Fixed an issue with motors not properly syncing power and free dog positions across their member conveyors.
- Various other minor fixes.

FlexSim 7.5.0 (January 28, 2015)

- Added a new conveyor module and deprecated the legacy conveyor set.
- Improvements to the User Manual's user interface.
- You can now use the windowtitle attribute for a User Library.
- Fixed a crash when you call stopobject() on a FluidToItem.
- Fixed an issue with Export All Dashboards to HTML not properly laying out the widgets.
- Fixed a bug with importing Chinese characters from Excel.
- Fixed an issue with the Rack never returning to an idle state after calling for a transporter.

AGV

- Fixed a bug with preemption.

AStar

- Made Preferred Paths take priority over barriers, so that you can make a Preferred Path act as a "bridge" across an area where an object barrier would otherwise block it.
- Improved the object barrier implementation to better take into account rotated objects.
- Made a mechanism that allows object members to customize the way the grid is defined for them, specifically to allow objects like the curved conveyor in the new conveyor module to work as an object barrier.
- Fixed a bug that caused TaskExecuters attached to the A* Navigator to not reset their positions on model reset.

Conveyor

- Released a new conveyor module. See About Conveyors for more information.

FlexSim 7.3.6 (October 9, 2014)

- Fixed negative events to go back to previous functionality of setting the event time to the current model time.

-
- Fixed drawRect command drawing dark when shaders were enabled.
- Fixed opening saved Default 3D Views.
- Fixed the Filter Field in the Library to work with User Library Objects.
- Fixed an infinite loop when resuming a Dispatcher with circular connections to it's team members.
- Fixed issues with adding Performance Measures from Dashboard Statistic Widgets with Groups and grouped objects.
- Fixed the double-click feature in the Performance Measures results window graph (to display the associated dashboard).
- Fixed issues with exporting Experimenter results to Web Viewer format.
- Fixed some issues with updating model libraries to the latest version of FlexSim.
- Added flow item packing method to items saved to model libraries.
  Fixed renaming User Commands from the Toolbox to no longer remove the user command code.
- Fixed number precision values on the Financial Analysis widget to correctly store high precision numbers.
- Fixed the Gantt Chart displaying erroneous data when using a collect time.
- Fixed the Duplicate button in Edit Selected Objects so it no longer copies objects inside themselves.
- Fixed the Draw Surrogate option for animation components.
- Fixed the code snippet popup in triggers to allow for multiple lines.
- Fixed grid snapping in the GUI Builder.
- Fixed undos with the sizing panel in the Quick Properties.
- Fixed FlexSim compiling with Visual Studio 2012.
- Updated Toolbox right click options.
- Updated commands documentation for bundle commands to include that they are base 0.
- SQL query columns will now first look for named sub-nodes and then for object labels.

FlexSim 7.3.4 (September 2, 2014)

- Fixed exceptions being thrown when using the Debugger.

FlexSim 7.3.2 (August 28, 2014)

- Fixed a bug in the MTBF/MTTR and TimeTable to allow for all Down States.
- Fixed a bug in the Optimizer GUI when checking Manual stop only.
- Fixed a bug on the merge sort not resetting its input table properly.
- Fixed Model Units for Gallons so it no longer displays as Fluid Ounces.
- Fixed an issue with older models not properly opening Global Tables from the toolbox.
- Fixed a bug in the queue when it has a non-integer maxcontent.
- Restored ExpertFit.

FlexSim 7.3.0 (July 31, 2014)

-
- Added an autosave feature to global preferences.
- Took shaders/shadows out of beta and added preferences for hard and soft shadows.
- Improved support for stereoscopic 3D including shadow support.
- Add a floor object.
- Added an autosave feature (set in Global Preferences).
- Added alignment property to dashboard model input widgets.
- Added a dashboard GUI class model input widget.
- Added support for linking to groups in dashboard statistics widgets.
- Added support for selecting multiple dashboard widgets.
- Dashboard dynamic model input fields and edit fields can be tied to global variables.
- Improved support for Full Screen mode including stereoscopic 3D and multiple monitors (F11 to switch in and out of full screen).
- Improved 3D mouse support.
- Removed the Tools menu and added an organizable Toolbox.
- Added Date Based mode to TimeTables.
- Added optional stateprofile parameter to stopobject and resumeobject commands.
- Added Stop Date & Time to the Experimenter.
  Fixed the Reports and Statistics available classes GUI.
- Fixed an issue with adding/editing triggers in the dashboard causing FlexSim to crash.
- Fixed an issue with exporting multiple TrackedVariables statistics to CSV files.
- Fixed an issue with aligning model input objects to the left/top of the dashboard.
- Fixed Content vs Time graph not collecting data when defined for a time interval.
- Fixed an issue with setting long object names/paths in bundles and dashboard statistic objects.
- Fixed tabbing in Quick Properties causing FlexSim to crash.
- Added FlexScript commands for use with Stat::Fit.
- Added search results to Command Helper window.
- Added copy/paste functionality to Dashboard (Model Input Widgets only).
- Updated the Automatic Data Distinction in the Excel Importer to more correctly import numbers.
- Added sql query() support for LIKE, GROUP BY, and non-trivial expressions in the SELECT column clause.
- Update the Excel Import/Export to handle importing into bundles and export bundle data.

Backwards Compatibility Note: the following changes may slightly change the way updated models behave.

- Expressions can now be used in select columns of an sql query. In order to do this we needed to change our sql grammar to be more compliant with standard SQL. This means you can no longer use the AS clause in a WHERE or ORDER BY statement. You can now only use AS in SELECT and FROM clauses.
- Some parameter types for the query command have changed in c++ for more flexible type definitions. This means if you use the query() command in a c++ dll, you'll need to update your dll's headers and recompile the dll. If you use straight c++ compiled code or FlexScript for your query() command, then this should just update.
- Removed expertfit. Use Stat::Fit.

AGV

- Fixed incorrect travel distance calculation.
- Fixed an issue with shadow rendering while manipulating paths.
- Added help manual file AGV > Objects > Control Point.
- Fixed a bug that threw exceptions when diverting AGVs to an elevator.

-
- Fixed various tool tips.
- Fixed default proximity stop distance.

AStar

- Added a picklist to the Transport Ref for adding TaskExecuter FlowItems to the AStar Navigator.
- Fixed an issue with attaching TaskExecuters to AStar when they weren't already connected to a navigator.
- Updated barriers to draw resizing arrows.

FlexSim 7.1.4 (April 15, 2014)

- Fixed an issue with the Excel Import not properly handling importing multiple sheets from the same file.
- Fixed Basic Conveyors and Merge Sorts not drawing with shaders on.
- Fixed loading SimpleDataType nodes in modules in the Experimenter.

AGV

- Fixed a bug with preemption not working properly.
- Fixed an issue with exceptions being thrown when beginning a travel task.
- Added support for accumulating paths.
- Added a feature to explicitly force a defined AGV orientation on a path.
- Added a feature to allow you to attach a load to the AGV as a trailer/tug.
- Added a quick properties button to switch the primary direction of a path.
- Added better support for preemption.

FlexSim 7.1.2 (March 27, 2014)

- Fixed a memory leak with the 3D view.
- Fixed GlobalTables so they save their cell width and cell height.
- Fixed a bug in the MTBF/MTTR that caused the downtime value in the down function to always be 0.
- Fixed a bug causing Keyframe triggers to not be fired in operator animations.
- Fixed Animation Editor > Draw Surrogate > Main Object Content.
- Fixed Animation Editor > Component > Rotational Centroid fields.
- Fixed issues with editing animations in models where units were not meters.
- Fixed an issue causing global variables to be renamed to NULL when creating multiple global variables at the same time.
- Fixed exceptions being thrown when you try to use shaders in compatibility mode.
- Fixed issues displaying skp files while using the A* Module.
- Fixed an issue with not being able to select views in the GUI builder.
- Fixed DLL loading to properly load DLLs from the model directory.

FlexSim 7.1.0 (March 10, 2014)

- Added Financial Analysis Dashboard Widget.
- Added Custom Dashboard Widget to allow any numeric data to be displayed in the dashboard, including table data, bundle data, global variables, etc.
- Added State Gantt Chart Dashboard Widget.
- Added a FlowItem Trace Gantt Chart Dashboard Widget.
- Dashboard table data can display current state values as strings.
- Added functionality to export dashboards to HTML.
- Moved the statistics tab out of object properties windows and into the Quick Properties window.
- Labels, tables, statistics, global variables and tracked variables can be "pinned" to the dashboard.
- More options to customize Dashboard Widgets (font size, bar size, custom display names, etc).
- More Model Input objects in the Dashboard including Radio Buttons, Listboxes, Trackers and Tables.
- Added picklist options for starting and stopping animations.
- Added picklist option to display labels on FlowItems in the 3D view.
- Moved the User Manual into FlexSim as a dockable window.
- Redesigned Robot GUI.
- Animation variables can point to components in an animation for quick referencing, not just surrogates.

-
- Improved User Toolbar items for Dashboard, GlobalTables, TimeTables and MTBF/MTTR buttons to allow for opening the objects.
- Improvements to shadows, shaders and mesh drawing.
- Improvements to the SKP Reader.
- Removed Model Views utility and placed it in the Quick Properties window.
- Added a "Headlight" feature to light sources.
- Updated Excel Importer/Exporter to handle relative paths of workbooks to the model.
  Improved the Excel Import's excelreadstr() and Automatic Data Distinction to more accurately read in values from excel, including dates and times.
- Added double click to open colors panel to all color wells.
- Fixed model unit scaling with flowitems when creating new models.
- Fixed issues with copying network nodes in containers.
- Fixed a bug in the Excel Import on Reset not handling multiple workbooks.
- Fixed a bug in the timetables causing the duration passed in to picklists to sometimes be negative.
- Fixed issues with closing/reopening dashboard widgets.
- Fixed sizing issues with dashboard widgets.
- Fixed Rack shelf tilt.
- Fixed an issue with having TaskExecuters using navigator logic for offset travel.
- Updated Move into Highlighted object to move into the model if no object is highlighted.
- Fixed a bug with running flypaths using model run speed.
- Fixed issues with adding flowitems to User Libraries.
- Fixed sizing issues with the Crane.
- Fixed a problem where Global Variables were not being properly loaded using save/load state.
- Fixed an issue with saving views in Full Screen Mode.
- Fixes to the Conveyor's drawing.
- Fixed issues with the Shape Frame tab of the Quick Properties window.
- Fixed a bug with the Basic Conveyor causing flowitems to not always exit when they're supposed to.
- Removed FlexSim Chart and save full history.

Backwards Compatibility Note: the following changes may slightly change the way updated models behave.

- The order in which MTBF/MTTR triggers is fired has been changed to reflect the documentation. Now the down/up trigger fires AFTER the down/up function. This may cause problems in models that depend on the down/up triggers firing before the down/up functions.
- Excel Multi Table Import using Automatic Data Distinction will import empty cells as string data, rather than as the number 0.

AGV

- Fixed a bug with cpaddlistener() where the control point wasn't passing the right values into the listener callback.
- Fixed a bug with custom control point allocations not being cleared on reset.
- Fixed a bug where an agv will "jump" if turning around at a path section that doesn't fit its full length.
- Fixed a floating point precision issue where the agv doesn't always deallocate a control area on arrival at the next control point and thus causes deadlock with himself when he needs to reallocate the area after turning around for the next task.
- Fixed a bug with deallocation events not being created at the correct time.

-

AStar

- Fixed calculations for total travel distance on TaskExecuter objects.
- Fixed an issue with using offset travel and traveltoloc tasks.
- Added Help Manual to FlexSim.
- Fixed an issue with TaskExecuter's being set to state TRAVEL_LOADED when the state should have been TRAVE_EMPTY.
- Added VisualTool to objects that can be added to the AStarNavigator as barriers.
- Fixed issues with FixedResource and TaskExecuter objects that are inside container objects.
- Barriers created by FixedResource objects can now be rotated 90 degrees.
  Fixed bug with using AStar in models using units other than meters.

FlexSim 7.0.6 (January 8, 2014)

- Fixed an issue in the installer regarding Optquest dll registration.
- Added support for compiling with Visual Studio and Visual Studio Express 2013.
- Fixed the duration passed into the TimeTable down function when the last row is combined with the first row.
- Fixed a bug with time tables when the model start time is on a Sunday.
- Changed the Event List to save its filters when the model is saved.
- Fixed an issue with global variables when saving or loading state files.
- Fixed a bug in Display Date and Time pick option.
- Fixed a bug with exporting bundle string data.
- Fixed an exception when documenting user commands with improperly formatted descriptions.
- Some fixes to the query() command.

FlexSim 7.0.4 (December 3, 2013)

- Fixed a bug with the refresh rate overlay showing for educational users.
- Fixed an issue with library installcomponents not getting installed.
- Fixed a bug with the model start time getting reset incorrectly.
- Fixed issues with Labels not showing up in Dashboard Properties window.
- Fixed issues with saving open properties windows and top level windows.
- Refactored TimeTables to minimize the number of events created.
- Fixed issues with recursive debugging hiding the debug toolbar.

FlexSim 7.0.2 (November 12, 2013)

- Fixed a bug with the Pick Operator with Animation picklist.
- Added scroll bar to Labels page Tree View.
- Fixed bug in Processor that showed item conveying when stopped using STATE_BLOCKED.

-
- Added code to read texture repeats/offsets into the assimp importer so that .ac file textures render more accurately.
- Fixed some import and display issues with SKP files.
- Fixed a bug that was occasionally causing WebKit to crash.
- Fixed a bug in the Presentation Builder that was causing the first flypoint to jump beyond the second flypoint.
- Fixed issues with the TimeTable repeating daily.
- Fixed parqty() issue on nodefunction, user commands, function_s, etc.
- Fixed bug in Dashboard to display current content of Fluid Objects.
- OnCreate now gets fired for all objects inside a container when the container is copied.
- Fixed an exception in the Startup Page's OnPreLoad when there was an invalid recent models path.
- Updated Experimenter PFM graph to draw the box plot on top of replication points.
- Made it so foreign languages' dashboard statistic names will be properly associated with the visible name that they're dragged from.
- Fixed bug with changing the model start time and it not being reflected in the model stop time.
- Fixed reset exceptions on presentation slide.
- Fixed a bug that crashes FlexSim when you call startanimation with a rank that doesn't exist.
- Fixed bug causing FlexSim to crash when copying NetworkNodes.
  Fixed round() to work properly with negative numbers and large numbers.
- Fixed triangular distribution from dividing by 0.
- Fixed issue with debugging on a script window script when the first line is commented.
- Other various bug fixes from the development list.

FlexSim 7 (October 14, 2013)

- New User Manual with a more usable structure.
- 64-bit version (enables FlexSim to use more RAM).
- Windowing interface overhaul to use a docked window paradigm.
- Created a Quick Properties docked window that is context sensitive. The window will display the most used properties based on the current selection or the active document window.
- Tree Find/Replace is now integrated with the Quick Properties window and has support for caseinsensitive searches as well as searching for node names.
- Library Icon Grid enhancements to include filtering, collapsible groups and edit modes.
- Library Icon Grid is context sensitive and changes its display based on the current selection or the active document window.
- Added a sampler button that is placed throughout the software to allow users to sample images, 3D media, objects, nodes, numbers, strings and colors. The sampler helps to eliminate some need for writing code.
- Downloads page that gives functionality to download and install Modules, 3D Shapes, Images and Models.
- Added a Measure/Convert tool
- Improved script console allowing scripts to be saved both in individual models and to the user environment. You can also now debug your script console code.
- Improved Presentation Builder interface.
- Improved the Flowitem Bin interface including making packing modes for container flow items visible and editable. Flowitem shapes may be changed through a drag and drop from the Library Icon Grid.
- Flowitems can now have their own custom animations.

-
- Improved the employment of shape frames in FR objects and Flowitems.
- Added a No Select flag to all objects.
- Added a multi-table Excel export and overhauled the Excel interface to match the MTEI. The new MTEI includes an option to automatically reimport tables on reset.
- Improved Animation Creator, including dynamic animations using animation variables, more detailed editing of keyframes, and keyframe triggers. 3D shapes may be added to an animation through a drag and drop from the Library Icon Grid.
- Created a global model start date/time that is tied to TimeTables. A stop date/time may also be specified.
- Revamped TimeTable window. A daily or weekly schedule may now be imported through the MTEI.
- Added and updated several picklist popups removing all text based picklist options.
- Improvement in the Code Editor and other areas where logic is defined through draggable constructs in the library icon grid and sampler buttons throughout popups and picklist widgets to automatically add code, etc. FlexSim commands also display a short description when typing in the code editor.
- Picklist fields and many popup fields have code highlighting and autocomplete.
- Added some FlexScript implementations of lambda expressions.
- Better debugger that allows you to access the tree and other areas of FlexSim while in debug mode. Hovering over variables during debug mode will display their current value.
- Panel control GUI enhancements.
- Added dashboard constructs that will replace most need for the GUI builder: Users can now do model input through dashboards instead of having to use the GUI builder. Multiple dashboards may be created.
- You can now pick which navigator a TE is connected to through their properties page (allows you remove them from all navigators).
  New hot keys/accelerators. Ctrl+K and Ctrl+L to resize objects up or down by 5%. Ctrl+W to close the active document window or the active floating window. Updated Ctrl+Tab and Ctrl+Shift+Tab to moved between tabs in the active floating or document window.
- Added the FluidConveyor to the default fluid library.
- Can now view an object's events by right clicking an object in the 3D view and selecting View | View Object Events.
- Complete OptQuest overhaul (includes multi-core support and experimenter integration).
- Better support for importing 3D shapes. FlexSim now supports the following formats: *.wrl; *.3ds; *.dxf; *.stl; *.skp; *.dae; *.obj; *.ac; *.x; *.ase; *.ply; *.ms3d; *.cob;*.md5mesh; *.irr; *.irrmesh; *.ter; *.lxo; *.csm; *.scn; *.q3o; *.q3s; *.raw; *.off; *.mdl; *.hmp; *.scn; *.xgl; *.zgl; *.lvo; *.lvs; *.blend
- Added a new mesh class for drawing in OpenGL.
- Stereoscopic 3D rendering (requires workstation Quadro or FireGL card for frame-sequential rendering).
- Enhanced graphical compatibility with integrated Intel cards.
- Improved 3D rendering, including shadow rendering, specular highlights on 3ds objects, bump maps, parallax maps, etc.
- Module Development SDK, including:
    - Added SimpleDataType data type, which is a low-overhead class for fast, memory-efficient aggregation of data and for better object-oriented module code, with an easy mechanism for saving in the tree.
    - Updated visual studio wizards that work with VS 2012 o A module sample tutorial.
    - More Documentation.

Backwards Compatibility Note: the following changes may slightly change the way updated models behave.

- Removed 2D Shapes from objects.
- Removed the Planar view.
- Many open gl commands have been deprecated. The model update mechanism tries to replace all old usages with the new graphics usages. Usually this should work, but in some cases it may not. Note that

- glBegin(), glEnd(), glVertex(), glNormal(), glTexCoord() have all been completely deprecated, and eventually will no longer work. Going forward you should use the mesh api.
- The spaceobject() command has been deprecated and no longer works. In optimizing the graphics engine we realized that a 5-20% refresh rate improvement can be attained simply by removing functionality that is solely there to make the spaceobject() command work. So we've deprecated this command. It is still in the command list so models will update, but updated models that use it will have weirdly drawn shapes.
- OnLoad is no longer dispatched by the engine when a project/tree is loaded. If you have custom objects that depend on this event, you will need to use some other mechanism to fire logic when the model loads, i.e. through the OnModelOpen, or through model libraries' OnModelOpen.
- Kinematics functionality has changed so that by default kinematics will automatically be pruned off as you pass their individual end times. Note this required a restructuring of the kinematics data, so if your models don't do it on reset anyway, you'll need to re-initialize kinematics in models that are updated from older versions.
- In previous versions, getdatastat() was documented incorrectly for the parameter p2 (degrees of freedom). It was actually interpreting p2 as the number of samples in the set, not degrees of freedom. We've fixed that by simply not using that parameter and inferring the number of samples/degrees of freedom from other parameters. This means if you used this command previously it will return different/better results in this version. Also we've changed the way the confidence interval "clamps" to percentages in order to be "safer". Again this affects the values that were returned in previous versions vs this version. See the command documentation on getdatastat() for more information.
- We changed the name of the class FlexsimObject to FlexSimObject in-line with our naming scheme going forward. We have implemented an update script that replaces all instances of "FlexsimObject"

- with "FlexSimObject" in updated models. This means if certain things in your model are dependent on the name "FlexsimObject" (I don't know what that could be, except for maybe dll code that uses the name FlexsimObject) there may be issues with the update.
- The assimp 3ds importer is translating some of the files (namely the robot clamps) in a way that is different than our old 3ds importer. It is possible that other 3ds files may need their offsets manually adjusted after updating.
- Fixed the per-event 50-flowitem receive limit on the queue. This will break old models if your model is dependent on this bug.
- Fixed a bug with deceleration on a network when the task executer is blocking space and is given two travel tasks in a row (this might change old models)
- Several attributes were removed, so if you use these attributes in your model you will need to update your model properly:
    - Removed Attributes: assertshape, asserttexture, billboard, distcutoff, events, instances, OnCaptured, OnCollision, OnInterrupted, state_graph, state_histo, state_percent, stats_contenthisto, stats_customgraphs, stats_throughputgraph, stats_throughputgraphmaxpoints, stats_throughputhisto, tables, textureaxis_s, textureaxis_t, travelstarttime, traveldirection, travelendtime, traveltimealpha, traveltimebeta, travelvpeak, traveldistance, travelstartx, travelstarty, travelstartz, travelendx, travelendy, travelendz, travelvmax, travelacc, traveldec
    - Removed Draw Attributes: ignoredraw, ignorezbuffer, nochildrotate, nochildscale, noondraw, nopredraw, shapetype, reflective, luminous
    - Removed Commands: travelto, traveltoupdate, ntravelto, ntraveltoupdate

FlexSim 6 (March 23, 2012)

- Enhanced the experimenter to use multiple cores.
- Added a new web browser GUI widget.
- Added a new Dashboard window with HTML5 canvas statistics graphs.
- Redesigned the experimenter interface to integrate the new statistics objects.
- Added new experiment variable options "number of objects in group" and "number of task executers."
- Developed web accessibility: Opening, configuring, running, and viewing models over the web (using a web browser or handheld device).
- Added a new AutoCAD dwg importer.
- Added model units and conversion windows. When building a new model, a screen will ask you what model units you will be using. When updating an old model, a screen will ask you what model units were used to build the old model. These settings are stored in the Tools folder of the model.
- Modified the picklist and trigger gui widgets to be easier to use.
- Created a new node datatype (DATATYPE_BUNDLE) for storing large amount of information efficiently (see documentation of bundle commands).
- Implemented a way to package media (3D shapes and bitmaps) into the model file so that you only need to distribute one file instead of a whole directory of files.
- Added an embedded command documentation window that can be opened by highlighting a command and pressing F1 in the code window or tree view.
- Added a FlexScript call stack to the debugger.
- Added a step-in function for FlexScript user commands and nodefunctions in the debugger.
- Added a logic builder interface for writing FlexScript logic without writing code. (You can change the default editor back to Code by unchecking 'File > Global Preferences > Environment > Use Logic Builder by Default'.)

- Enhanced the flexibility of pull logic and added a new pullitem() command for use in the Pull Strategy trigger.
- Improved templates to allow for popup gui windows on picklist options.
- Added a mechanism for multiple state profiles.
- Changed tables to always show the headers despite scrolling.
  Fixed a bug that was messing up the save operation when out of memory.
- Toggled the large-address-aware switch so that FlexSim can allocate more memory.
- Added floor() and ceil() commands.
- Added a switch for hashing the node's subnodes' names for quick lookup.
- Added an option in the Labels tab so that labels' values can be automatically reset.
- Other various fixes from the development list.

Backwards Compatibility Note: the following changes may slightly change the way updated models behave.

- Fixed a critical bug in curved network length calculations.
- Fixed issues with gettenetnode() and distancetotravel() on TEs that use "do not travel offsets and block network space."
- Made a change to distancetotravel() to base the "back-to-node" distance on the center of the object instead of the object's location.
- Changed pulling to no longer override the send-to. Now both send-to and pull must check out to transfer a flowitem.
- Changed receiveitem() so that it doesn't behave as if it were pulling.
- Made the Rack's OnEntry trigger fire before evaluating the dwell time.
- Added a new random number generator to generate seeds based on the replication number for the existing random number generator. Email support if you need a script to initialize the streams of a particular model back to the values used in prior versions.
- Added a new overload to the command tonode() to handle large memory addresses. Dlls will need to be recompiled with updated headers to obtain this fix.
- Fixed a bug where an endspeed of 0 wasn't properly telling a task executer to continue at full speed at the end of a travel task. The behavior now correctly matches the documentation.

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 reallife system, then study that system in a shorter time frame and 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 educational 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 at 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.

Interacting With FlexSim

Topics

- Creating An Object
- Naming An Object
- Editing Objects
- Connecting Objects
- View Navigation

Creating An Object

Objects can be created through entering a Create Objects mode, by drag-and-drop or through the Quick Library:

1) Enter the Create Objects mode by clicking and releasing on an object in the Library window. Click again in the 3D view to create an object.

2) Alternatively, to enter the Create Objects mode, you may click on the  button on the main toolbar. Then, click the object you wish to create in the Library and click again in the 3D view where you want the object to be created.

3) Click and hold the left mouse button on the object in the Library, then drag it to the position you want to place it in the model and release the mouse button.

4) To open the Quick Library, double click on empty space in the 3D view (not on an object), then click on the object to create.

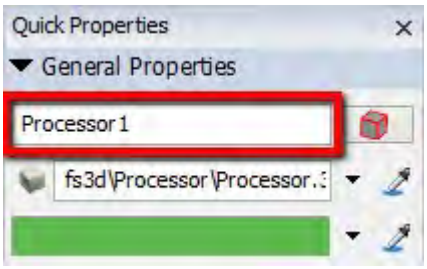
Naming An Object

•

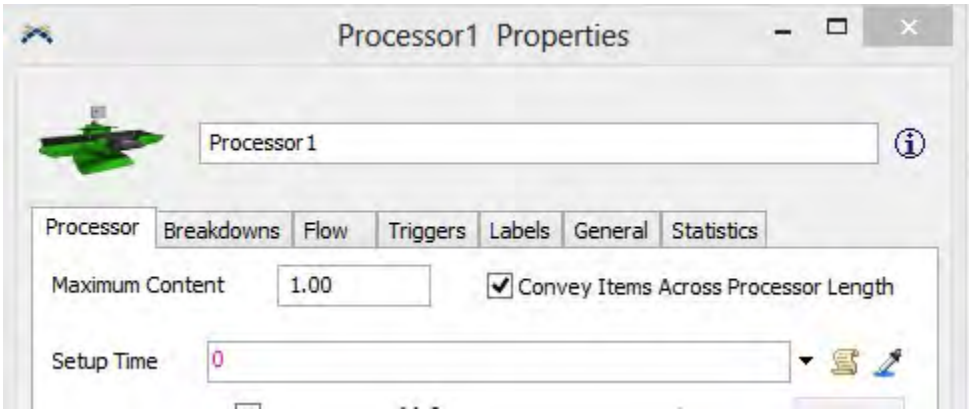
As objects are created, they will be given default names such as Source#, where # will start at 1 and increment until a unique object name is found.

There are two methods to rename an object:

1) Click on the object in the 3D view to display the object's properties in the Quick Properties window. Then edit its name at the top of the Quick Properties window.



2) Double-click it to open its Properties window. Then edit its name at the top of the window and press Apply or OK.



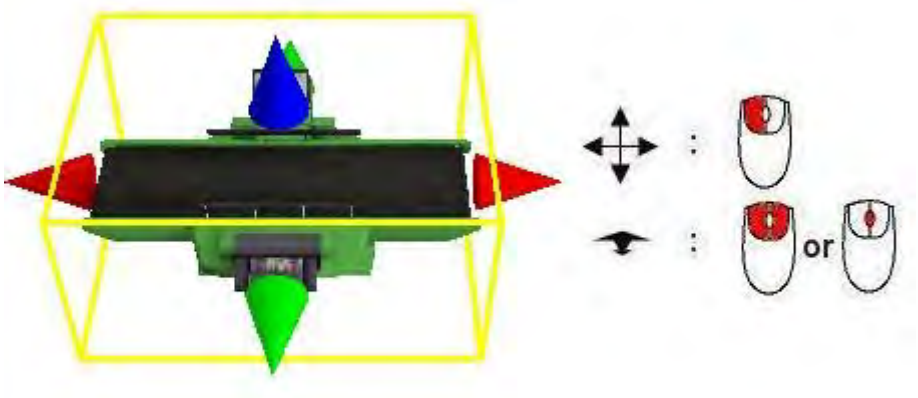
Editing Objects

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 then dragging the mouse forward and backward.

Size and Rotation – To edit the object's size and rotation first click on the object. 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 drag the mouse forward or backward.

Click the left and right mouse buttons down on any of the sizer arrows and drag to resize the object proportionally.

You may also scale an object up or down by 5% by holding the Ctrl key and pressing the K or L key. Note: You can toggle Resizing and Rotating Objects to be on or off by selecting the main menu option Edit > Resize and Rotate Objects.



Properties – 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. See here for more details.
Destroying Objects – To destroy an object, click on that object and press the Delete key.

Connecting Objects

Ports are created and connected in one of two ways:

1) 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:

	Output - Input	Center
Disconnect	Q	W
Connect	A	S

You can display the Quick Library for fast object creation with connections by A or S connecting from an object to blank space.

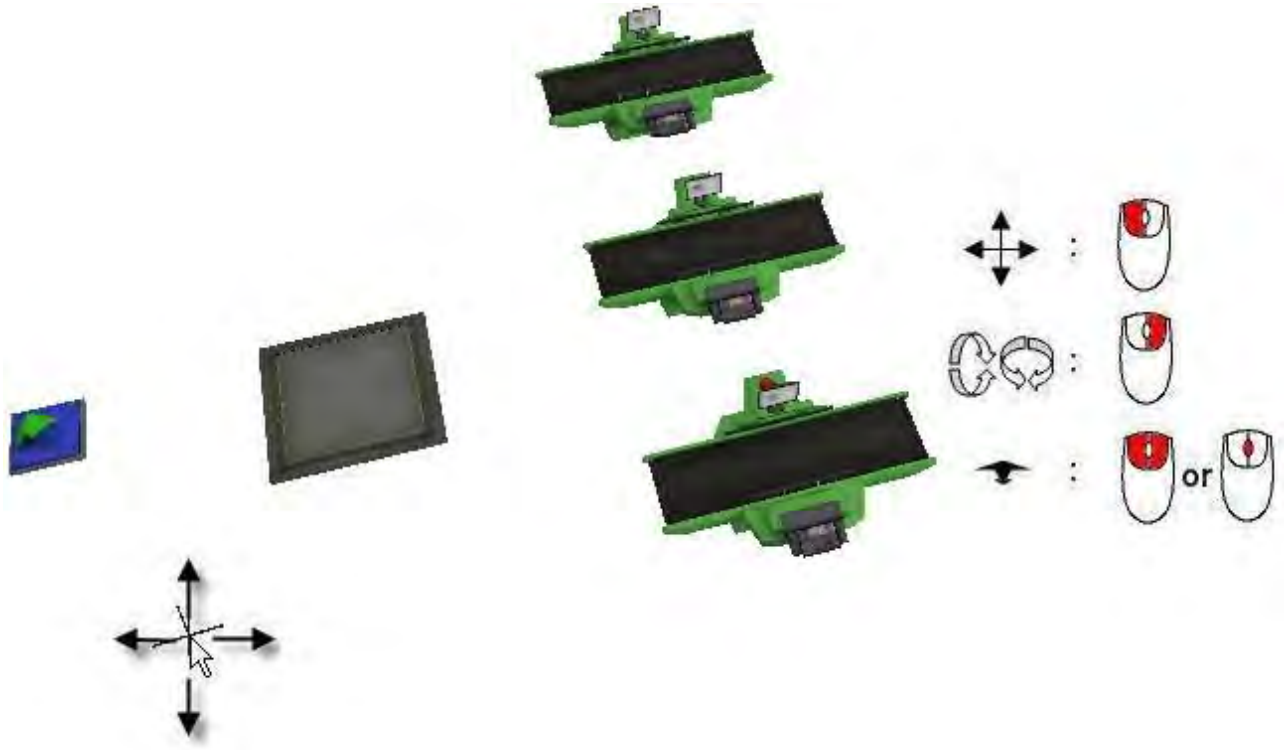
2) By entering the Connection Mode, which can be entered by clicking the  button in the main toolbar. Once in the Connection Mode, there are a couple of ways to make a connection between two objects. You can either click on one object, then click on another object, or you can click and drag from one object to the next as with method one. Either way, keep in mind that the flow direction of a connection is dependent on the order in which you make the connection. Flow goes from the first object to the second object.

Connections can be broken by clicking the  button then clicking or dragging from one object to another in the same manner as when you connected them. Center port connections are not affected by the order in which the objects are connected.

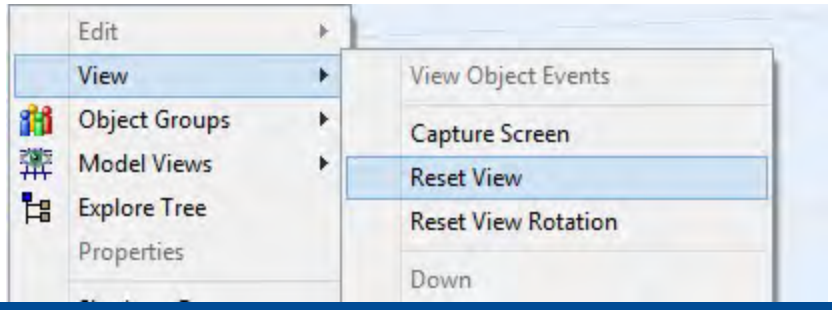
View Navigation

Basic Navigation – To move the model view point, click in an empty area of the view with the left mouse button, and drag the mouse around. You can also click and drag with the middle mouse button which will ignore any clicks on objects. Holding the ALT key down and clicking in the view will also ignore any objects. 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.

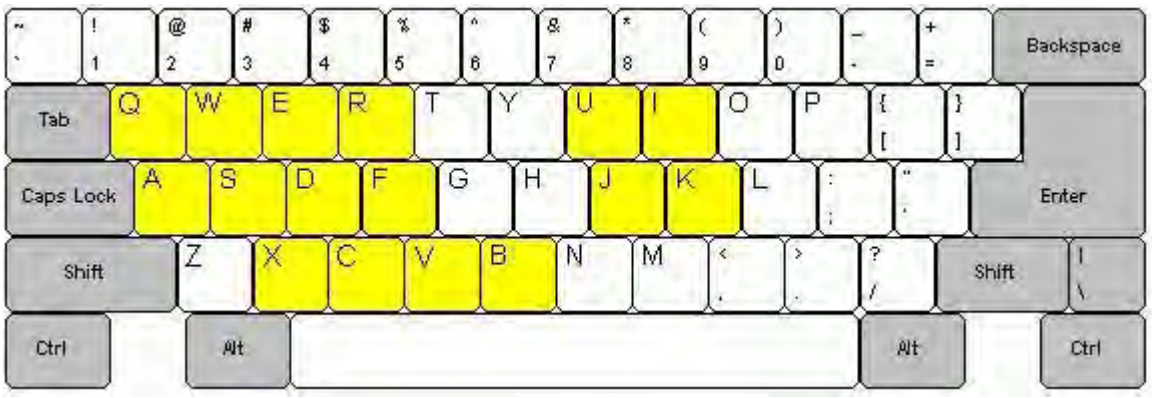


Reset View – You can reset the view to the default view point by right clicking in empty grid space, and select the menu View > Reset View.



Keyboard Interaction

When you are working in the 3D 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.



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 travellers, 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, click on an empty area in the 3D view and make sure the Show Connections button is checked in the Quick Properties window. 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 travellers, 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.

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. 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.

Ignore Objects

Hold the ALT key down to cause all objects in the 3D view to ignore any mouse clicks. This can be useful for moving around in a view with a lot of objects. You can also click and hold the middle mouse button and drag to move around the view, ignoring all objects.

Hot Keys / Accelerators

Ctrl + Space - Start and stop the model run.

Ctrl + Left Arrow - Reset the model.

Ctrl + Right Arrow - Step to the next model event.

Ctrl + Up Arrow - Increase the simulation run speed.

Ctrl + Down Arrow - Decrease the simulation run speed.

Ctrl + F - Find text in the open view.

Ctrl + H - Find and replace text in the open view.

Ctrl + C - Copy the selected object(s) to the clipboard.

Ctrl + X - Cut the selected object(s) to the clipboard.

Ctrl + V - Paste the object(s) in the clipboard.

Ctrl + T - Open a new tabbed document window (based on the currently active document window, 3D or Tree only)

Ctrl + Tab - Switch to the next window tab.

Ctrl + Shift + Tab - Switch to the previous window tab.

Ctrl + L - Scale the selected object(s) up by 5%.

Ctrl + K - Scale the selected object(s) down by 5%.

Ctrl + Shift + D - Add a keyframe to the presentation builder.

Ctrl + W - Close the active document window or floating window.

Ctrl + Alt + E - Switch Environments. If an environment is currently active, exit the environment, otherwise, enter the Minimal environment.

F1 - Open the Command Helper and search for the selected text.

F11 - Switch in and out of Full Screen Mode (3D View).

P - Takes a screen shot of the active view and saves it to projects folder/screenshots (adds pointer data if highlighted a node in the tree)

Tree Window Shortcuts

The following are available in the Tree Window:

Spacebar - Insert a new node after.

Enter - Insert a new node into.

Del - Delete selected node(s).

N - Add number data to the highlighted node.

T - Add string (text) data to the highlighted node.

O - Add object data to the highlighted node.

P - Add pointer data to the highlighted node. Shift + Del - Delete node data.

FlexSim Terminology

Before you start your first 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, a queue of waiting calls at a customer service center, etc. 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 Library Icon Grid.

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. Flowitems are managed in the Flowitem Bin.

Labels

Labels are strings or numbers that are stored on Flowitems and objects. Labels can be dynamically altered through the course of a process flow. Labels can be useful for storing information like cost, processing time and other information. Labels can be accessed through the object's quick properties or its Labels Page. Click here to learn more about labels.

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 center.

- Input and Output Ports: These 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 (found in the Conveyor Module), 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.

- Center Ports: These ports are used to create references from one object to another. A common use for central ports is for referencing TaskExecuter objects such as Operators , Fork Lifts , and Cranes from FixedResources such as Processors or Queues.

First Model

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 item type 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 bottleneck 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 before the tester important?

Applying the Model to Different Industries

While we are using the manufacturing industry for this example, the same 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. As each job is finished, it is placed in a queue for the cashier to finalize the sale and give to the customer. However, sometimes the customer is not satisfied with the job that was done. In such cases, the job must be given back to the appropriate 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 customer queue and the time they spend waiting, 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 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 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

If at any time you encounter difficulties while building this model, a fully functional tutorial model can be found at http://www.flexsim.com/tutorials.

Steps

- Step 1: Start FlexSim
- Step 2: Select Units
- Step 3: Create the Objects
- Step 4: Connect the Objects
- Step 5: Define the Inter-Arrival Time
- Step 6: Assign an Item Type and a Color
- Step 7: Define the Queue's Maximum Content
- Step 8: Define Queue1's Routing
- Step 9: Define Process Times
- Step 10: Define Queue2's Maximum Content
- Step 11: Define Tester's Process Time
- Step 12: Define Tester's Routing
- Step 13: Reset and Run the Model
- Creating a Dashboard
- Randomness
- Results

Step 1: Start FlexSim

- Open FlexSim by double-clicking on the FlexSim icon on your desktop. The Start Page will appear. Select the "New Model" option in the upper left hand corner of the window.

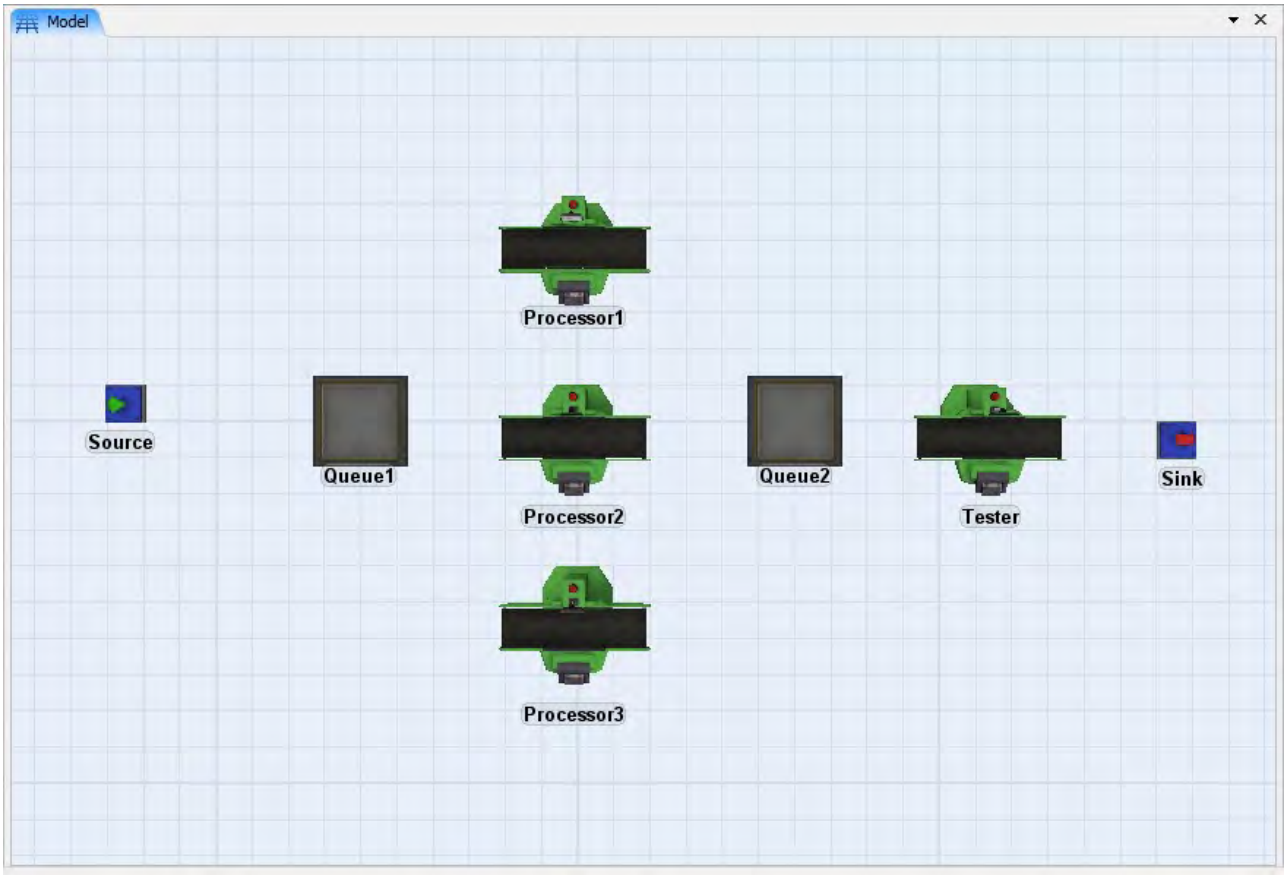
Step 2: Select Units

FlexSim allows the user to select appropriate units for a model. By default the Model Units window will appear for each new model. You can select units for time, length, fluids and a Model Start Time. The units you choose will be used throughout the model. The Model Start Time may be changed after the model is created, however, the Time, Length and Fluid units CANNOT be changed. For this model, use the following:

- Time Units: Seconds.
- Length Units: Meters.
- Fluid Units: Liters.
- Model Start Time: Leave as default.

Step 3: Create the Objects

- Create a Source, two Queues, four Processors, and a Sink in the model. Name and place them as shown below (note that one of the Processor objects will be the "Tester").
- To review the process for creating objects in FlexSim, refer to the Creating An Object section of the Interacting with FlexSim page. To review how to rename an object, refer to the Naming An Object section.

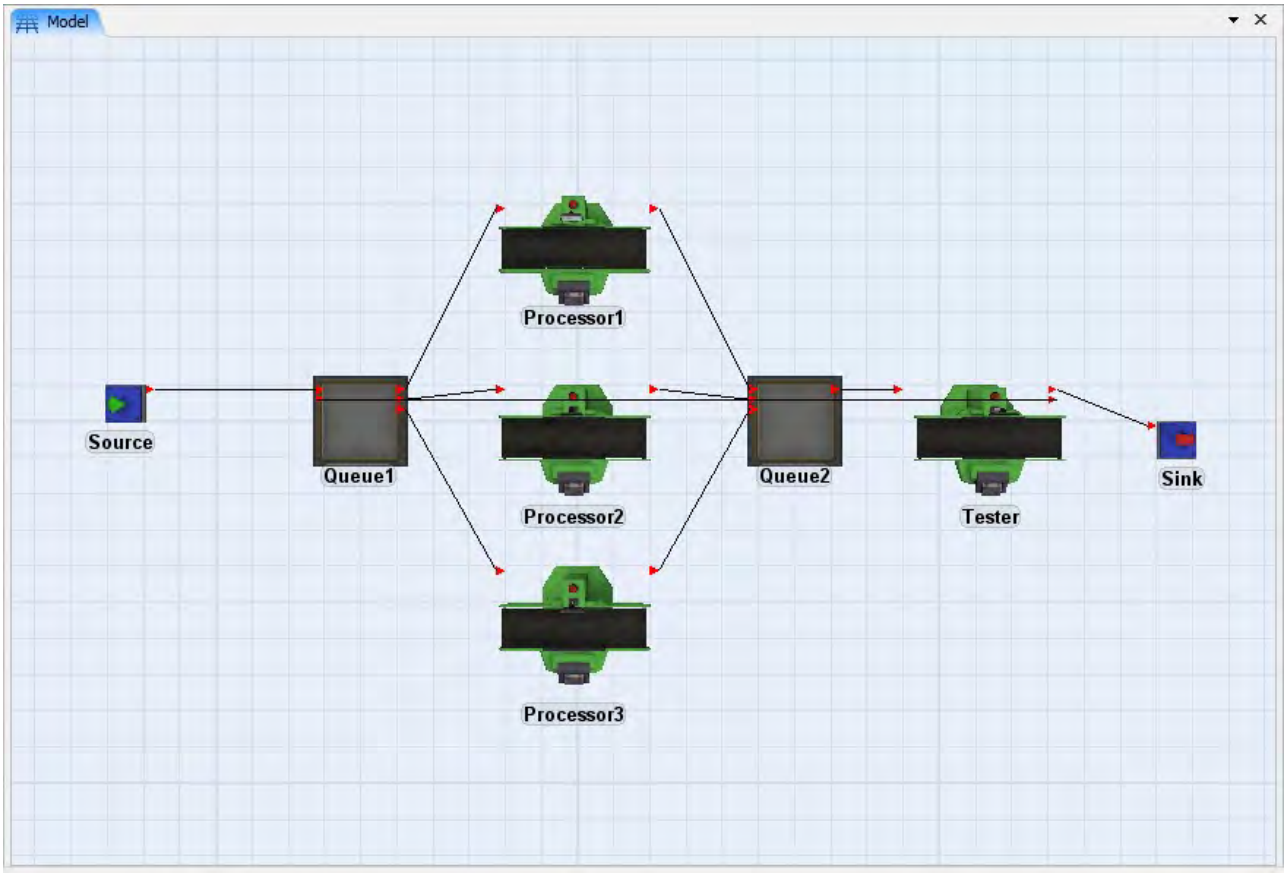


Step 4: Connect the Objects

To review how to connect objects, refer to the Connecting Objects section of the Interacting with FlexSim page.

Notice the Tester object has an output connection to Queue1. This will allow rejected items to be sent back to the start of the process.

- Connect *Source* to *Queue1.*
- Connect *Queue1* to *Processor1*, *Processor2*, and *Processor3.*
- Connect *Processor1*, *Processor2*, and *Processor3* to *Queue2.*
- Connect *Queue2* to *Tester.*
- Connect *Tester* to *Sink* and *Queue1.*



The next step is to change the properties 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.

Each object has its own properties window through which data and logic are added to the model. Doubleclicking on an object accesses the object's properties window.

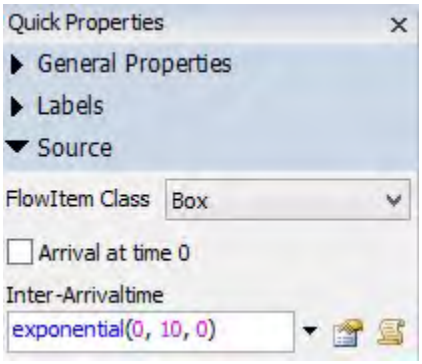
For this model, we want three different product types to enter the system. To do this, each flowitem's type will be assigned an integer value between one and three using a uniform distribution. This will be accomplished using the source's exit trigger.

Step 5: Define the Inter-Arrival Time

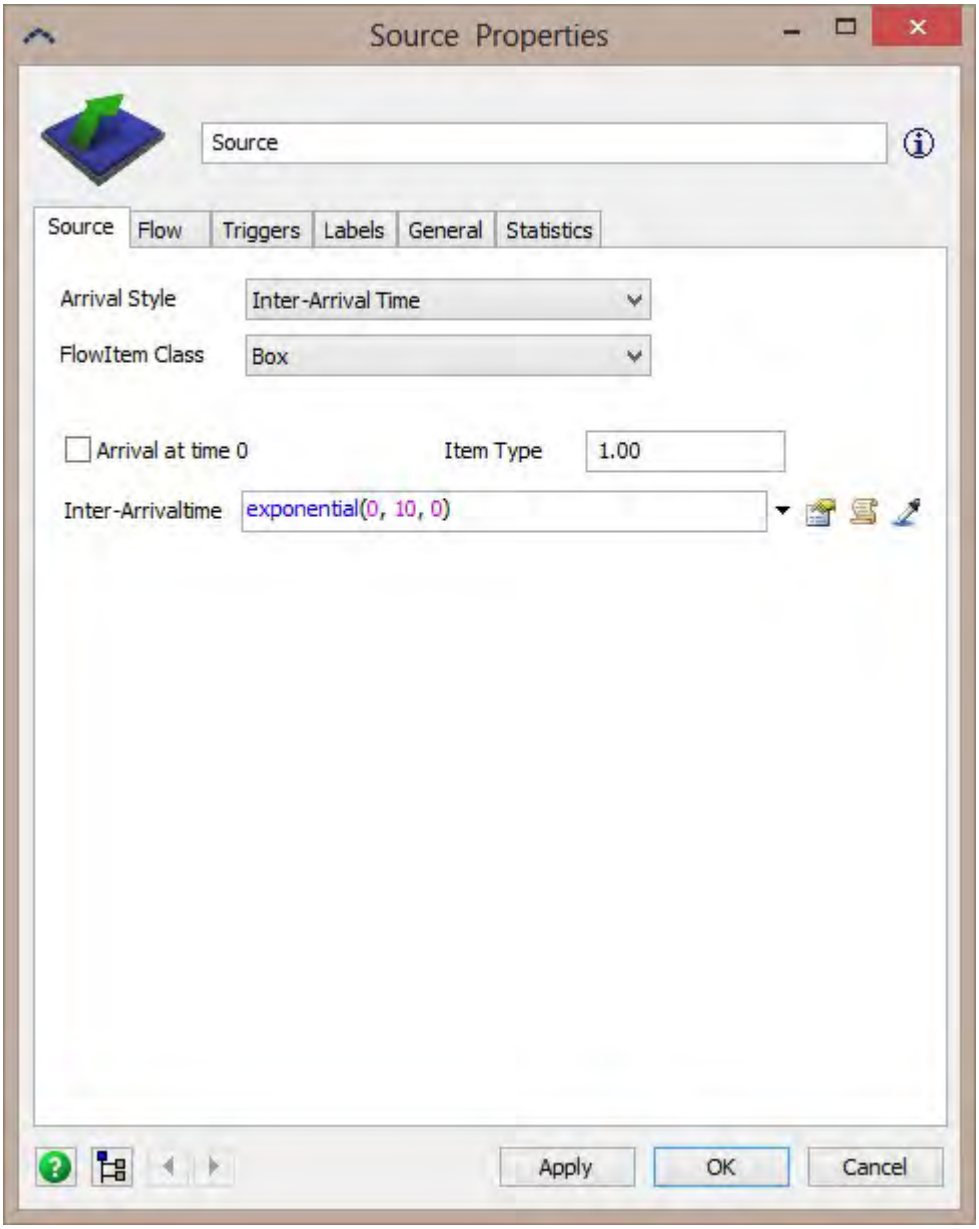
Products arrive every 5 seconds, exponentially distributed. The Source, by default, uses an exponentially distributed inter-arrival time, but you will change the mean of that distribution. Statistical distributions like exponential distribution are used throughout simulation in order to model the variations that occur in reallife systems.

You may edit the Source's Inter-Arrival Time from two different windows:

- 1) Click on the *Source* to bring up its properties in the Quick Properties window.

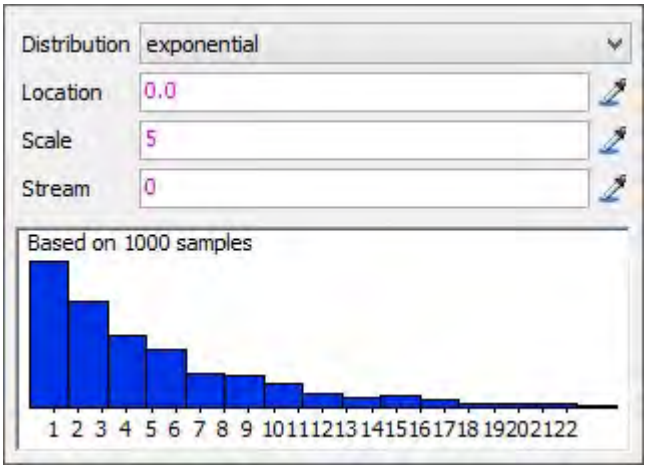


- 2) Double-click on the *Source* to bring up its Properties window.



- On the Source tab, click on the  button. A popup will appear.

- Set Distribution to exponential. o Set Location to 0.
- Set Scale to 5.
- Set Stream to 0.



Click anywhere outside the popup to save these settings.

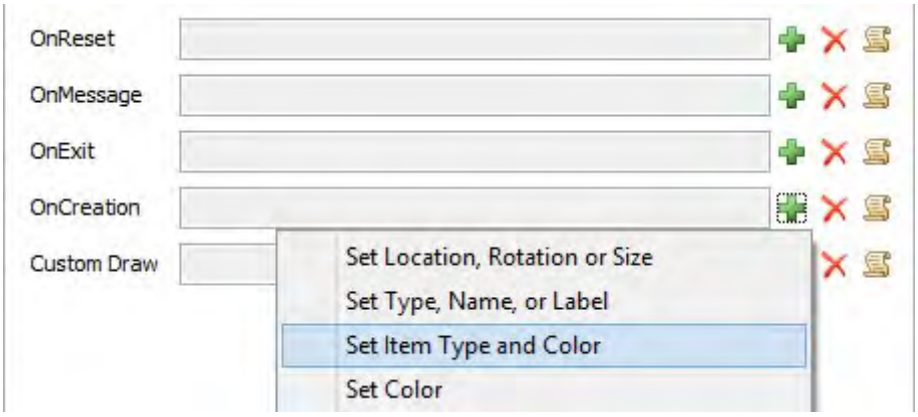
Remember that units were set at the beginning. Setting Scale to 5 sets the mean of the distribution to 5 seconds. If the units had been set to hours, the mean would have been 5 hours.

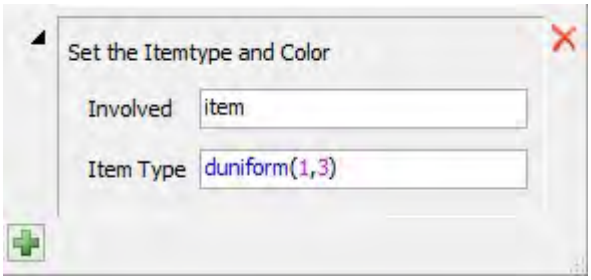
- If you edited the Inter-Arrival time through the Quick Properties window, you'll need to open the Source's Properties window in order to perform Step 6. This can be done by clicking the More Properties button under the General Properties section of the Quick Properties.

Step 6: Assign an Item Type and a Color

The next thing we need to do is assign an type label 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 best way to do this would be to change the type in the OnCreation trigger of the Source.

- Click the Triggers tab. Add a function (press the button) to the OnCreation trigger. Select Set Item Type and Color from the list. A popup will appear.



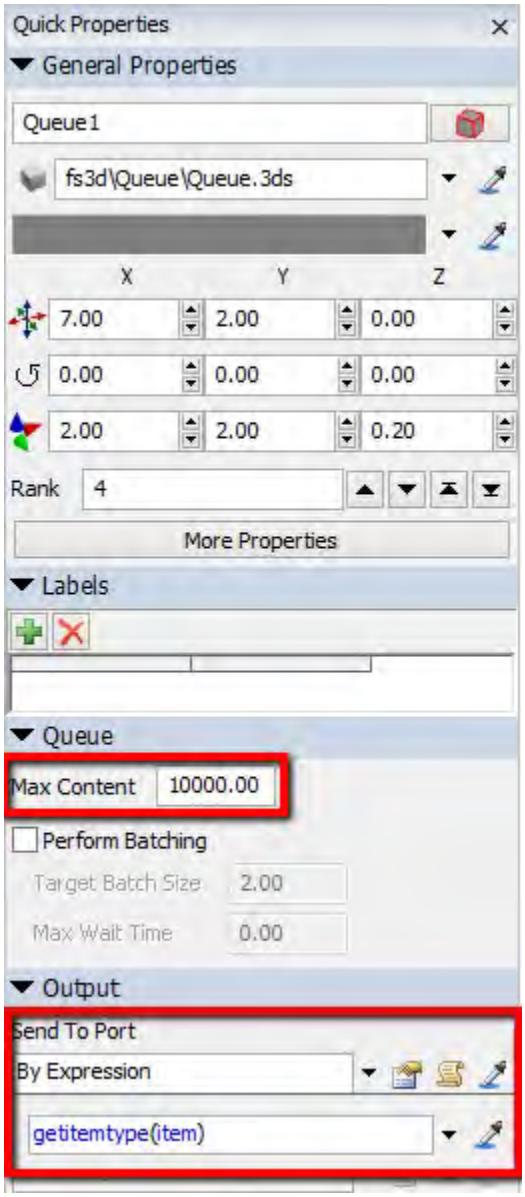


The duniform distribution is similar to a uniform distribution except that instead of returning a real number it will only return whole numbers. Click OK to apply the changes and close the window.

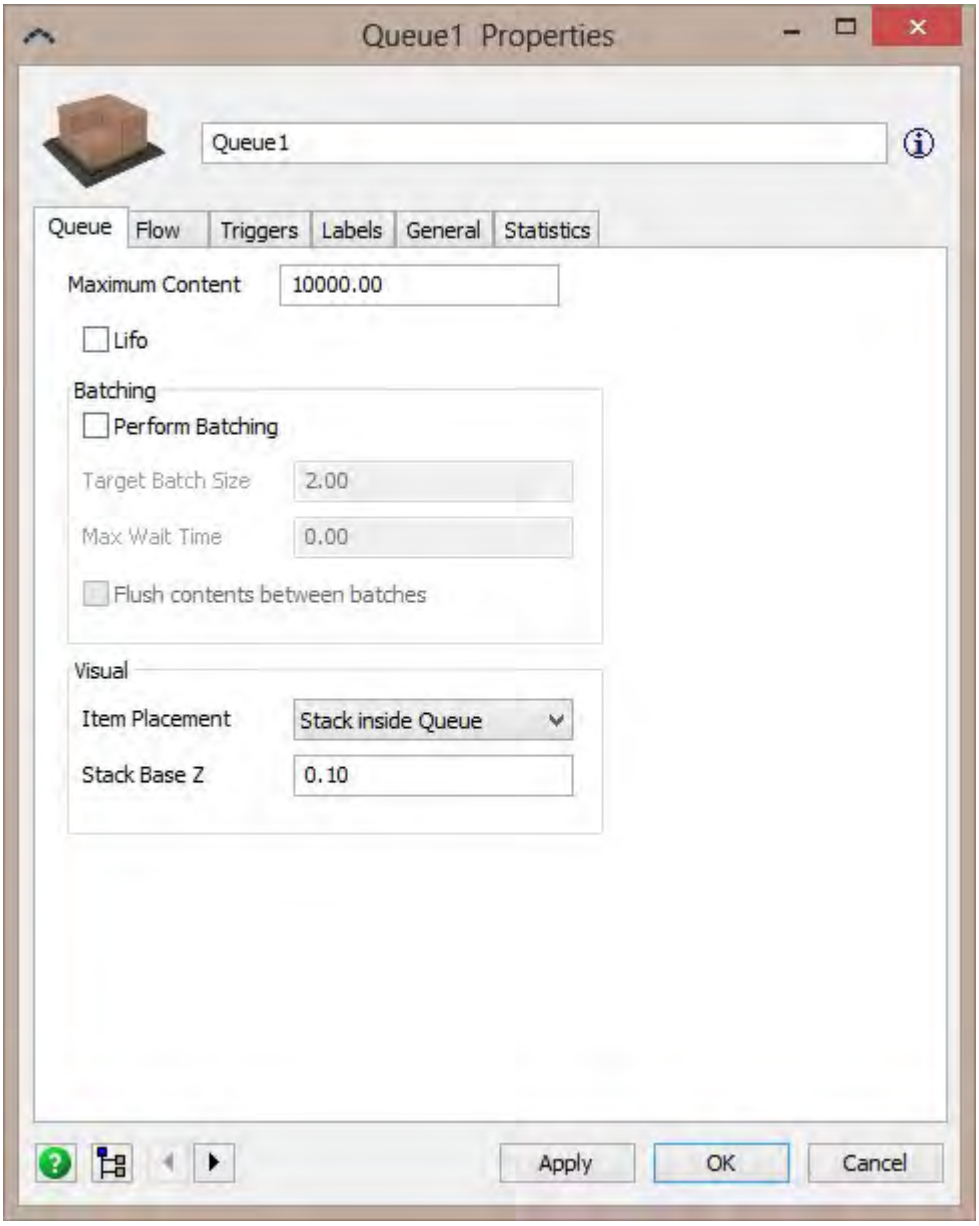
Step 7: Define the Queue's Maximum Content

The next step is to edit the Queue. There are two things we need to configure on *Queue1*. First we need to set the Maximum Content of the Queue. Second, we need to have the Queue send type 1 to *Processor1*, type 2 to *Processor2*, and type 3 to *Processor3*.

This step, along with step 8 can be done through the Quick Properties window as shown below, or by opening the Queue's Properties window as described.

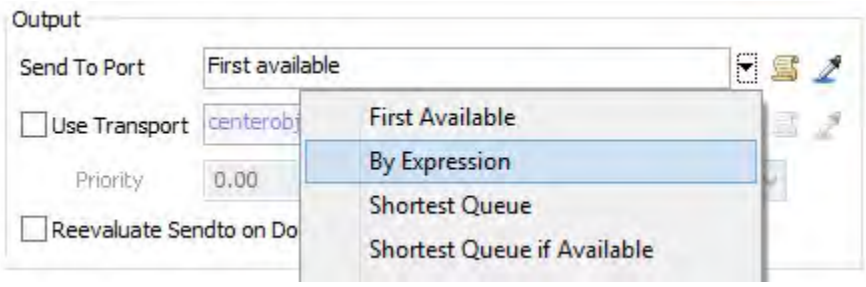


- Double-click on *Queue1* to open its Properties window.
- On the Queue tab, change the Maximum Content to 10000.
- Click Apply, but do not close the Properties window.

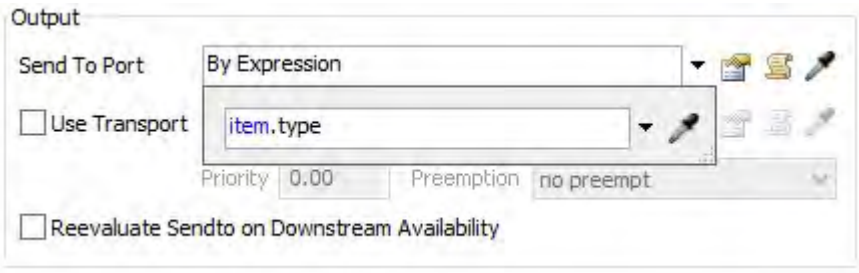


Step 8: Define Queue1's Routing

- Click the Flow tab.
- Under Output, select By Expression from the Send To Port drop-down list.



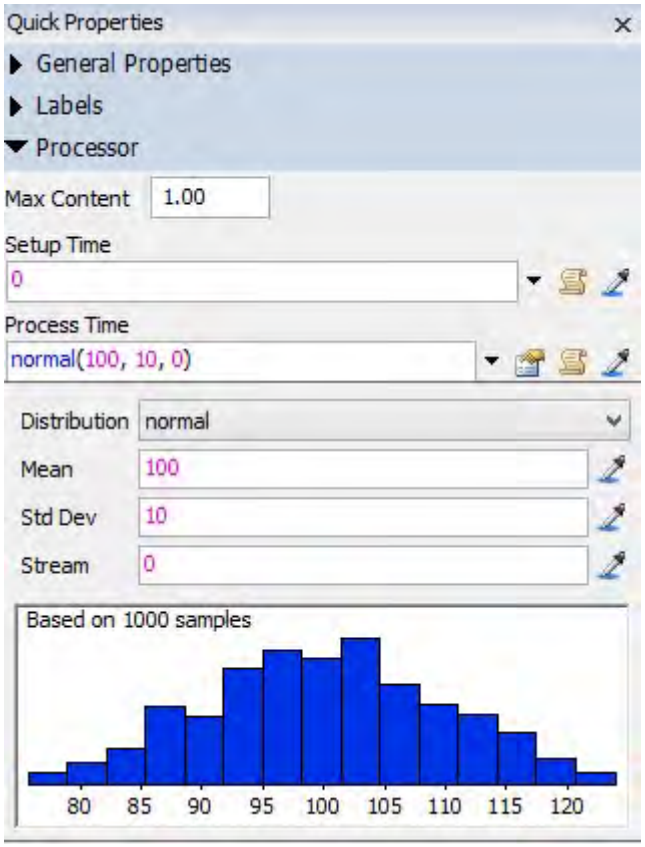
- A popup with suggested expressions will appear. The default expression for By Expression is item.type . This will send type 1 to port 1, type 2 to port 2, and so on. Click anywhere outside popup to close it, and then click OK to apply the changes and close the window.



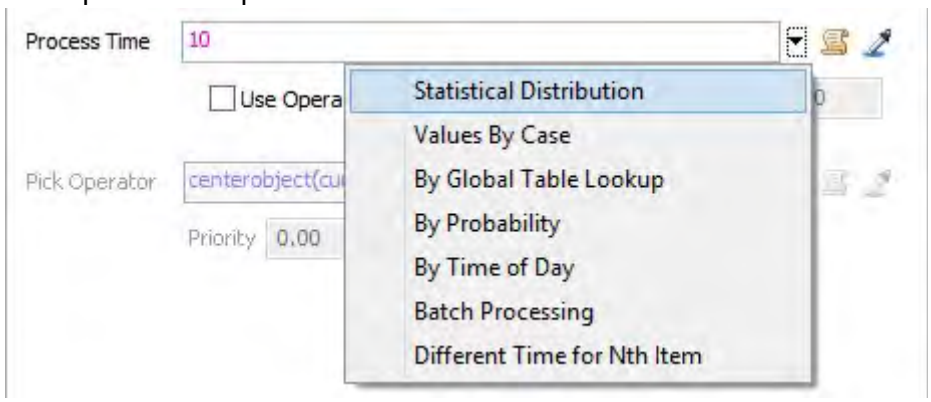
Step 9: Define Process Times

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

As described for Step 7, the Process Time can be set through the Quick Properties window by clicking on the object once in the 3D view.



- Double-click on *Processor1* to open its Properties window.
- On the Processor tab, select Statictical Distribution from the Process Time list.
- In the Statictical Distribution popup, set Distribution to exponential. Use the default parameters given for this distribution.
- Click OK to apply the changes and close the window.
- Repeat this step for *Processor2* and *Processor3*.



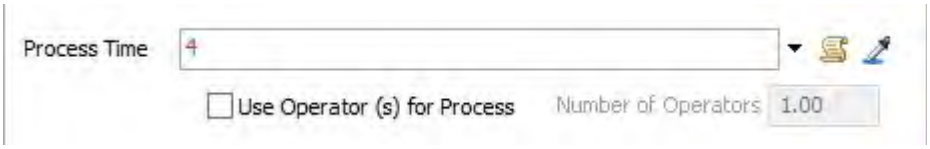
Step 10: Define Queue2's Maximum Content

Follow Step 7 to change Queue2's Maximum Content to 10000.

Step 11: Define Tester's Process Time

As described in Step 9, this can be set through the Quick Properties window as well.

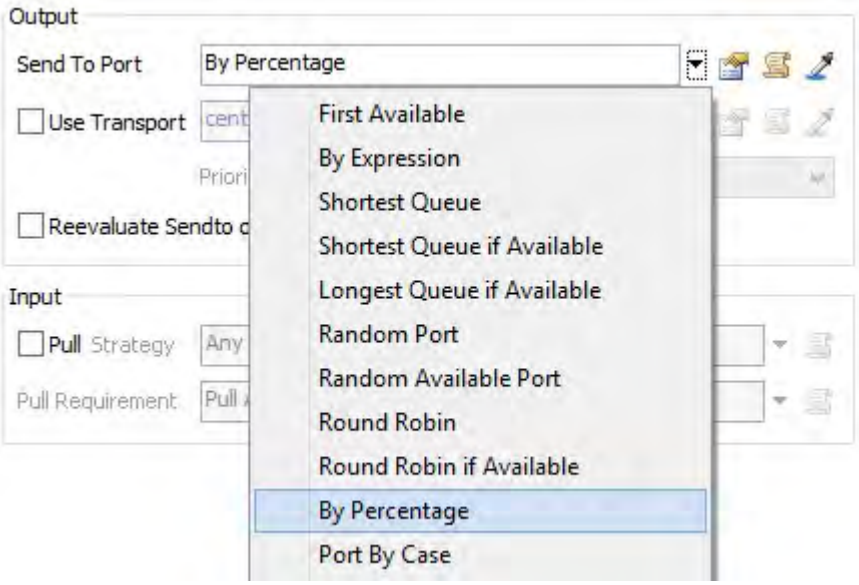
- Double-click on *Tester* to open its Properties window.
- On the Processor tab, highlight all the text in the Process Time field.
- Replace the text with 4. This sets the process time to a constant four seconds.
- Click Apply, but do not close the Properties window.



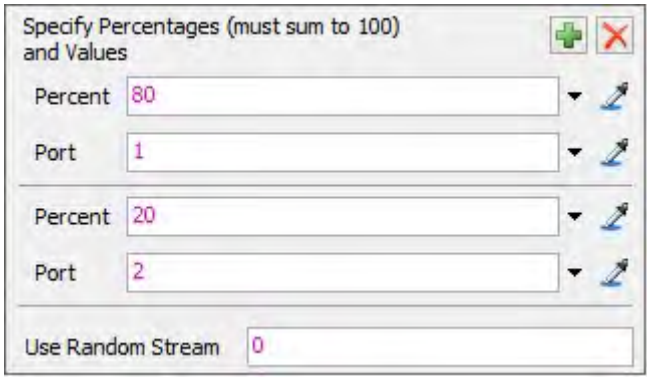
Step 12: Define Tester's Routing

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 the testing station be connected to the sink and the second output port be connected to *Queue1*. You can verify that the ports are correct by clicking Output Ports in the Ports panel, which is at the bottom of the General tab. If the ports are out of order, you can use the "Rank ^" and "Rank v" buttons to reorder the ports. Now we want to route to the appropriate port number based on a certain percentage.

- Click the Flow tab. Select By Percentage from the Send To Port list.



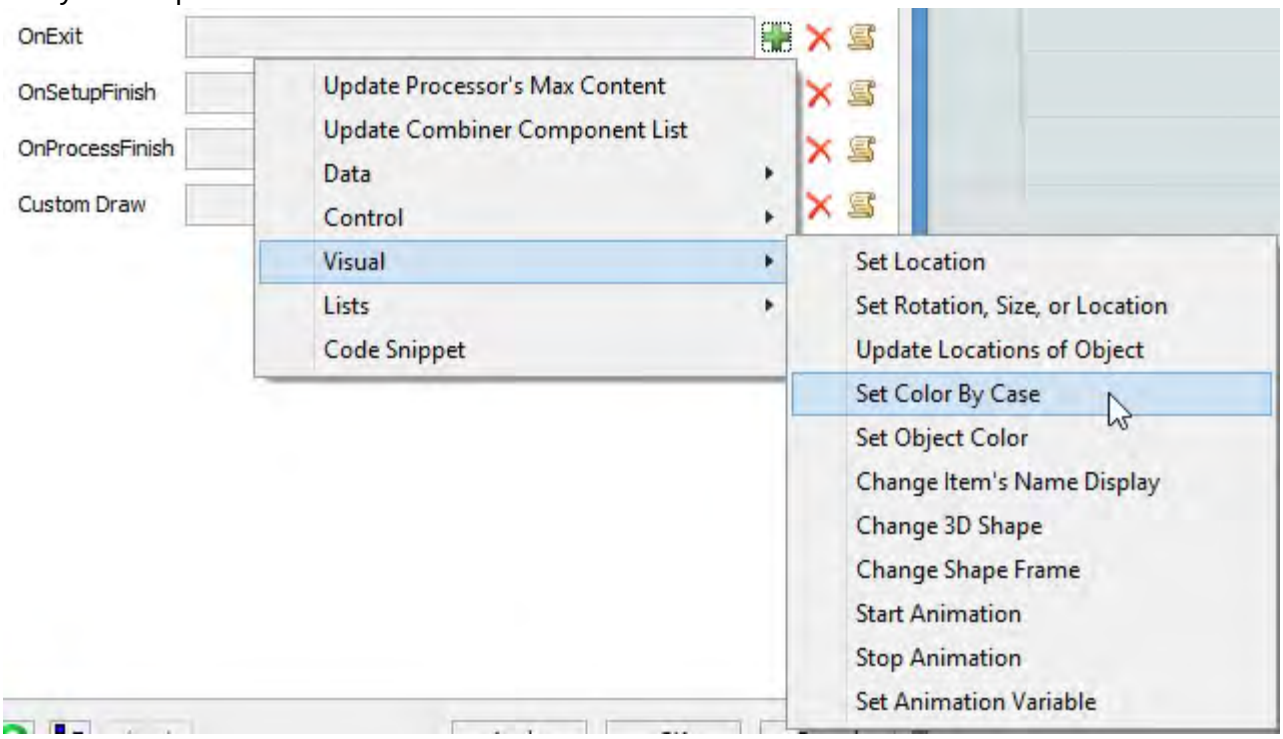
- Use the + to add another field.
- Fill the fields to match the picture below.

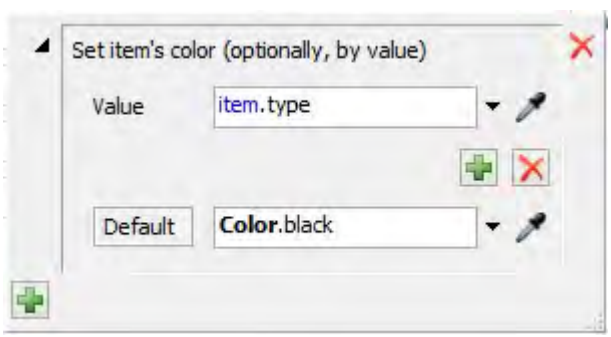


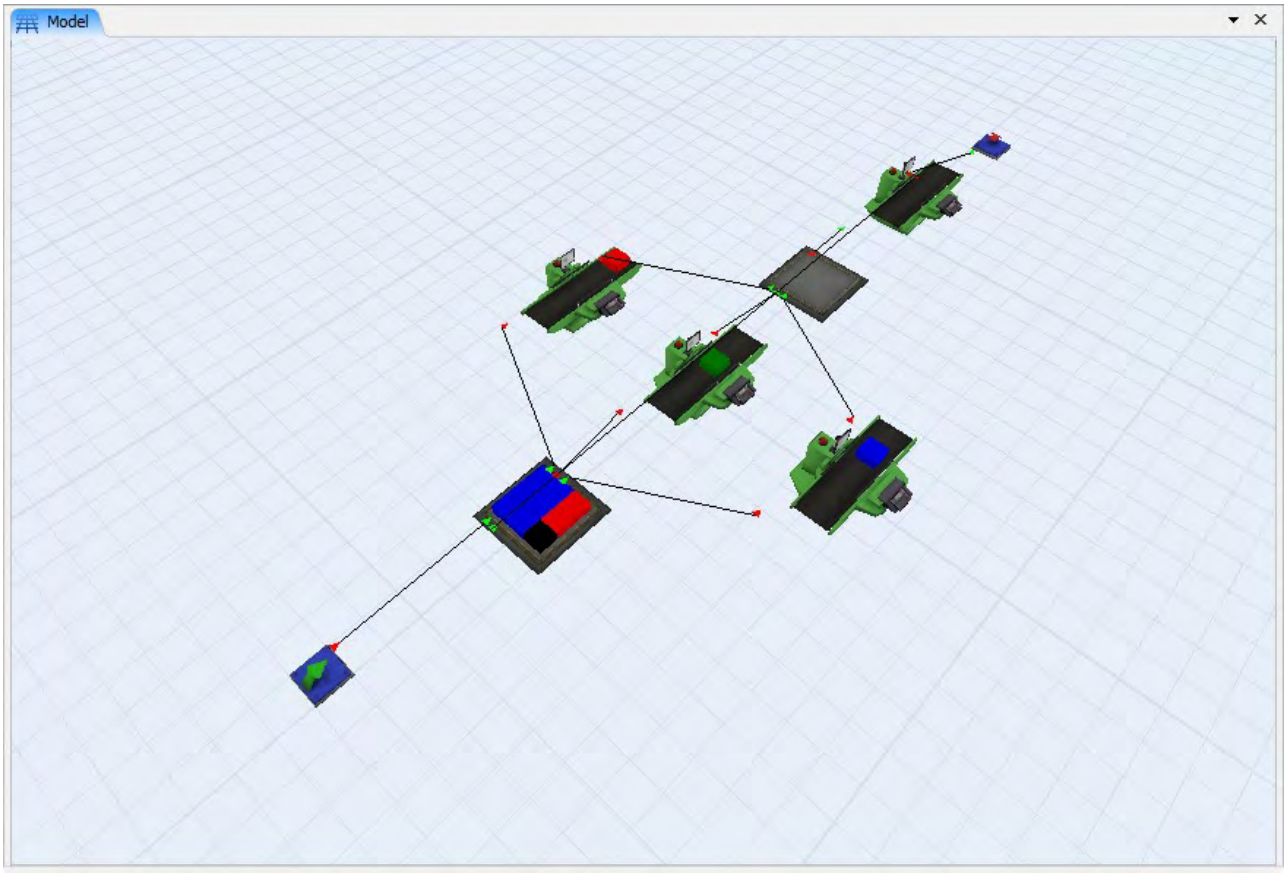
This means that 80 percent of the products (the correctly manufactured products) will be sent through output port 1 to the Sink, and 20 percent (the incorrectly manufactured products) will be sent through output port 2 back to the first queue.

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 the Triggers tab. Add a function (click the ✚button) to the OnExit trigger and select the Set Color By Case option. Select Black from the list.







To stop the model, press the Stop

- Press OK to close the Properties window.

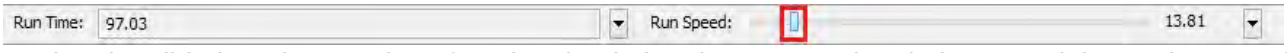
Step 13: Reset and Run the Model

- Click on the Reset button, located at the upper left-hand corner. Resetting the model sets all system variables to their starting values and clears any flowitems present in the model. Resetting is also necessary any time new connections are made between objects.
- Click the Run button, located right next to the reset button.

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.

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 Simulation time slide bar at the top of the window to the right or left. Alternatively, you can press the Ctrl + Down Arrow and the Ctrl + Up Arrow to increase or decrease the run speed.



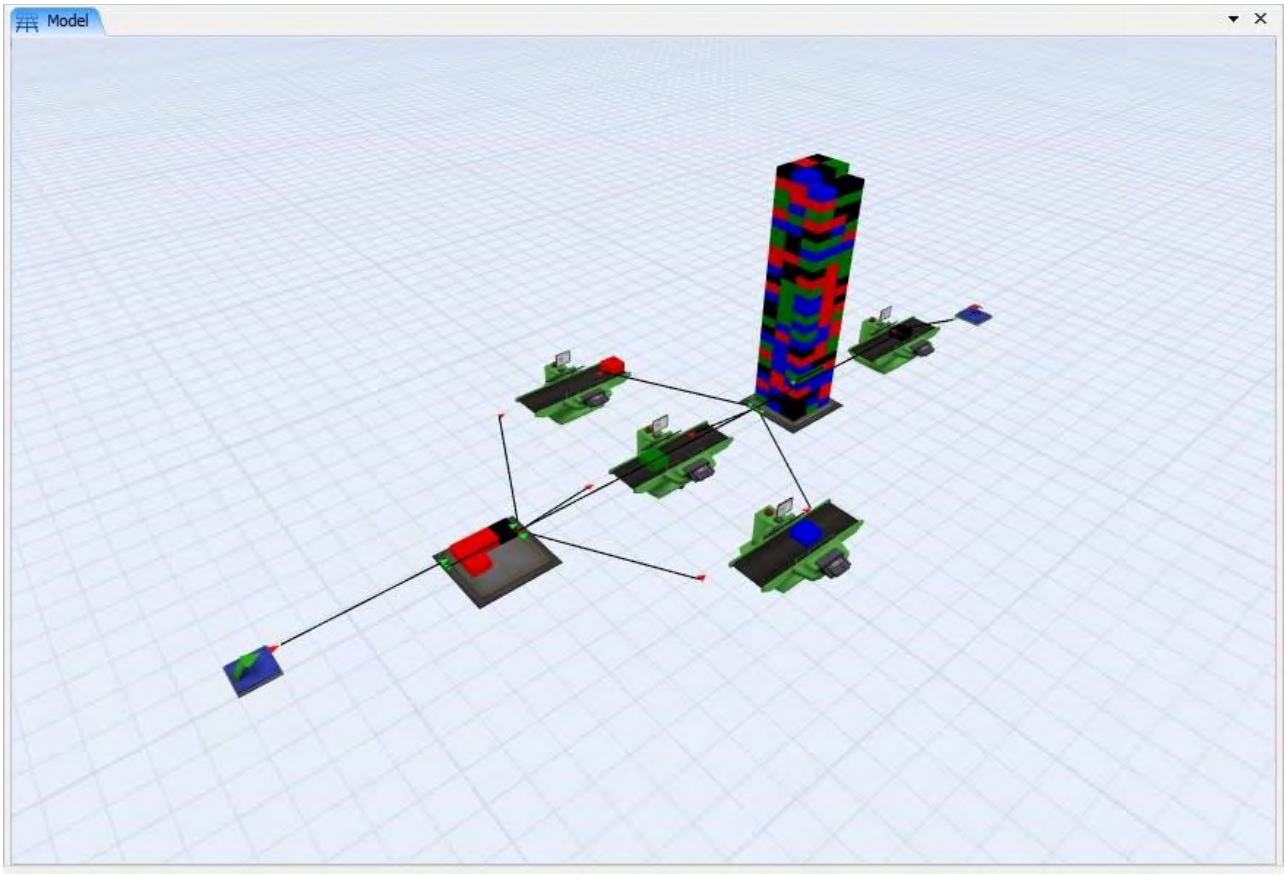
Moving the slide bar changes how fast the simulation time proceeds relative to real time. It has no effect on model results.

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

Creating a Dashboard

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.

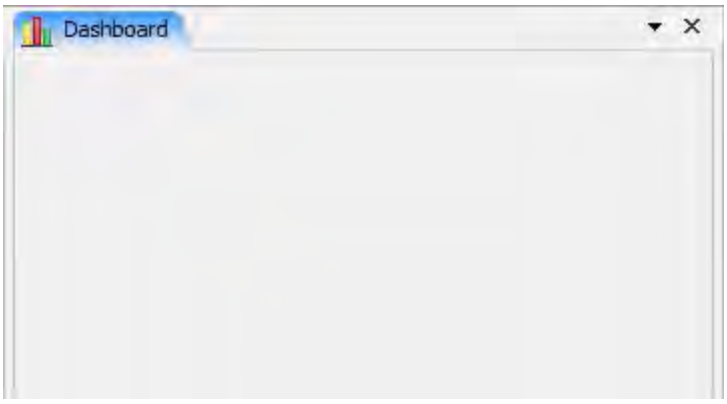


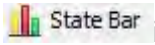
Another way of finding the location of a bottleneck is by examining the state statistics of each of the processors. If the three upstream processors are always busy, while the testing station is often idle, then the bottleneck is likely to be at the three upstream processors. On the other hand, if the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

Evaluating the New Configuration

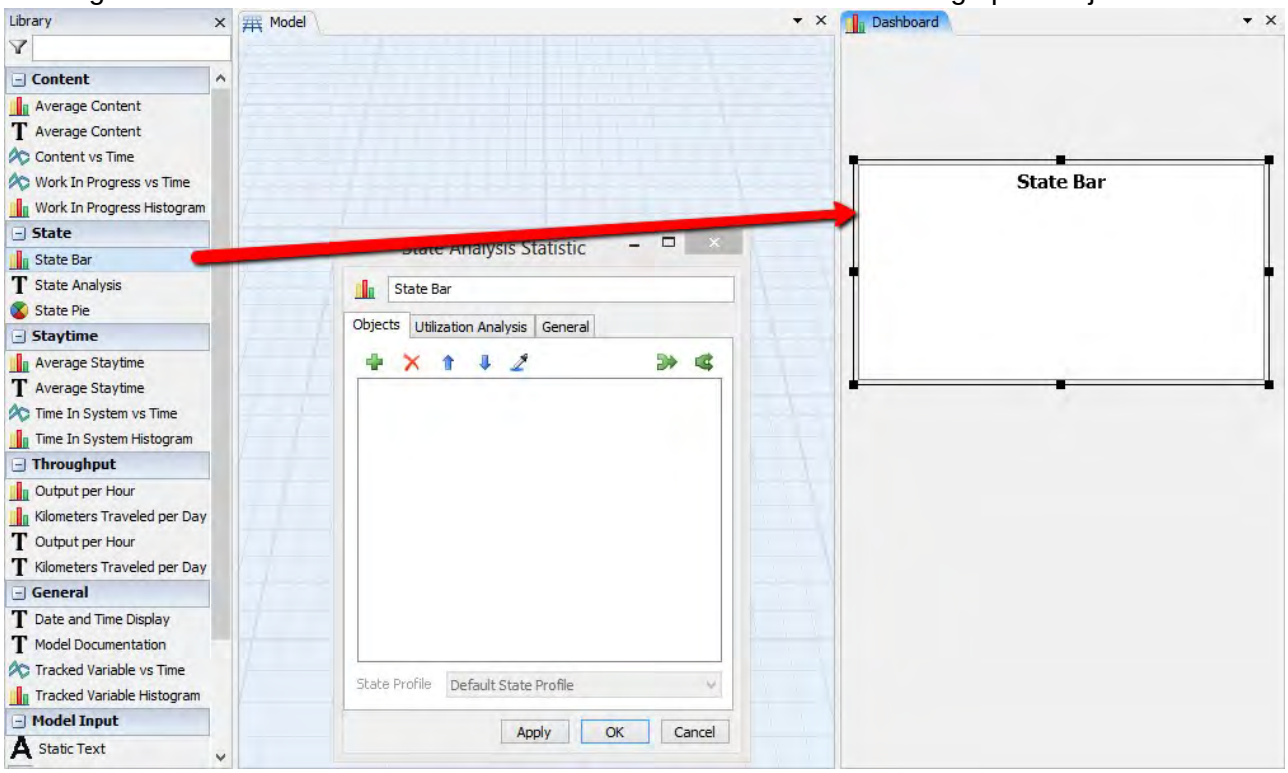
Run the model for at least 50,000 seconds. Notice first that *Queue2* is now almost always empty, whereas the Queue for the 3 processors backs up quite often. Let's use the dashboard to compare the two testers side by side.

- From the FlexSim Toolbar click the Dashboards button then Add a dashboard. The Dashboard window will appear.

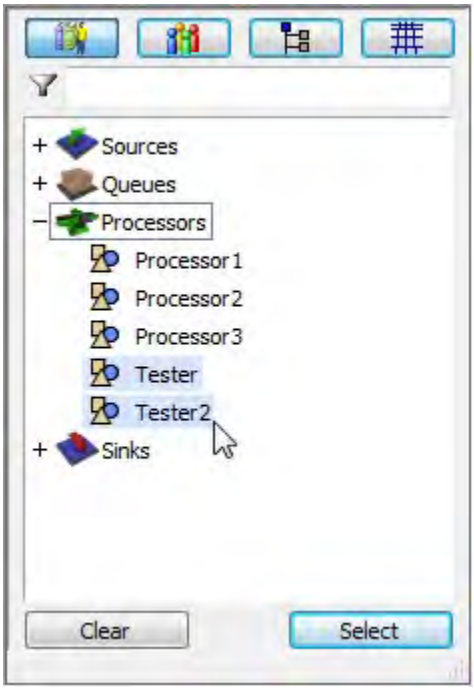


- Drag the 

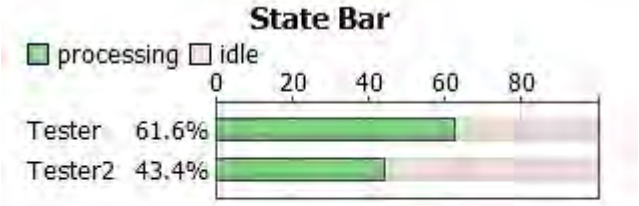
icon into the Dashboard window. This should bring up an object selection window.



- On the Objects tab, click the . This will open a popup.
- In the popup, click the  , expand Processors, and select *Tester* and *Tester2*.



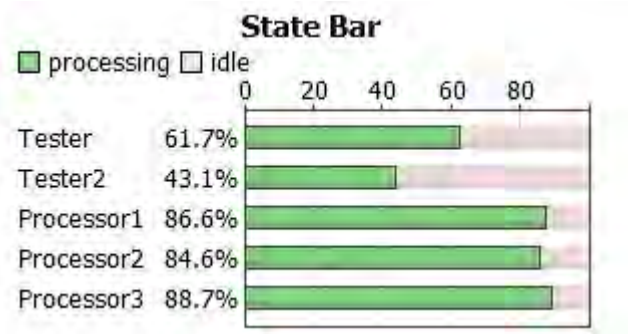
- Click the Select button on the popup to finalize your selection. Then click OK. A blank chart should appear in the dashboard.
- Reset and run the model again. The graph in the Dashboard will dynamically update.



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.

Now add the other three processors to the State Bar graph.

- Double-click on the graph in the dashboard and the same object selection dialog opens.
- Select *Processor1*, *Processor2*, and *Processor3* from the selection list. Whatever you select is added to the previous contents of the graph.
- Reset and run the model again. Now all five processors can be compared side by side.



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. 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.

Try changing any parameter (like a processor's process time) and watch its effect on the model. Even small changes can dramatically change the overall model.

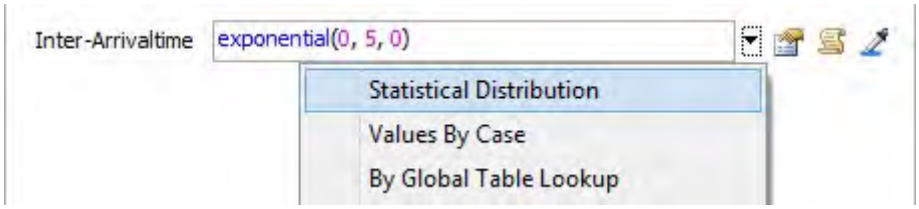
Randomness

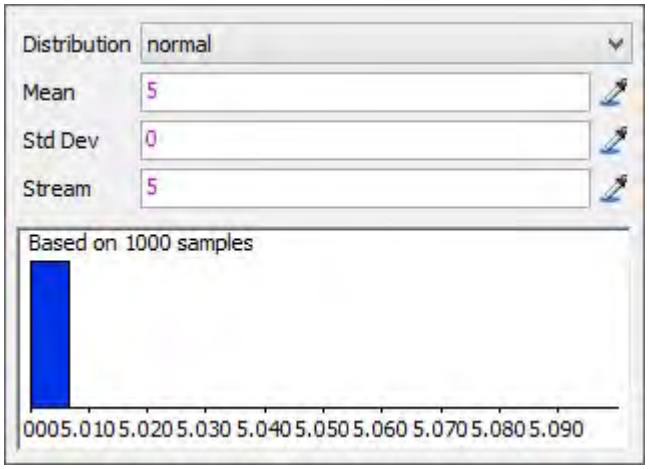
Let's do some more testing before we actually decide to add another tester. Since on average one product arrives from the source every 5 seconds, and on average one product goes to the sink every 5 seconds, why should the queue accumulate at all? Products are leaving just as fast as they arrive, so there 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.

- Edit the *Source's* Inter-Arrivaltime to match the following.





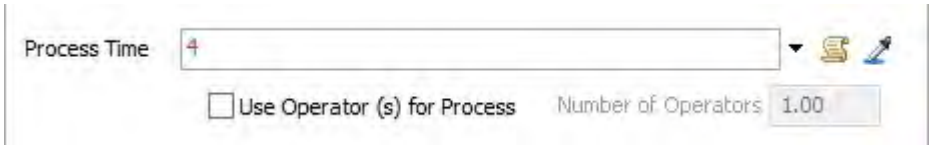
- Once set, Reset and Run the model again.

If you do not still have *Queue2*'s properties window available, open it again by double-clicking on *Queue2*. Continue to run the model. You will notice here that the queue's maximum content doesn'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 150 or 200. This is a significant difference caused by simply changing the type of randomness in the model.

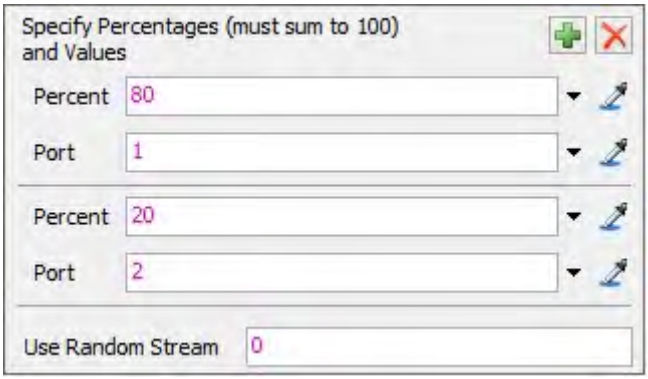
Higher Throughput

Now suppose that 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.

- Edit the *Source's* Inter-Arrivaltime to be a normal distribution with a mean of 4.25.
- Now we will create a second tester. Create another Processor object in the model, and place it below *Tester*. Name it *Tester2.*
- Connect *Queue2* to *Tester2*.
- Connect *Tester2* to *Sink* and to *Queue1*.
- Set *Tester2's* Process Time to 4.



- Change *Tester2's* Sent To Port to By Percentage Enter the same parameters as you did for *Tester1*.



- Add an OnExit trigger to change the color to black, just like the other Tester.
- Now that you have finished making the changes, Reset and Run the model again.

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 facility.

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 modelling. 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 3D world. This world is created automatically as you build your FlexSim models.

What Next?

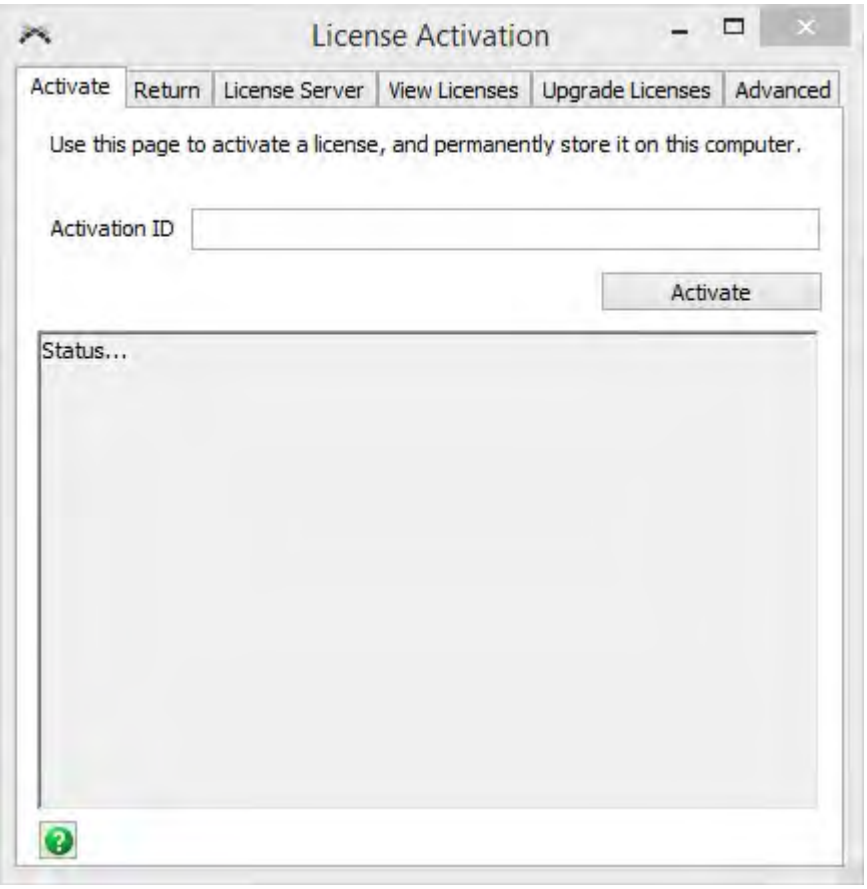
Now that you have become familiar with FlexSim and the use of simulation, we suggest that you go through the other tutorials included in FlexSim Help.

License Activation Concepts

The License Activation window is found in the main menu option Help > License Activation. From this window, you can activate standalone licenses, return licenses, repair licenses, configure the client to use a concurrent license server, view license rights held in Flexnet trusted storage, and upgrade standalone licenses.

Topics

- Licensing
- License Server
- Repairing Licenses
- View Licenses
- Upgrading Licenses
- Manual Activation and Return



Licensing

FlexSim does not require a license for it's trial version. The trial allows you to create 20 objects in your model and run that model with various scenarios.
If you would like to purchase a license for this software you may contact our sales department at (801) 224-6914 or email us at sales@flexsim.com

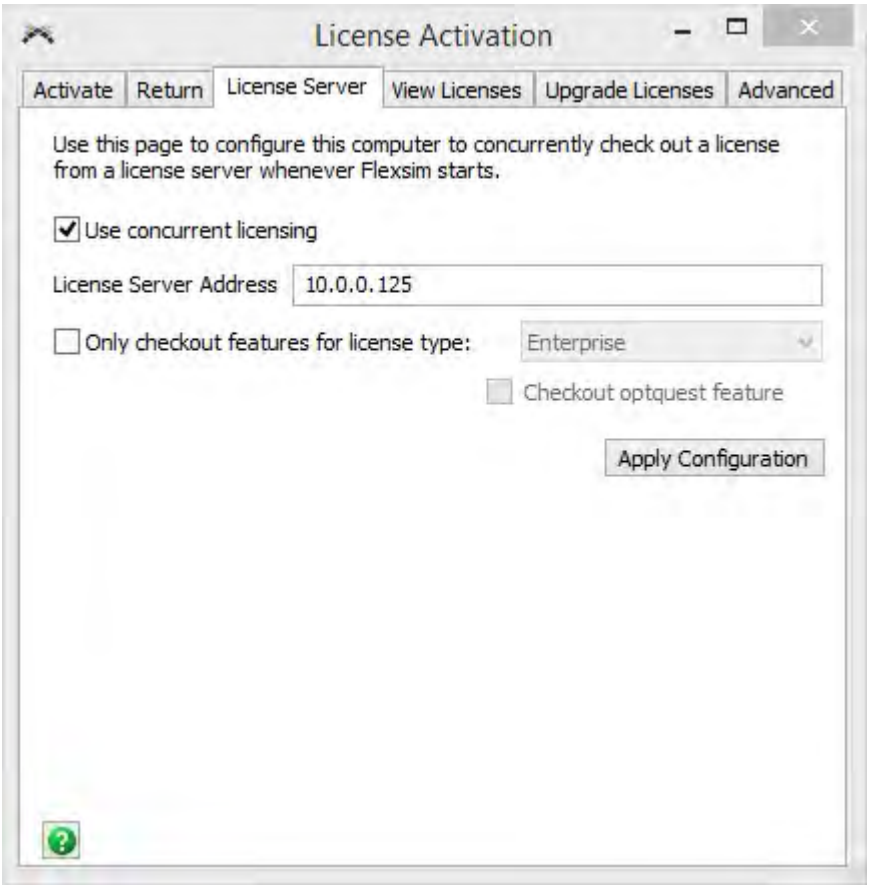
Once you have successfully activated a license from either your company's server or the FlexSim license server, many of the menu options that were grayed out in the demo mode will be available for use. You will also see your license type in the Help > About FlexSim window or the Start Page.

For an explanation of errors received while activating/returning licenses, visit the reference page.

Repairing Licenses

On the Advanced tab under the repair sub-tab, you will see any disabled licenses on your computer. Licenses are held in Flexnet trusted storage on a computer. Certain information about the licenses are stored in various places on your hard disk and registry. Tampering with these locations may break the license trust flags, which disables the license. This may happen with some registry cleaners or Windows restore points. If your license was disabled, then you can use the repair tab to generate an xml repair request file. Email that file to FlexSim support using the website (www.flexsim.com/support) and then you will be emailed back a repair response xml file to process on the repair tab. Processing the repair response from FlexSim's support team will repair the trust flags on your disabled license and allow it to be used again.

License Server



If you are using a concurrent server license, you can configure the clients to connect to the server using this tab. Check the box and enter the license server ip address. Press the Apply Configuration button. FlexSim will immediately try to connect to the server and check out license features.

Before attempting to connect the clients to a concurrent server, you must activate your licenses on the server and start a License Server Manager program on the server. Instructions and files necessary for configuring the server are available in the LAN License Tools download available in the account section of the FlexSim website if you have a concurrent server license on your FlexSim account.

The Windows registry entry for FLEXlm, our license manager, can store multiple locations where FlexSim should look for a license server or file. In previous versions of FlexSim, whenever you entered a new value in the Concurrent License Server Address and restarted FlexSim, it added more entries to the registry. If

you have successfully checked out a license from a server, FlexSim may still successfully check out a license from that server when you start FlexSim, even if the "Use concurrent licensing" box is unchecked. For reference, the registry key that holds a semicolon-delimited list of locations is HKEY_CURRENT_USER\Software\FLEXlm License Manager\FLEXSIM_LICENSE_FILE. If desired, you can manually edit the value of this registry key to remove locations that you don't want the license system to search any more.

If you have multiple types of licenses on your server, such as Enterprise and Runtime licenses, you can check the "Only checkout features for license type:" box and select the feature set that you want to check out. This will tell FlexSim to only try to check out the features required for that type of license instead of requesting every available feature. You can also use the "Checkout optquest feature" checkbox to specify whether the Optquest feature should be checked out.

For help with troubleshooting concurrent server errors, visit the reference page.

View Licenses

On the view licenses tab, you can see the contents of this computer's Flexnet trusted storage.

Licenses are not in any way tied to any FlexSim installation. Your computer itself is licensed. The actual FlexSim install or version doesn't matter at all. One license can license every FlexSim program on the computer. You simply use the FlexSim program to activate/return/view licenses on the computer for simplicity. The licenses are not tied to an installation of FlexSim.

For example, you could install FlexSim, activate a license, uninstall FlexSim, and then install FlexSim again, and your license will still be there. The FlexSim programs are entirely separate from where licenses are stored on your computer in Flexnet trusted storage.

It doesn't matter what version of FlexSim you use to activate or return licenses; they are calling exactly the same code.

You can have just one license of 5.1 on the computer and that license will work with both 5.0.4 and 5.1.0 just fine. You don't need multiple licenses to license multiple FlexSim programs. The license is not tied to the installation. The license is tied to the computer. Any FlexSim programs on a licensed computer will work. However, a computer that has a 5.0 license won't be able to run 5.1 software. The computer's license must be >= the version number of the software in order to run.

If you uninstall FlexSim without first returning your license, you will need to reinstall Flexnet in order to return the license so that you can move it to another computer. Because licenses are tied to the computer, not the FlexSim installation, you do not need to return your license if you are simply uninstalling and reinstalling FlexSim.

Upgrade Licenses

On the upgrade licenses tab, you can request upgrades to your licenses. This is necessary to run newer versions of the software. For instance, if you previously had FlexSim 5.0 installed and you upgraded to 5.1, you would need to also upgrade your license to 5.1.

The process to upgrade a license to a newer version has three steps:

1. The FlexSim License Server needs to be told to upgrade the license, which creates a new Activation ID for the upgrade.

2. The old license needs returned to the FlexSim License Server.
3. The new license needs to be activated on the client machine.

This process can be done manually by returning any old licenses, pressing the Upgrade Licenses button on the "My Licenses" page in the account section of the FlexSim website, and then activating the new Activation IDs on that page (you may need to refresh the licenses page table).

The Request Upgrades button on the Upgrade Licenses tab automates this process into a single button click by sending http requests to the FlexSim server. If this automatic process doesn't work, then you can do it manually instead.

To upgrade server licenses, you must manually return the licenses and then activate the new licenses.

Manual Activation and Return

In some cases you may not be able to activate or return licenses using the default activate or return tabs. This may be the case if you don't have an internet connection to the computer that is running FlexSim or firewall settings prevent the automatic activation from working. In this case you can use the options in the Advanced tab to manually activate or return licenses.

To do a manual activation or return you generate an xml request and send it to FlexSim where it will be processed and a response xml file will be sent back to you. This is what you can then manually process to complete the licensing.

- In the Manual Activation sub-tab you can enter your activation ID and generate an activation request xml file and process the response xml file.
- In the Manual Return sub-tab you can select a license and generate a return request xml file as well as process the response xml file.

License Activation Example

Topics

- Activating a Standard License
- Returning a License
- Repairing a License
- Activate a Concurrent License

Activating a Standard License

This will take you through the steps necessary to activate a standard FlexSim license through the FlexSim License Server. If your company is using a concurrent license server, see the Activate a Concurrent License section.

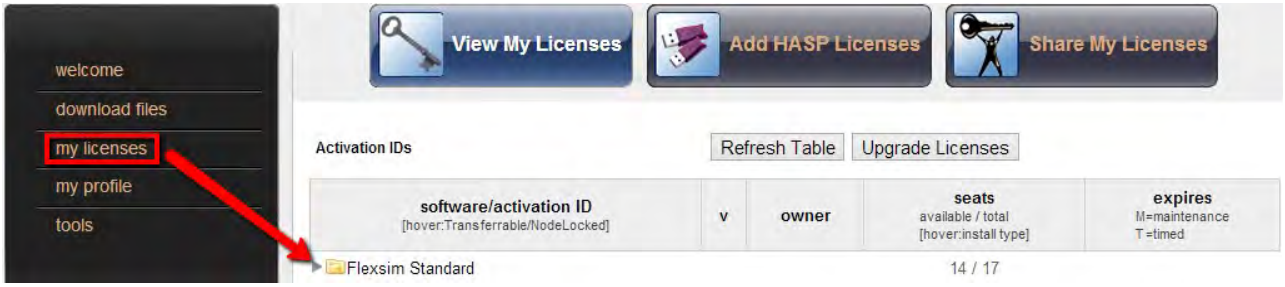
Login to your FlexSim.com account

First you'll need to retrieve an available license from your FlexSim account.

- Go to http://www.flexsim.com/account/ and login.

Find an available license

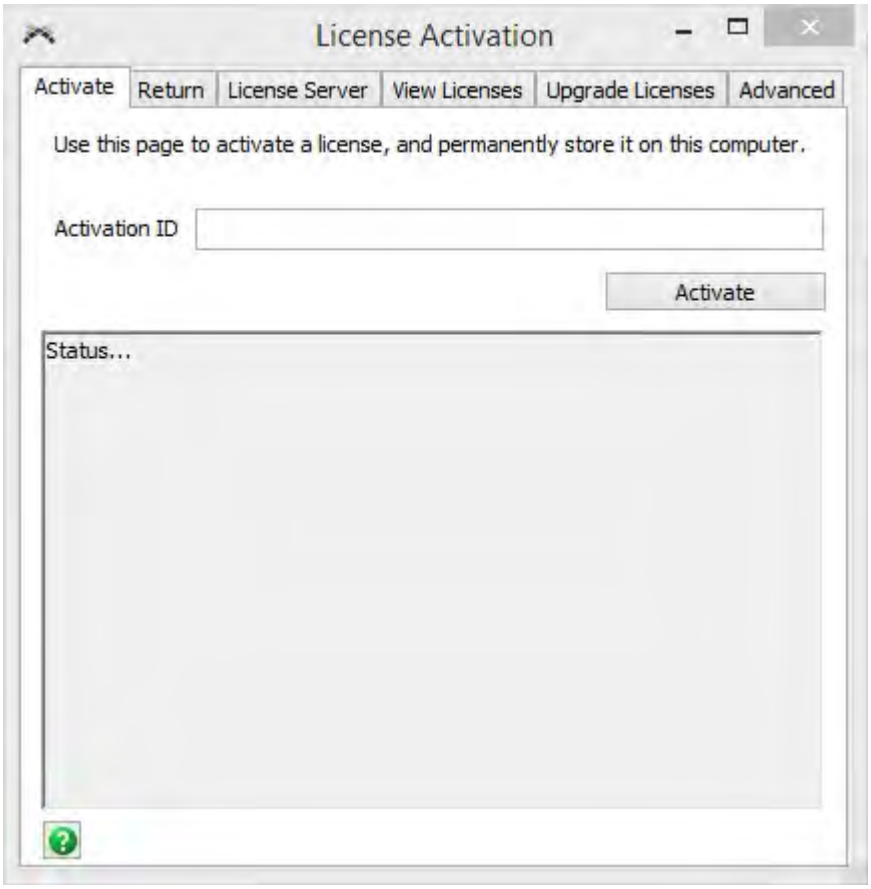
- Once logged in, go to the 'my licenses' page.
- Expand your Flexsim Standard folder to show your available licenses.



You can see the number of seats available to the right, as well as expiration dates for timed and maintenance licenses.

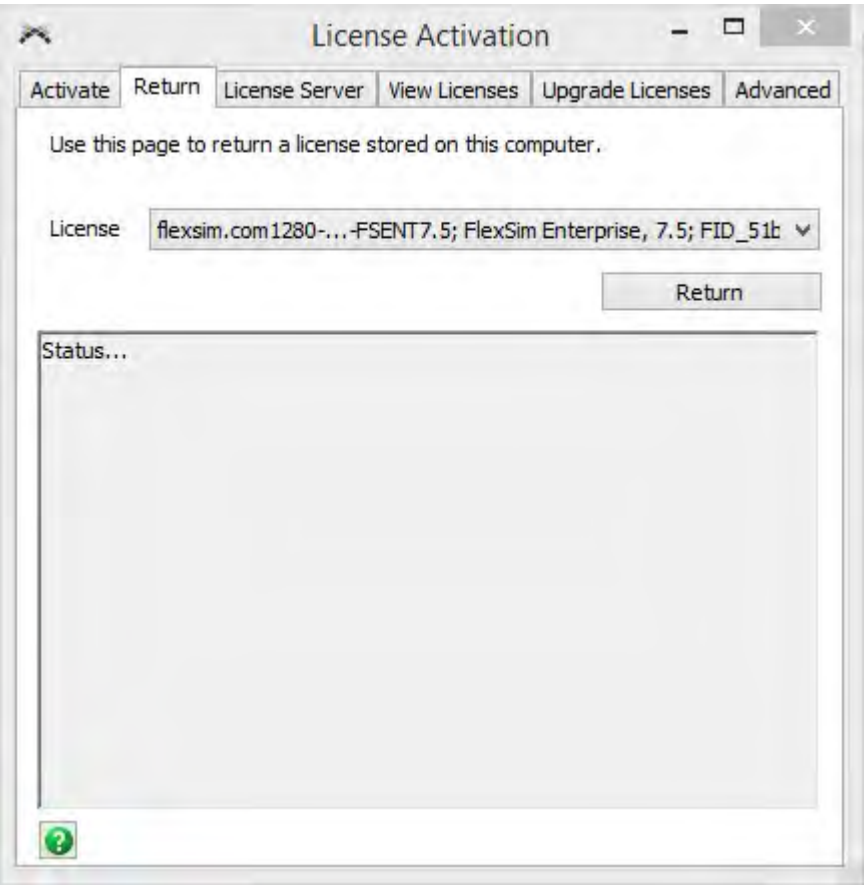
Activate the license

- Copy the Activation ID for an available license. The activation ID will start with flexsim.com.
- Paste the Activation ID into the Activation ID field.
- Press the Activate button.



For an explanation of errors received while activating licenses, visit the reference page.

Returning a License

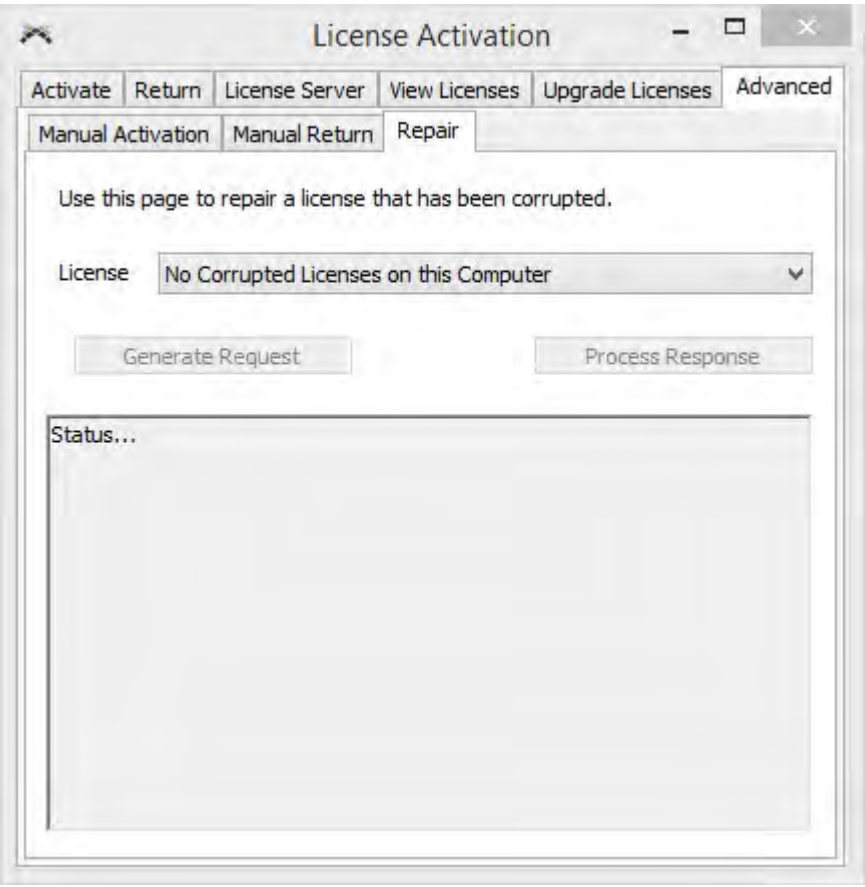


Returning a Standard License

- Select the license you want to return from the License dropdown menu.
- Press the Return button.

For an explanation of errors received while returning licenses, visit the reference page.

Repairing a License

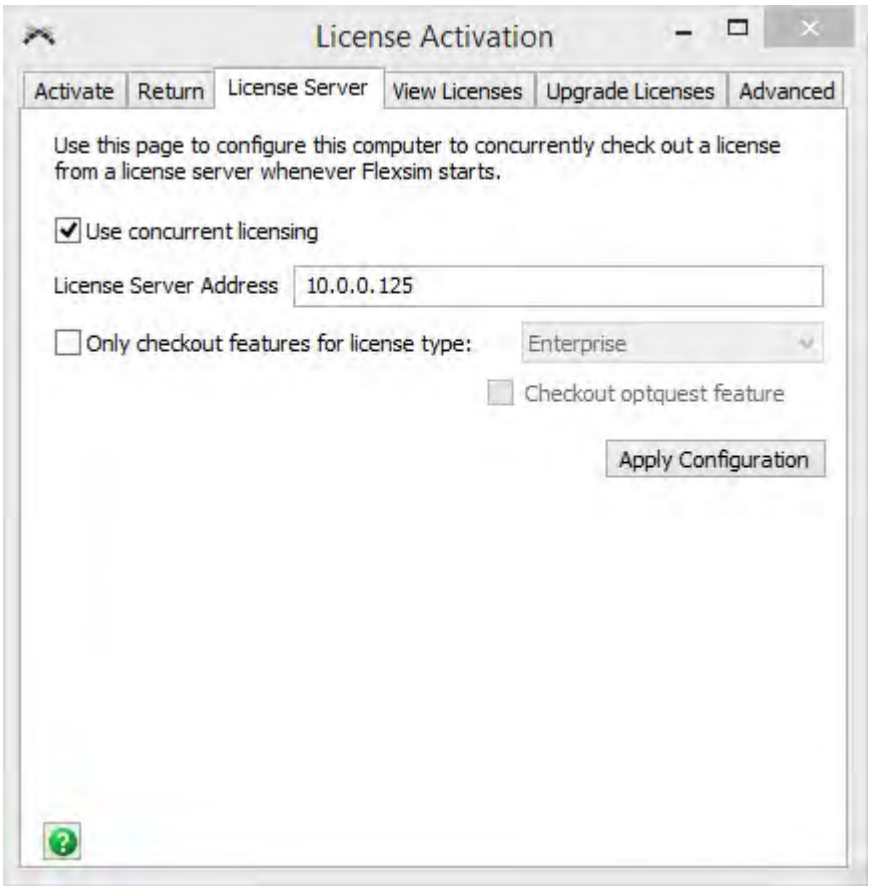


Repairing a Standard License

- Select the corrupted license from the License dropdown menu.
- Press the Generate Request... button.

- Email the request file to FlexSim support using the website (www.flexsim.com/support)
- Once the FlexSim support team sends you the XML response file, press the Process Response... button and select the response file.
- Press the Repair button.

Activate a Concurrent License



- Check the Use concurrent licensing button.
- Enter the ip address of your company's license server in the License Server Address field.

- Press the Save Configuration button.

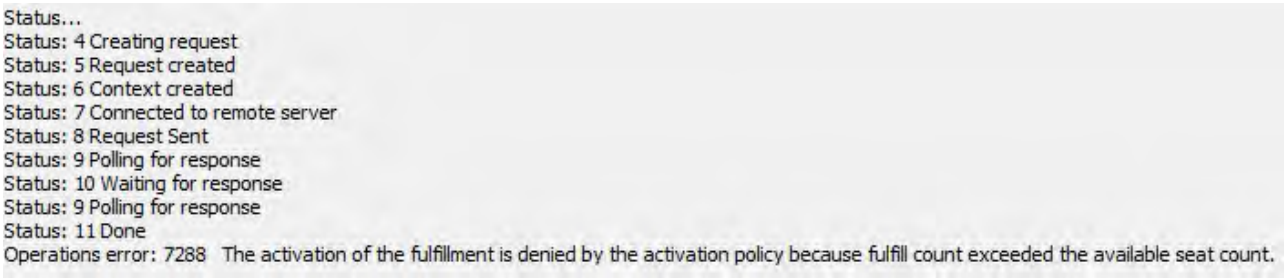
For more information on setting up a concurrent license server, see the documentation in the LAN License Tools.

Topics

- Common Activation Errors
- Common Return Errors
- Troubleshooting Concurrent License Server Errors

Common Activation Errors

Operations Error 7288



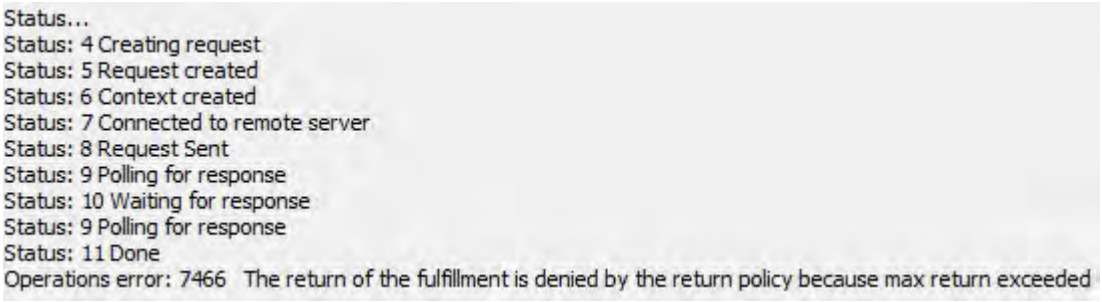
Operations error 7288 means that your license has already been successfully activated on a computer. In order to activate the license, you must find the computer that contains the license and return it. You can find the computer that contains the license by checking the output of View Licenses on any computers that you may have activated the license on.

Error 50041

Error 50041 usually means that your computer is not connected to the internet or your network/firewall settings are preventing communication with the FlexSim license server. The license activation mechanism uses soap requests sent through http port 80. Make sure this type of communication is available on your network.

Common Return Errors

Operations Error 7466



Operations error 7466 means that returns are disabled on your license. Your license is configured to be a one-time activation onto a computer. If you believe your license should be able to transfer from one computer to another, contact your local distributor or FlexSim support to discuss the situation.

Troubleshooting Concurrent Server Errors

1. Make sure the client computer is connected to the server on a local area network. Be sure you can ping the server at the address specified.
2. Make sure that the server has valid licenses using the menu option Tools > View License Rights in the flexsimserveractutil.exe program on the server. This program is contained in the LAN License Tools available for download on the FlexSim website.
3. Make sure that the server's License Server Manager program (lmadmin or lmtools) is properly configured and has licenses available for use.
4. Consult the Flexnet License Administration Guide for additional details on how Flexnet server licenses work. This manual can be found at http://www.globes.com/support/fnp_utilities_download.htm.
5. If you still are still having troubles, contact FlexSim Support. Be sure to send them the information that is printed when you press Tools > View License Rights in the flexsimserveractutil.exe program and also a screenshot of the Dashboard view in lmadmin (or the Status Enquiry output from lmtools).

FlexSim Express is a free version of FlexSim Simulation Software. It is limited in several key ways, but is still quite useful for small simulations, as a model viewer, or for evaluation purposes. Our fully featured Enterprise edition has no limitations. To learn more about FlexSim Enterprise and have an opportunity to try it out, please contact a FlexSim partner near you https://www.flexsim.com/contact/).

Topics

- Model Size Limitations
- Restricted Menu Options and Tools

Model Size Limitations

The main limitation of FlexSim Express is the number of objects you are allowed to add to a model.

- You can open and run models of any size (model viewer)
- You can create new models or add to existing models, up to a total of 30 3D objects and 35 Process Flow Activities.
- You can save any model under the size limitation for sharing or later use.

Restricted Menu Options and Tools

The following menu options are restricted in FlexSim Express:

- File > State Files (can't load or save state files - i.e. a model part way through a simulation run)
- View > Model Tree (tree view is not available)
- Build - every option is disabled except Build FlexScript (can't link to a compiler and compile your model)
- Statistics > Experimenter (no Experimenter or OptQuest Optimizer)
- Statistics > Repeat Random Streams (simulation will always repeat random streams)
- Statistics > ExpertFit (curve fitting software is not available)
- Debug > Script Console (no script window access)

The following Toolbox tools are restricted in FlexSim Express:

- User Events
- User Commands
- Global Variables
- Global Macros
- Custom GUIs
- Model Triggers (On Model Open, On Reset, On Run Start, On Run Stop)
- Tracked Variables
- Experimenter
- Visio Import
- Video Recorder
- Fly Path

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 within 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.

Introducing... FlexScript

Don't let the name fool you – FlexScript is a powerful yet easy-to-learn scripting language that will help breathe life into even the most complex simulation model. Unique to FlexSim's line of software solutions, FlexScript provides a straightforward approach to allow users to quickly customize triggers and parameters within simulation projects.

Throughout the Concepts sections we 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 we 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 we 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.

For more information on writing FlexScript code refer to the topic on writing logic in FlexSim.

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.

Flowitems are copied into the model from the Flowitem Bin. You can learn more about the Flowitem Bin in the Modeling Tools - Flowitem Bin section.

Flowitems only remain in until the model is reset, then all flowitems are destroyed.

In FlexSim, ports are used to connect objects together logically. Each object can have an unlimited number of ports. There are three types of ports: input, output, and central.

For Fixed Resources, input and output ports are used in the routing of flowitems. For example, a mail sorter places packages in one of several queues 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 Queue objects, meaning once the Processor (or mail sorter) has finished processing the flowitem (or package), it sends it to a specific queue 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.

See Fixed Resource Concepts for more information on how Fixed Resource input/output port hand-shaking works.

For Task Executers and Dispatchers, input/output ports are used to define task sequence dispatching. See the Dispatcher topic for more information on how this works.

Creating Ports

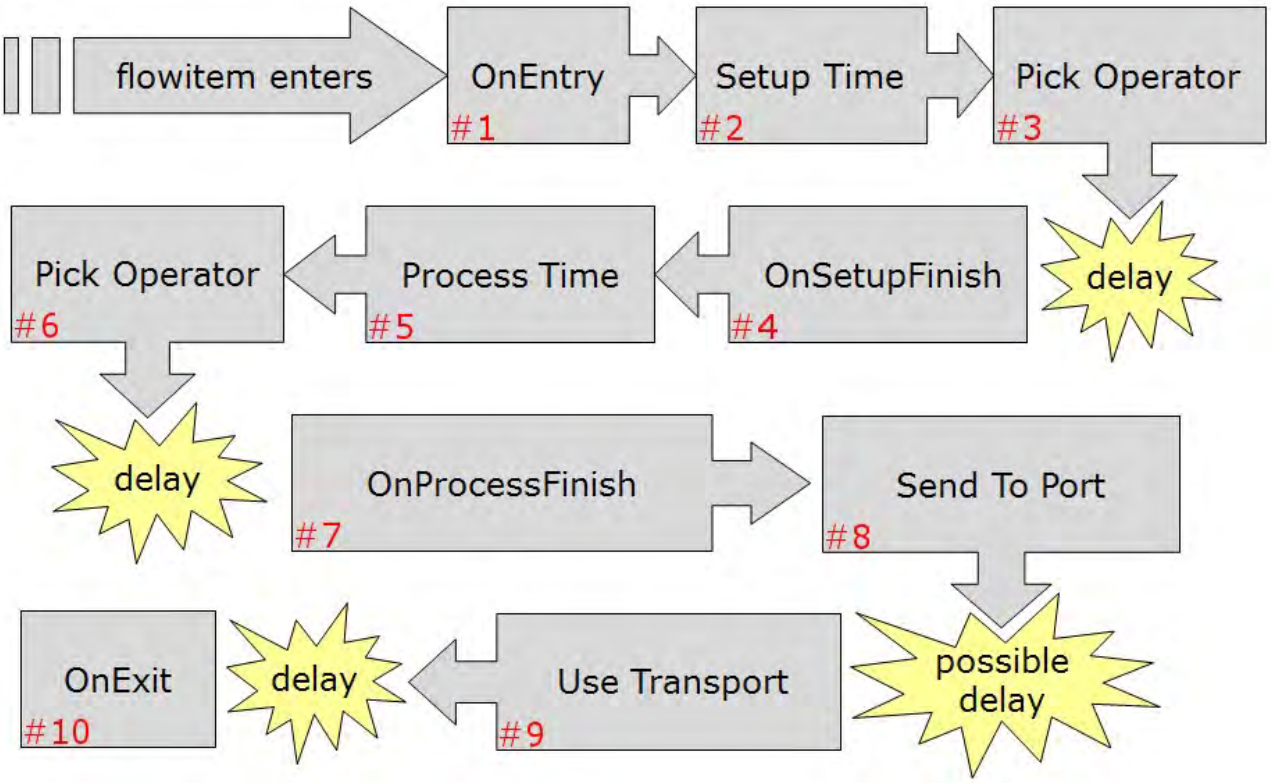
Ports are created and connected in one of two ways:

1) 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. The First Model from the Getting Started section demonstrates how to properly make port connections.

	Output - Input	Center
Connect	A	S
Disconnect	Q	W

2) By entering the connection mode via clicking the  button. Once in the connection mode, there are a couple of ways to make a connection between two objects. You can either click on one object, then click on another object; you may also click and drag from one object to the next as with method one. Either way, keep in mind that the flow direction of a connection is dependent on the order in which you make the connection. Flow goes from the first object to the second object. Incidentally, connections can be broken

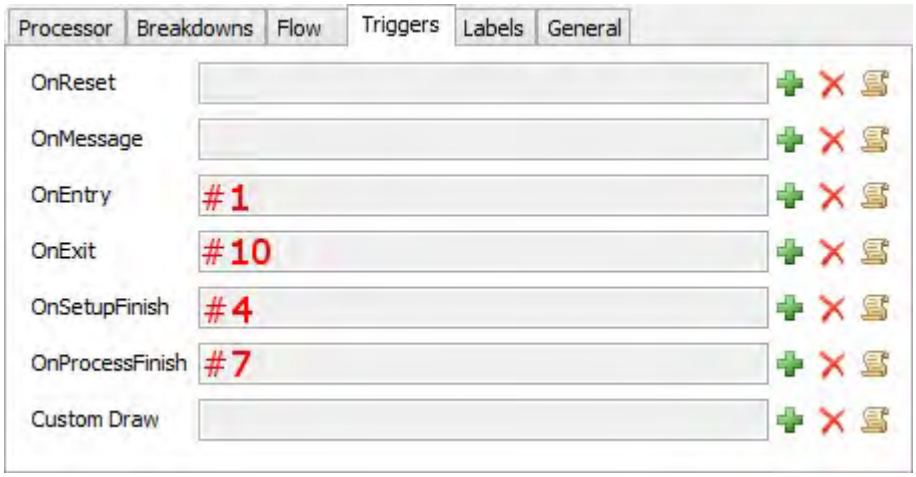
by clicking the button then clicking or dragging from one object to another in the same manner as when you connected them. Center port connections are not affected by the order in which the objects are connected.

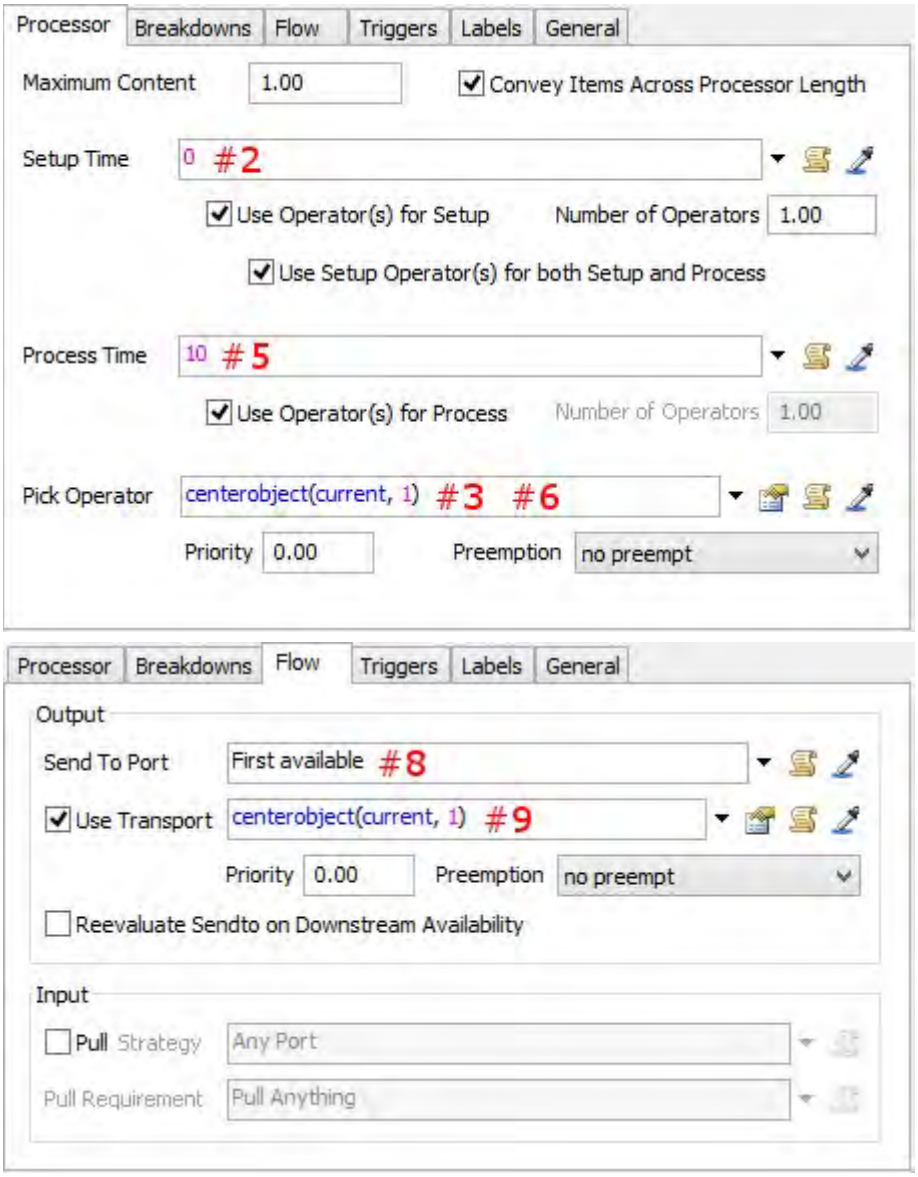


As flowitems move through your model, they enter and exit Fixed Resource objects. Each flowitem will follow a path similar to the one displayed above. This path is for a Processor object.

Example

Consider a flowitem entering a Processor. The following images show where each trigger and picklist can be accessed through the Processor's Properties window. Some of these can also be accessed through the Quick Properties.


Processor
Breakdowns
Flow
Triggers
Labels
General
OnReset
OnMessage
OnEntry
#1
OnExit
#10
OnSetupFinish
#4
OnProcessFinish
#7
Custom Draw



Step 1: OnEntry

When the Entry Trigger fires, the flowitem has already been moved into the Processor object, so the content of the Processor increases by one. Code can be added to the OnEntry trigger to customize what occurs when the flowitem enters.

Step 2: Setup Time

The next segment of code to fire is the Setup Time. This picklist gives you the option to define the number of model time units it takes for the flowitem to be "setup" in the Processor.

Step 3: Pick Operator

If the "Use Operator(s) for Setup box is checked, the Pick Operator code will fire, calling to a TaskExecuter object. There is a delay associated with calling an operator as the operator must move from their current

location to the Processor. The delay may lengthen if there are no available operators when the Pick Operator code is fired.

Step 4: OnSetupFinish

Once setup is complete, the Setup Finish Trigger will fire.

Step 5: Process Time

Similar to the Setup Time, this picklist allows you to define a Process Time for the flowitem.

Step 6: Pick Operator

If the "Use Operator(s) for Process" is checked, the Pick Operator will fire. If operators were used for the Setup and Process, and "Use Setup Operator(s) for both Setup and Process" is unchecked, then this picklist will fire twice, once to call a setup operator and then again to call a process operator. Otherwise, this picklist will only be called once.

Step 7: OnProcessFinish

Once setup is complete, the Process Finish Trigger will fire.

Step 8: Send to Port

At this point, the flowitem is ready to leave the Processor, or FixedResource object. The Send To Port will fire and attempt to find a destination for the flowitem. The possible delay happens if the object is unable to find an available destination for the flowitem. This typically happens when downstream objects are full and have no available space to accept the flowitem. It may also be because downstream FixedResource objects are stopped, or their inputs are closed.

Step 9: Use Transport

If "Use Transport" is checked, and once a destination has been founding using the Send to Port, the Use Transport picklist with fire to call a Task Executer to the Processor to transport the item from the Processor to its destination. Again, this will result in a delay as an available operator is found and moves to the Processor.

Step 10: OnExit

Finally, the Exit Trigger fires. When this trigger fires, the flowitem has not yet exited the Processor, so the content of the processor remains unchanged. Once this trigger is complete, the flowitem will be moved from the processor (decreasing its content by 1) and into the Task Executer or downstream Fixed Resource object. If you need code to be executed when an item leaves a FixedResource, but it needs to be fired immediately following the item actually be moved from the object, call senddelayedmessage() with a delay time of 0.

Labels

Labels are also a key concept to understand in building models in FlexSim. Labels store data on objects that can be used in making decisions in the model.

- Each label has a name that is defined by you the modeler.
- Labels can be defined on objects as well as flowitems (e.g. Sources, Queues or Processors).
- An object can have as many labels as you choose to give it.
- Labels can have number, string, pointer or array values. Labels can even hold lists or tables of values.
- You can dynamically add labels to an object as the model runs or you can add the labels to the object through it's Labels page and assign a reset value to it.
- 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.

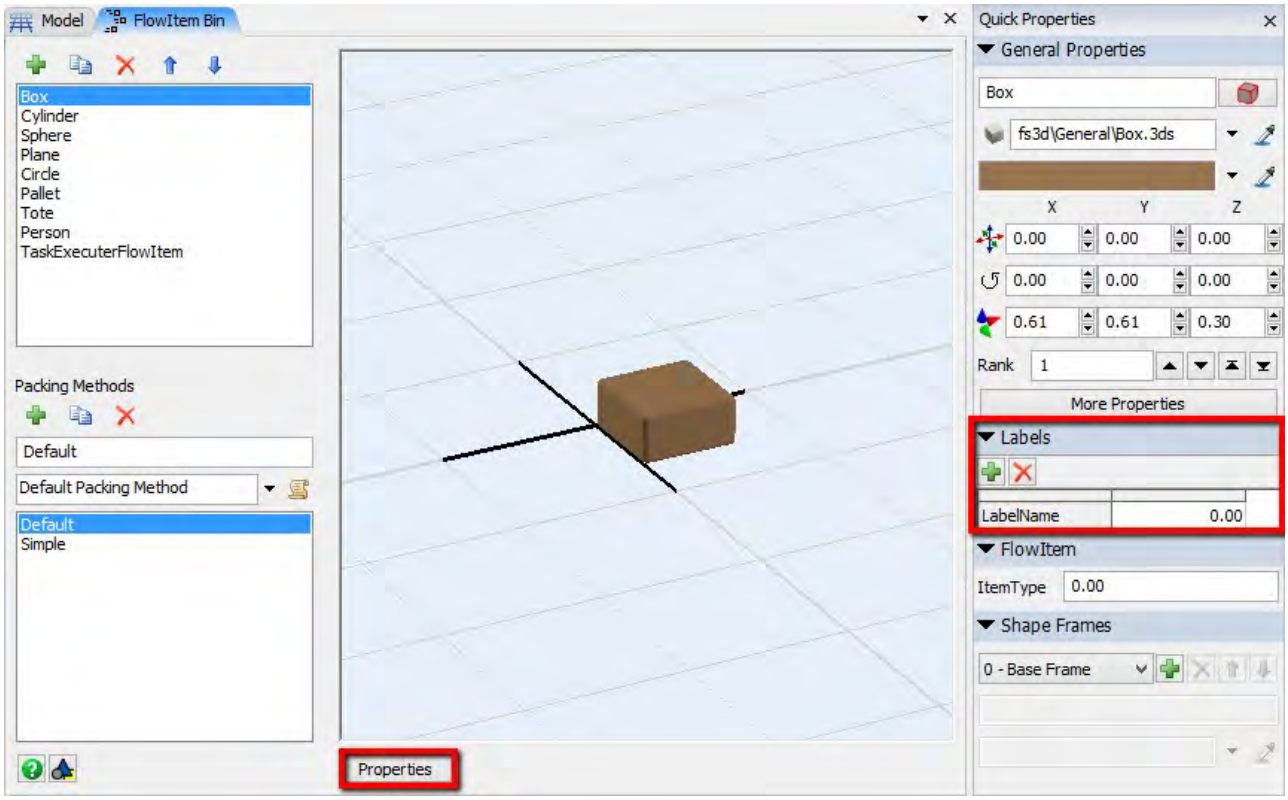
Values

For flowitems, the value you specify for labels will be the default 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 either have logic within the object that changes the label's value.

Label values will not reset on their own unless you check the Automatically Reset Labels button in the label tab page. Alternatively, you can add code or a picklist option to the object's OnReset trigger to reset the label value. Both options will set the label back to an initial value when you press the Reset button to reset a model.

Example

You can get to the FlowItem Bin either through the User Toolbar  or through the Toolbox. To add labels to flowitems, go to the FlowItem Bin, select the flowitem class that is being created by your Source (Box in this example), and edit the labels from the Quick Properties view. Alternatively, you can press the properties button (our double click the box) to display the Flowitems properties and edit the labels tab. You can add string or number labels to your objects.



The process works similar for other FlexSim objects. You can edit their labels from the Quick Properties window, or double-click on the object and edit and go to the labels tab. Specify each label's name in the row headers column on the left, and its value to the right of its name.

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 item type 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 and click on Properties. Here we would add a number label ("Add Number Label") and give it a name like "numOfCopies". 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:

```
item.type = bernoulli(60,1,2); if(item.type == 2)
        item.numOfCopies = duniform(1,1000);
```

As described previously, the .type command sets a label called type on the item to a 60/40 split between 1 and 2. Now we add an "if statement". This if statement basically says: if the type of the exiting flowitem is 2 (it is a copy customer), then set the value of the flowitem's label named "numOfCopies" to a random number between 1 and 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,

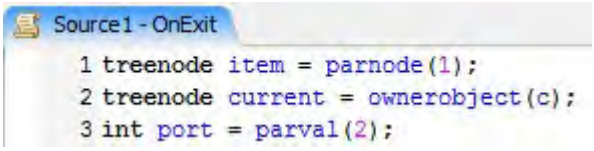
as opposed to the uniform() command, which may return values like 1.5 or 2.5. Since there will never be a customer that needs 1.5 copies made, we use the duniform() command.

Note that by adding the "numOfCopies" label in the Flowitem Bin, every flowitem that is created from that flowitem class will have that "numOfCopies" 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(item.type == 1)      return 3;
else return 5 + (item.numOfCopies)*(5.0/60.0));
```

As before we use an if statement to give type 1 (package customers) a service time of 3 minutes. In the else portion (copy customers), though, we return the expression: 5 + (item.numOfCopies)*(5.0/60.0)). This is the base service time of 5 minutes plus the number of copies that the customer needs item.numOfCopies 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 crosscompatible 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. We can use labels to store data on flowitems (or objects), and then we can access that data to make decisions in the model.



The terms item and current are two access variables that refer to objects in FlexSim. When you edit the code of a given trigger or picklist, 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:

Object item = param(1);

Object current = ownerobject(c); int port =

param(2);

For more information on port, see the Ports page.

Example

In this example, the first statement is what we call a variable declaration. For example, the second line declares a variable called current. The variable type of current is an Object. 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 Object or 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 Object variable 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 an Object variable, called item this time, that is given the value: param(1). Again, in order not to get side-tracked, I'm not going to explain the param 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: item.type = bernoulli(60,1,2); or:

if(item.type == 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:

```
Object current = ownerobject(c);
Object item = param(1); int port =
param(2);
```

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

```
Object 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 picklist section.

Review

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 (e.g. 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 *item.numOfCopies*. We do not use *current.numOfCopies*. 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 label 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 label command.

FlexSim allows the user to select appropriate units for a model. By default the Model Units window will appear for each new model.

You can select Time Units, Length Units, Fluid Units and a Model Start Time. The units you choose will be used throughout the model. The Model Start Time may be changed after the model is created, however, the Time, Length and Fluid Units CANNOT be changed.

You can also define what you want a new model's default model units to be through Global Preferences

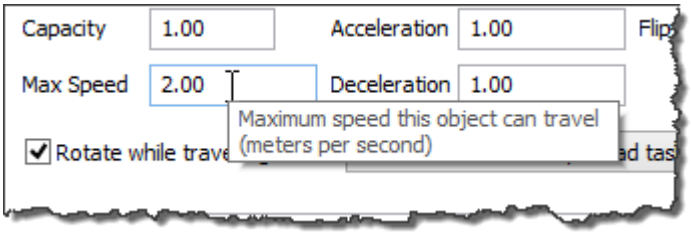
Entering Unit-Based Values

There are many fields in FlexSim that require unit-based values. In some cases the user interface will allow you to define what units the value you enter represents. However, if the field does not include units, you should enter the value in simulation model units. For example, if you've defined your model units in meters and seconds and a field requires a speed, if the user interface doesn't let you enter units, you should define the field's value in meters per second.

Use Simulation Model Units

If the user interface does not allow you to enter units explicitly, values should be entered in simulation model units, i.e. the units you defined when you created the model.

The user interface will often provide a hover tip that tells you what units you need to use, as shown in the following image.


Capacity
1.00
Acceleration
1.00
Max Speed
2.00
Deceleration
1.00
Maximum speed this object can travel
(meters per second)
Rotate while trave

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 modeller 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.

Example

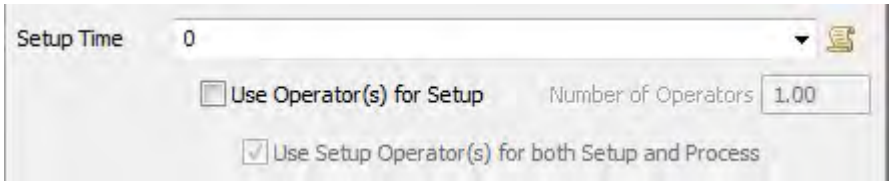
Let's refer back to the post office example mentioned previously. In that example we implemented the process time code as:

```
if (getitemtype(item) == 1)    return
3; else
  return 5 + item.numOfCopies * (5.0 / 60.0));
```

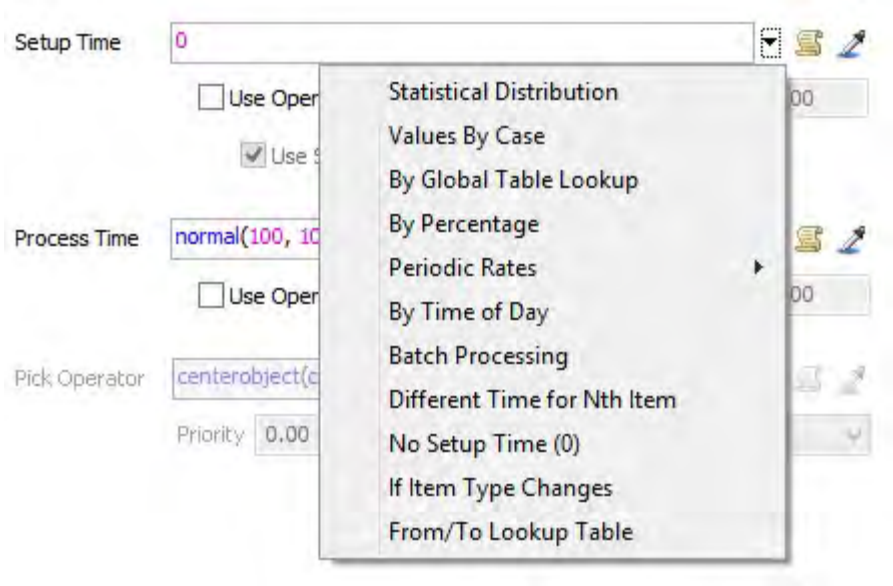
The Processor will execute this field each time it receives a flowitem, just before it starts the process time. By executing this process time field, it is essentially asking you what the process time for that item should be. In this code we are using the return statement to pass back to the Processor 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.

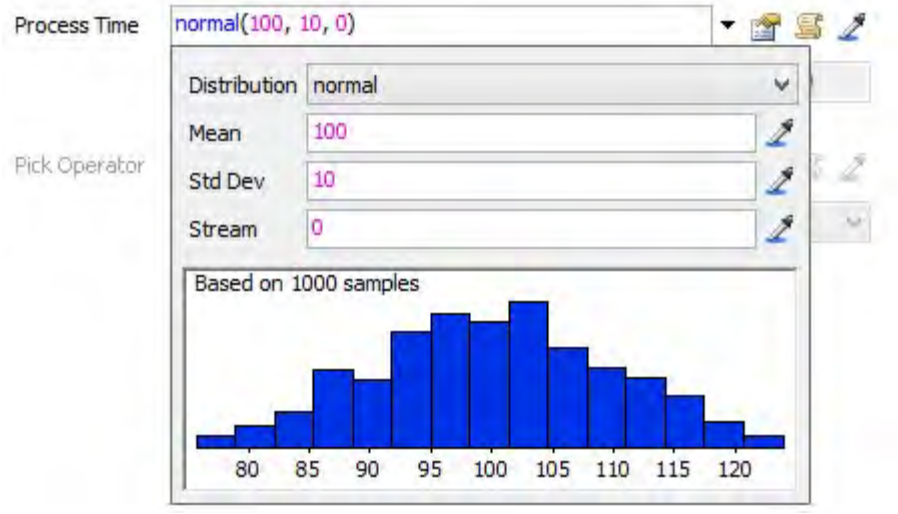


You will find pick list windows throughout FlexSim. These windows give you an easy interface for implementing functionality in FlexSim. Behind the scenes, each of these windows refers to a piece of code. The advantage about these pick list interfaces is that they allow you to write behavior without writing code. You just pick a behavior from a list. You can also select the text and enter an expression manually.



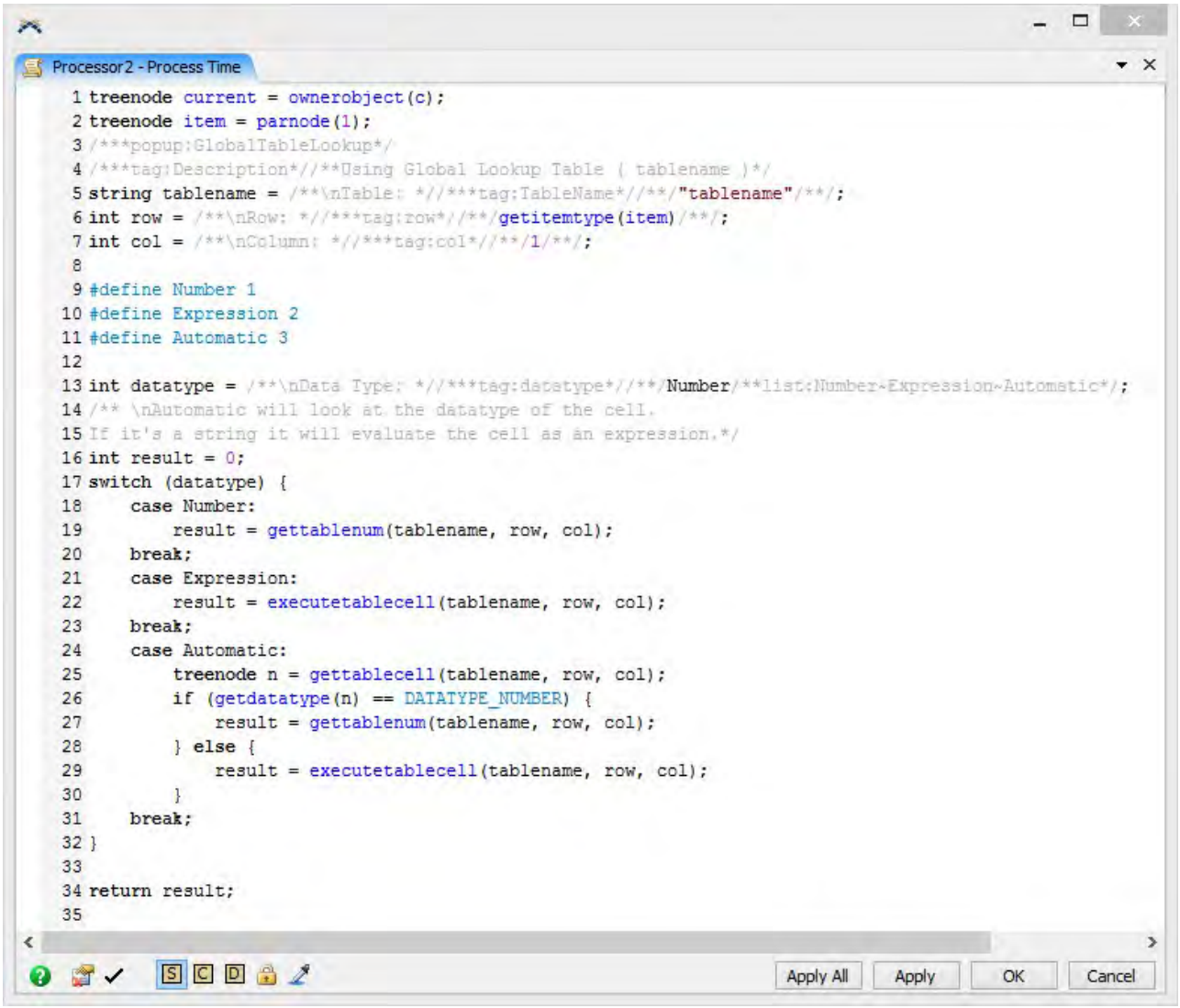
Popups

Each pick list option has an associated p opup. Popups allow you to easily edit the option parameters. You can edit these options at any time by clicking the button. Once you have entered the values you want in the popup, click anywhere outside the popup to close it.



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 (see the Template Code page).



To learn more about the Code Editor, see the Code Editor page.

For more information on how to write code in FlexSim, refer to writing logic in FlexSim.

In the code edit fields you may 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/**/; return value;

/** \n\nNote: The expression may be a constant value or the result of a

command (getitemtype(), getlabel(), etc).*/

The gray text is called template code. Template code is used in picklist popups as talked about in the Picklists page. When the button is pressed, FlexSim parses the template code and displays parameters in the correct places within the popup.

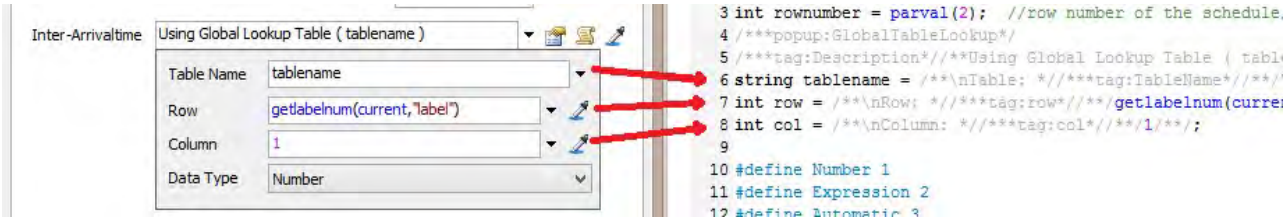
/***popup:ValuesByCase:hasitem=1:valuestr=Port:doreturn=1*/

This template code is an instruction to open a popup. int case_val =

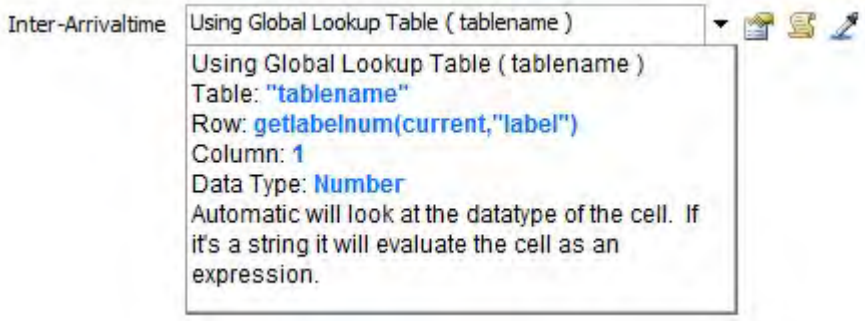
/***tag:ValueFunc*//**/getitemtype(item)/**/;

When the popup opens, it parses the script for instances of /***tag:TagName*//**/Value/**/ and places Value in the appropriate field in the popup. When the popup closes, the values in each field take the place of their respective Value. In this way, the /***tag: mechanism creates a two-way link between the script and the popup.

Here's a screenshot from from Using Global Lookup Table from the Inter-Arrivaltime drop-down menu in the Properties window of a Source object:



Template Text Popups



If no popup exists for the template code, or if a user creates custom template code, FlexSim will display a template text popup. The above popup is editing the same values as the Popup displayed above. Template Text consists of black and blue text. Blue text is editable text, just like you would edit in an edit field of a normal popup. 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 at the top of this section, the text: /**By Expression*/ makes it so the text "By Expression" will show up in black text in the template drop-down.

To specify the 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 modeller) 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.

Comments

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

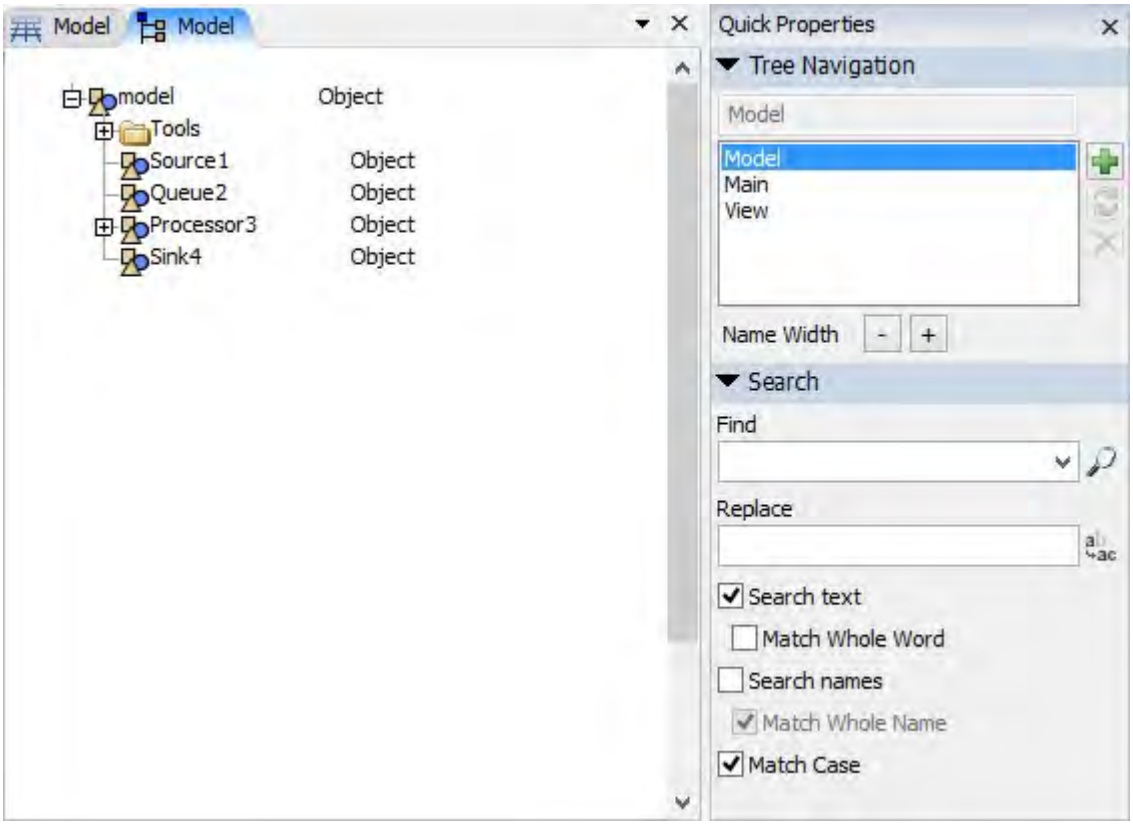
```
// This is my one-line comment, it ends at the end of this line
```

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
*/
```

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 Tree button from the toolbar. The Tree Window will then appear and the Quick Properties window will change to display the Tree Navigation and Search sections.

Note: If you are using the Evaluation version of FlexSim, you will not be able to use the 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 windows
- Edit the model, delete objects, and modify all data

If you follow a few simple navigation rules you will find the tree view to be 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 as well as all the currently open windows. When a session is saved the MAIN tree and the VIEW tree are saved together.

The folder icon identifies a node that does not have any object data, but may contain other folders or objects.

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 used to hold executable code. There are four types of code nodes in FlexSim:

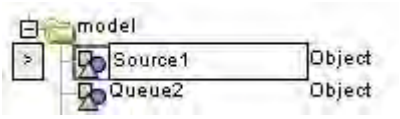
C++. C++ code must be compiled before running the model.

Flexscipt. This code will be auto-compiled during the running of the model. Read more details on writing logic in FlexSim.

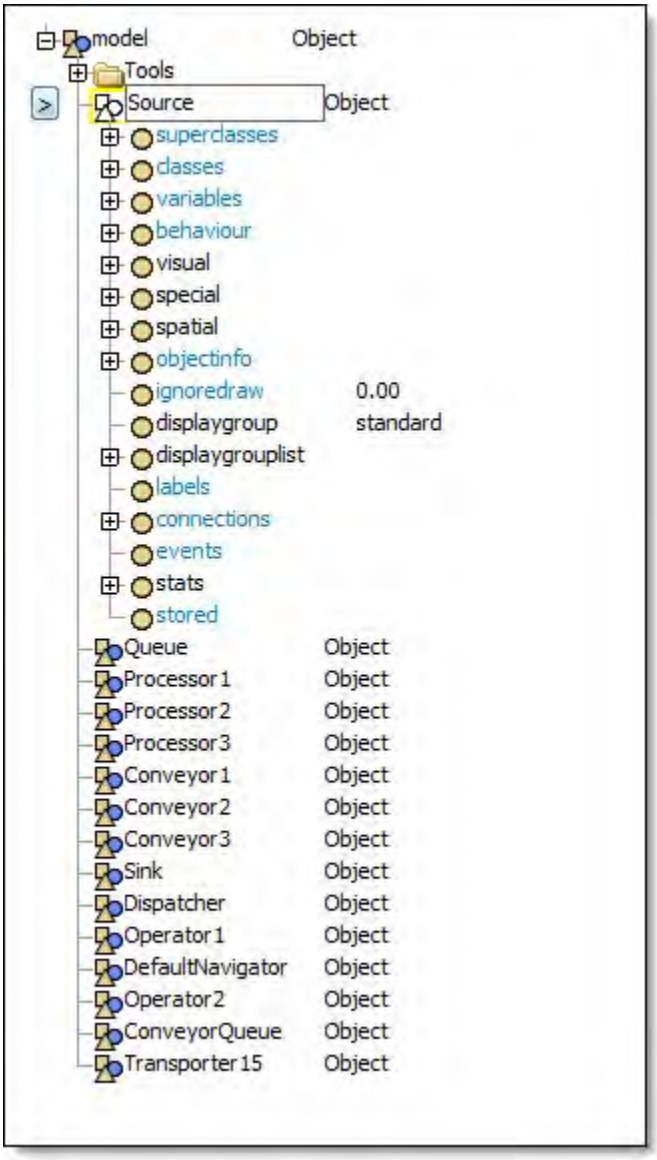
Dll. This node refers to a FlexSim compatible pre-compiled dll. To create such a dll you would need to use a special Visual C++ project. This project is available on the user community.

Global C++. This is C++ code that is globally scoped. It must contain complete functions. These functions can be accessed by any node nested below this node. Typically this type of node is found as the first node under the main Tool folder.

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:



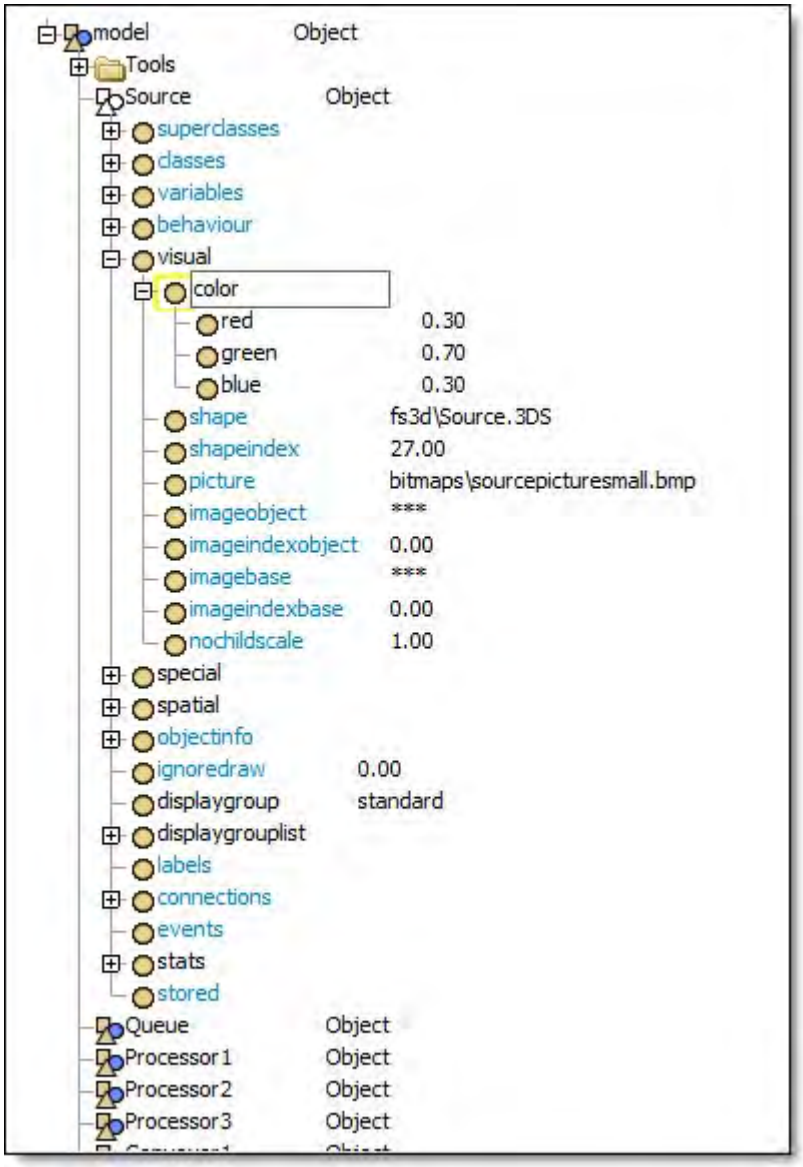
A highlighting box will be placed around the object icon and an expand tree symbol  will be place to the left of the object icon. If you select this expand tree symbol, the data nodes for that object will be displayed as shown below.



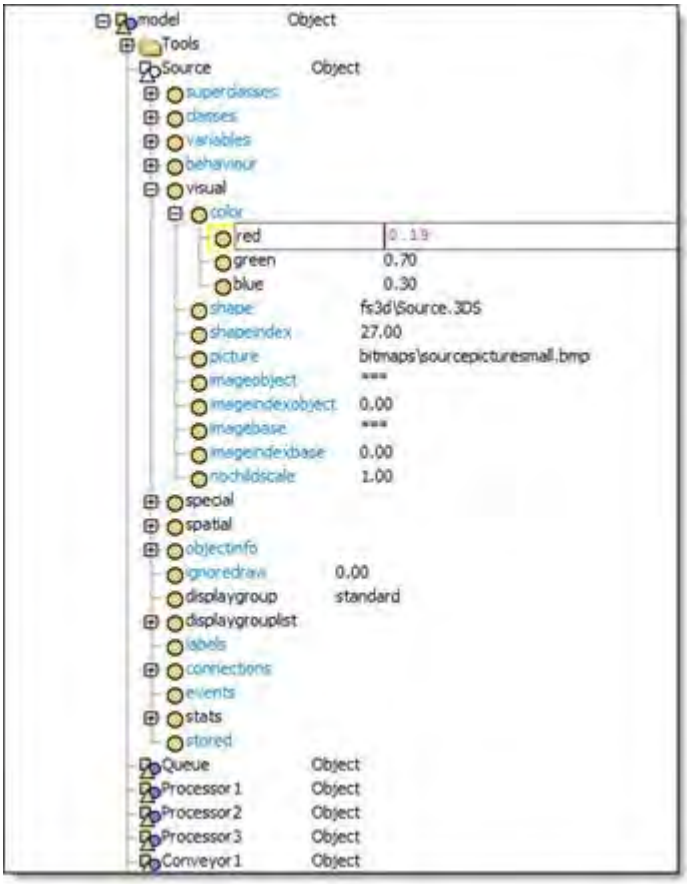
As objects and data nodes are expanded, the tree view can quickly grow to be outside the viewing limits of the tree view window. FlexSim allows you to move the tree around in the window by using the mouse. To move the tree around in the window just click-and-drag on the left side of the tree, use the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

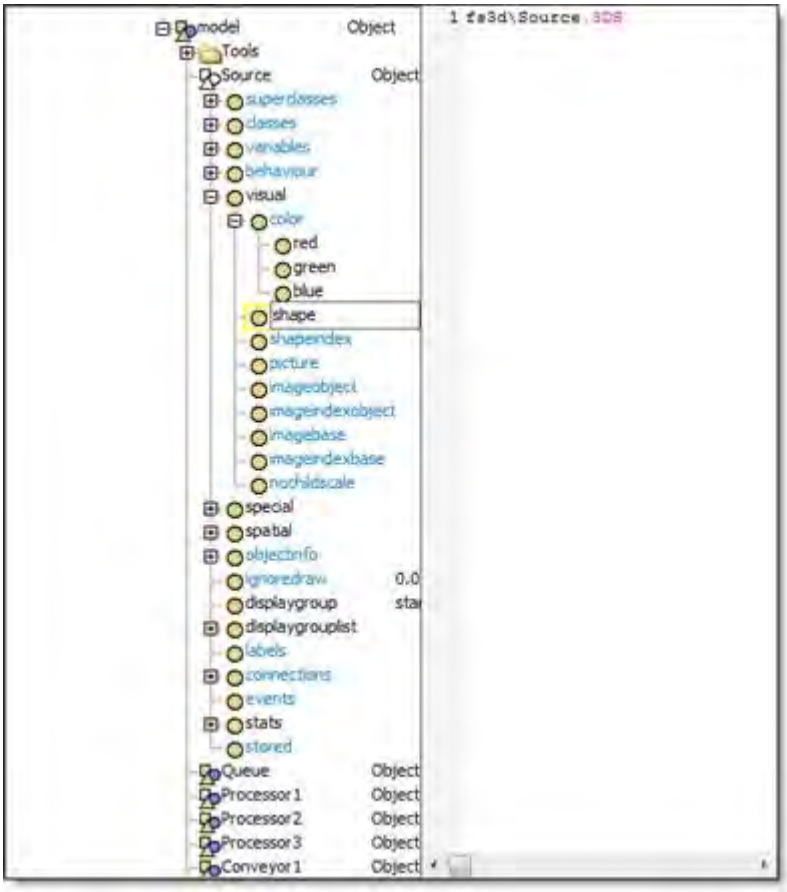
Data nodes can be expanded by clicking on the "+" to the left of the node icon. Since data nodes can have values or text you will see the text information or the data values to the right of the node.

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 up and down. You can also use the scroll bar, mouse wheel, or PageUp/PageDown buttons to move the tree up and down.



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. 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.


model Object
Tools
Source Object
superclasses
classes
variables
behaviour
visual
color
red 0.19
green 0.70
blue 0.30
shape fs3d\Source.3DS
shapeindex 27.00
picture bitmaps\sourcepicturesmall.bmp
imageobject ***
imageindexobject 0.00
imagebase ***
imageindexbase 0.00
nochildscale 1.00
special
spatial
objectinfo
ignoredraw 0.00
displaygroup standard
displaygrouplist
labels
connections
events
stats
stored
Queue Object
Processor1 Object
Processor2 Object
Processor3 Object
Conveyor1 Object


1 fs3d\Source.3DS
model Object
Tools
Source Object
superclasses
classes
variables
behaviour
visual
color
red
green
blue
shape
shapeindex
picture
imageobject
imageindexobject
imagebase
imageindexbase
nochildscale
special
spatial
objectinfo
displaygrouplist
labels
connections
events
stats
stored
Queue Object
Processor1 Object
Processor2 Object
Processor3 Object
Conveyor1 Object

As you can see, the tree is the repository of all data for the model. The properties 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 properties windows to avoid inadvertent deletion of model data. The properties windows are accessible in a tree view by double-clicking on the object icon  or by selecting Properties from the right-click menu. For more information on the Tree view, see the Tree Window page.

Model Repeatability

FlexSim allows you to have variability in your simulation runs through the use of statistical distributions. These randomly generated values give you more accurate results than a static set of data. Using the Experimenter you can run multiple replications of various scenarios, each of which can produce different results. You can compare and analyze those results for a more thorough understanding of your simulated system, or for the creation of confidence intervals.

However, when you are building a model and want to validate your results, or you are trying to fix an error in your model, these varying results may be undesired. In these cases it can be useful to have your model be repeatable.

This topic lists all the ways to make your model repeatable.

Repeat Random Streams

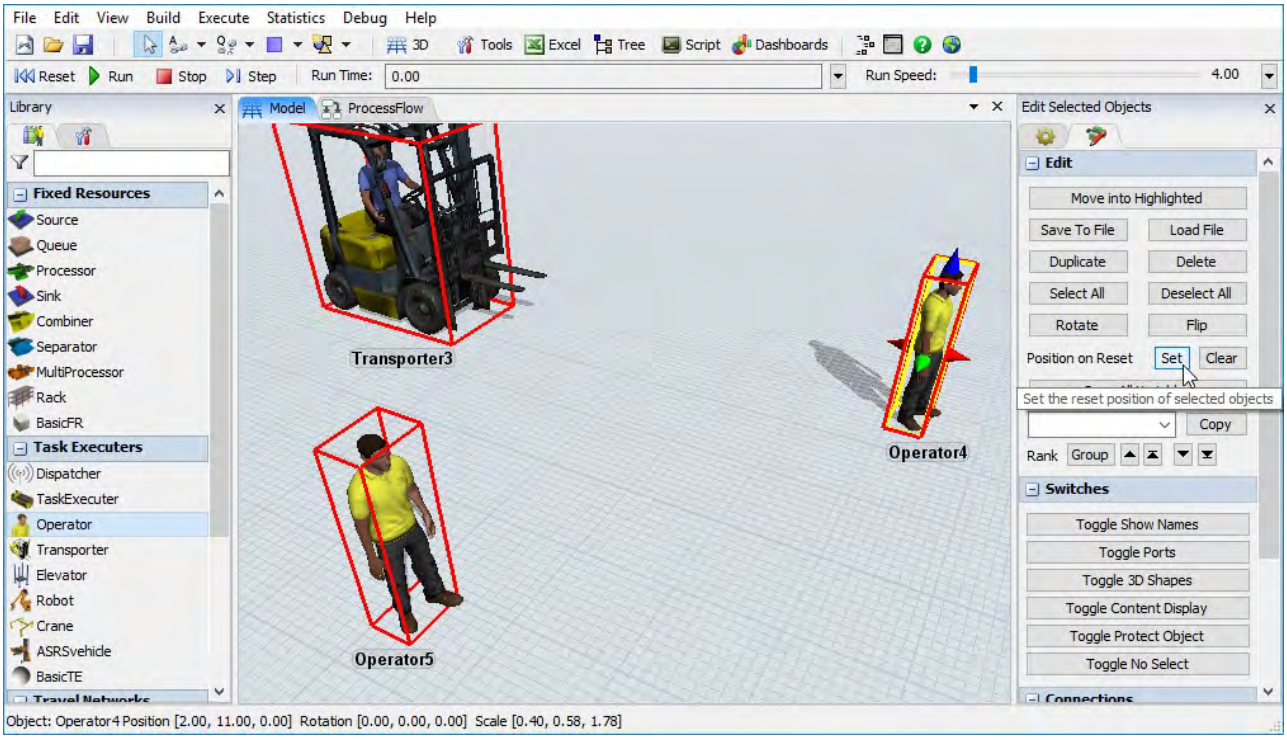
This option can be found under the Statistics Menu. If checked, every time you reset and run the model the random number generator will produce the same sequence of numbers. This makes it possible to have the results of your model be the same for each simulation run. If repeatable results are desired, this must always be checked.

To truly have a repeatable model, the conditions below must be satisfied.

Task Executer Reset Position

If your Task Executers do not have a reset position, then when you reset and run your model, those Task Executers will start from wherever they were at the end of your last simulation run. This means that the Task Executers will have different travel times from run to run, which will affect your results.

In order to set a reset position, right click on the Task Executer and select Edit > Set Object Reset Position. Or to set the reset position for multiple Task Executers at once, shift-select or Ctrl-click a group of Task Executers, then highlight one of the selected objects and click the "Set" Position on Reset button in the View > Edit Selected Objects section of Quick Properties.



Custom Draw Code

If any of your objects have code in the "Custom Draw" trigger, make sure that that code does not include any calls to functions that use the random number generator, such as statistical distributions.

Item Locations

If any of your code queries an item's location, you must call the function updatelocations() on the item or the Fixed Resource object containing the item before making your query. If updatelocations() is not called, the location returned will be where the item was when it was last drawn. Drawing rates vary with the run speed, so if updatelocations() is not called, model results will vary between running the model slowly and running it quickly, or running on different hardware.

Keep in mind that loading/unloading an item while it is moving (on a conveyor, for example), requires that its location be queried. As such, updatelocations() should be called immediately before loading/unloading items to/from a moving location.

Initializing Variables

Declared variables in custom code should always be initialized to some default value. Failure to do so can result in variables containing random data. If there are any circumstances under which a variable might be accessed before it has explicitly been set, random data in that variable may cause non-repeatable behavior in your model.

```
int myNumber;                    //the myNumber variable will have random data
int myValidData = 0;             //by initializing to a value that I can account for in my code, I guarantee that no random behavior will be introduced by uninitialized variables treenode myObject; //because I did not initialize this value, there is no telling what it is referencing. It is not NULL by default!
treenode myNode = NULL;   //much better to specify a NULL treenode, or some other treenode, as the default value, rather than assume (or hope) that an uninitialized variable will be NULL.
```

Resetting Data

If at any time during your model run values of persistent data are changed, that data will not change back to its original state on model reset. Examples of persistent data include Global Tables and object labels. Be sure to add code into OnReset triggers that will change the data back to its original state.

For object labels, one way to do this is to check Automatically Reset Labels on the object's Label page.

Non-Repeatable Data

If any of your logic is based on data that will always vary, your model will never be repeatable. You should re-architect those portions of your logic to avoid basing decisions or conditions on non-repeatable data.

Examples of non-repeatable data include memory addresses (using tonum() to get the memory address of a treenode, note that tonum() is deprecated and should no longer be used) and real time (using the realtime() function to get real-world time stamps).

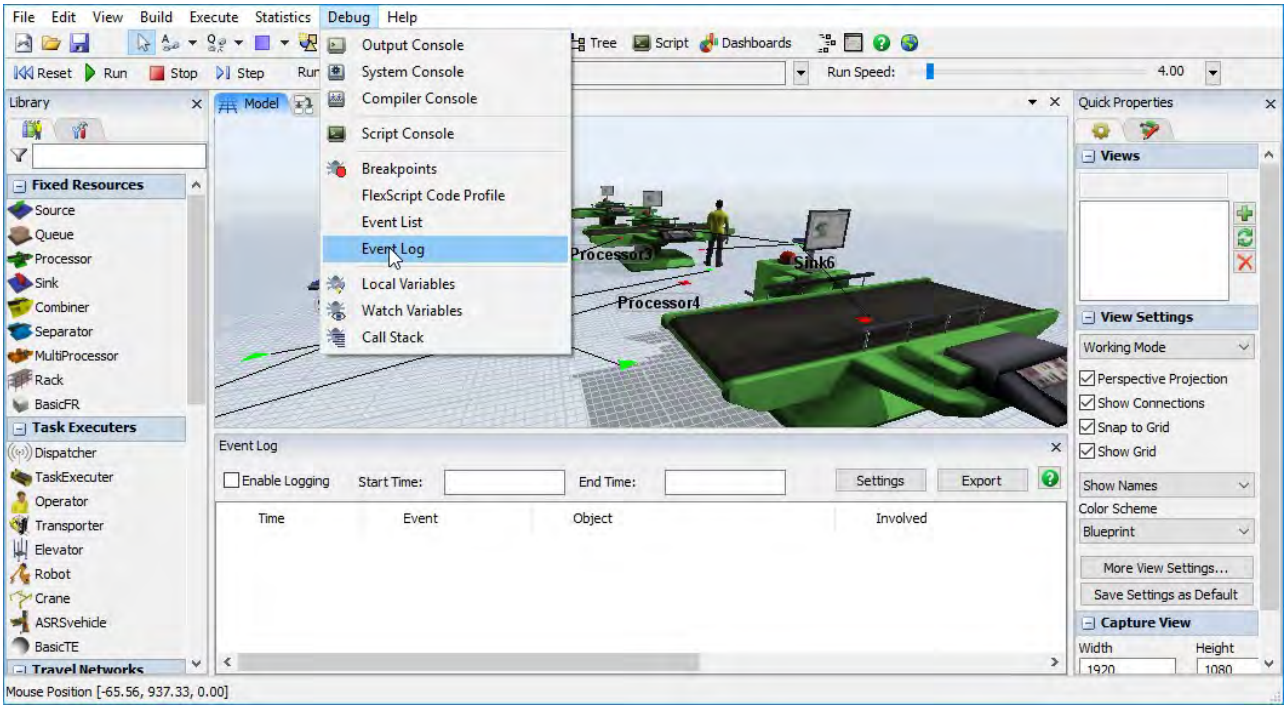
Eliminate all Modeling Errors

Track down and fix any issues that are causing System Console or Compiler Console errors. Do not ignore these errors! These errors tell you that something in your model is wrong and should be fixed.

Among other issues, errors in your model can cause unpredictable behavior and may cause your model to not be repeatable.

Debugging Non-Repeatable Models

If a model is still not giving repeatable results and the above suggestions have already been applied, the best way to debug the model is by looking at the Event Log, which can be found under the Debug menu.



1. Set up the model exactly how you want to test it:
    - Event Log open, "Enable Logging" checkbox ticked
    - Start Time and End Time values set
    - Various interface windows open or closed (3D views, Global Tables, etc.)
    - etc.
2. Save the model.
3. Close FlexSim.
4. Double-click on the model to open FlexSim with the model open.
5. Press Reset once and Run once.
6. Once the run is complete, or has passed your desired Stop Time in the Event Log window, press Export to save the log as a CSV file.
7. Press Reset once and Run once.
8. When the second run is complete, Export the new Event Log using a different file name.
9. Now that you have the two event logs, you can compare them using a text comparison tool such as WinMerge.
10. Find the first event where the time differs between the two logs.
11. Inspect the objects involved in that event, or the objects that created that event, to see what could be different between the two runs.
12. Rerun the model to just before that point, then step through events leading up to the diverging event. Use the debug tools to step through your code where necessary.

For some models, it helps to click the Event header on the Event Log and filter out particular events. For instance, if you are sending messages that include a memory address as one of the message parameters, those addresses will be different for every simulation run and will make finding other differences in your exported log file hard to find. Sometimes this is expected and can be accounted for by removing these events from your log file.

You can close and reopen FlexSim each time just in case the model is doing something to the application's state or variables in the MAIN or VIEW trees that aren't reset when you just open a model. This helps ensure that even if subsequent runs aren't the same, the first run in each test will match the first

run of each subsequent test. Steps 3-6 should produce identical runs even if the model isn't repeatable after a simple Reset/Run without closing FlexSim.

Additional memory addressing considerations

If memory addresses of objects (retrieved with the tonum() command, note that tonum() is deprecated and should no longer be used) are used in your modeling logic, they should be stored as doubles. You will have issues if you cast them into integers. int is a signed datatype, but memory addresses are unsigned. Therefore, if you try to call tonode() on a memory address that has been stored as an integer, it might be a negative number and return NULL instead of actually returning the node.

64-bit systems compound the problem. Integers are 32-bit and, on a 64-bit system, memory addresses are 64-bit. This means you may have a memory address that is larger than an int, and it will get truncated when you cast it into an int. When you try to reference the node with tonode(), some other location in memory will be referenced. Always store memory addresses as doubles!

Or, better yet, you should try to use the reference directly or cast it into a treenode as soon as possible. Avoid storing the memory address as a declared variable whenever possible. For example, if msgparam(3) returns a memory address:

treenode qtde = tonode(msgparam(3)); //convert to a treenode as soon as possible int qtde = msgparam(3); //don't do this! always convert to treenode as soon as possible or, if necessary, store the memory address as a double In summary:

- Always convert a memory address to a treenode using tonode() as soon as possible.
- If you must pass memory addresses through numbers, use a double datatype, not an int.
- A double stores 64 bits and will properly reconvert back to a treenode when you call tonode() on its value.

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. 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. Step-by-step model construction

The following lessons are contained in this tutorial:

Lesson 1

Building a simple model that will process 3 different flowitem types. Each item type 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 Dashboards. 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.

Labels

This lesson will introduce you to the use of labels. You will also be introduced to Pull Requirements.

Global Properties

This lesson will introduce you to the basic concepts of global properties that can be used throughout your model. You will also be introduced to the Combiner and Separator objects.

User Events

This lesson will introduce you to User Events. Which can be used to execute specific code at specified times during the model run.

Time Tables

This lesson will introduce you to Time Tables.

Kinematics

This lesson will introduce you to using Kinematics to perform simultaneous movements with a single object.

Task Sequences

This lesson will introduce you Task Sequences and creating custom Task Sequences.

SQL

This lesson will introduce you to reading from and writing to a SQL database from FlexSim.

Feel free to contact our technical support group if you have any questions while working on these tutorials. FlexSim technical support can be reached at 801-224-6914, or you can contact us through a support request by going to the Help Menu and selecting Support Request. We hope you enjoy learning how FlexSim can help you optimize your processes.

Note: The statistics in the models you build might not be exactly the same as those found in these tutorials. Small differences in simple things like object placement can change the long term results. For the purpose of these tutorials, these differences are not important.

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.

Conveyor Module Required
This lesson requires the conveyor module. If you do not have this module, you can download and install if from the main menu under Help > Online Content.

Approximate Time to Complete this Lesson

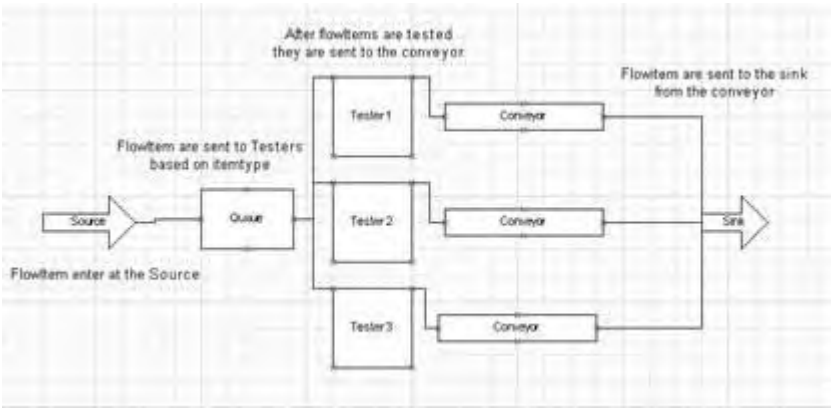
This lesson should take about 30-45 minutes to complete.

Model Views

FlexSim uses a three-dimensional modeling environment. There are two options for the view: perspective view and orthographic view. The orthographic view will look more flat, where as the perspective view will have a more real world feel to it. 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 types that will arrive based on a normal distribution. Types will be uniformly distributed between type 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 type 1, another for type 2, and the third for type 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.



Click here for the Step-By-Step Tutorial.

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: Type 1 to Tester 1, type 2 to Tester 2, type 3 to Tester 3.

Building Lesson 1 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.

If at any time you encounter difficulties while building this model, a fully functional tutorial model can be found at http://www.flexsim.com/tutorials

Starting a New Model

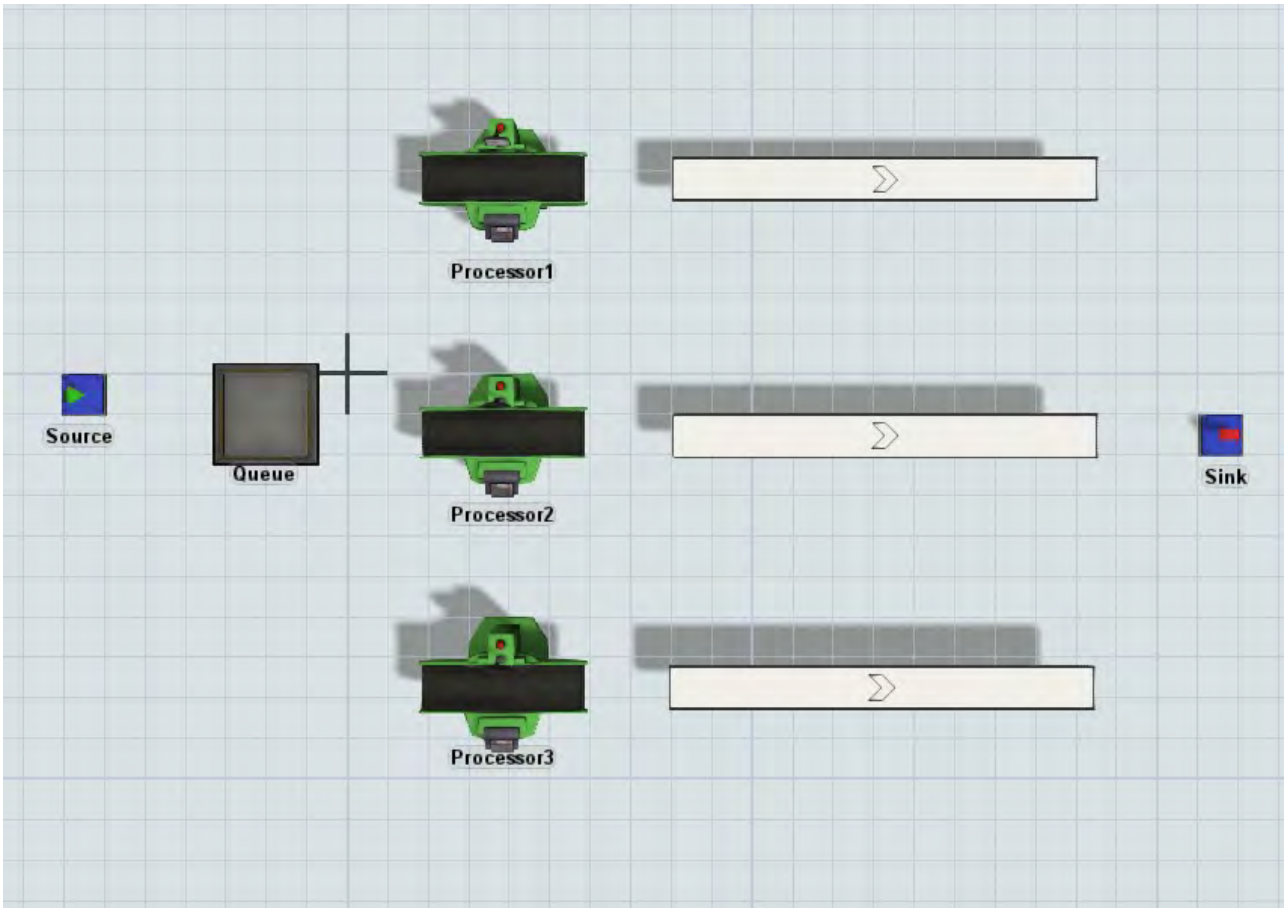
- Open FlexSim by double-clicking on the FlexSim icon on your desktop. The Startup Wizard appears by default. Select the "Build a New Model" option.

FlexSim allows the user to select appropriate units for a model. By default, the Model Units dialog will appear for each new model. You can select units for time, length, and fluids. The units you choose will be assumed throughout the model. For this model, use the following:

- Time Units: Seconds.
- Length Units: Meters.
- Fluid Units: Liters.

Step 1: Create the Objects

- Create a Source in the model and name it *Source* (To see how this is done, click here).
- Create a Queue, 3 Processors, 3 Conveyors, and 1 Sink in the model. Place and name them as shown below. To name an object: double-click on it, change its name at the top of the Properties window, and press Apply or OK. Click Here to see how this is done.

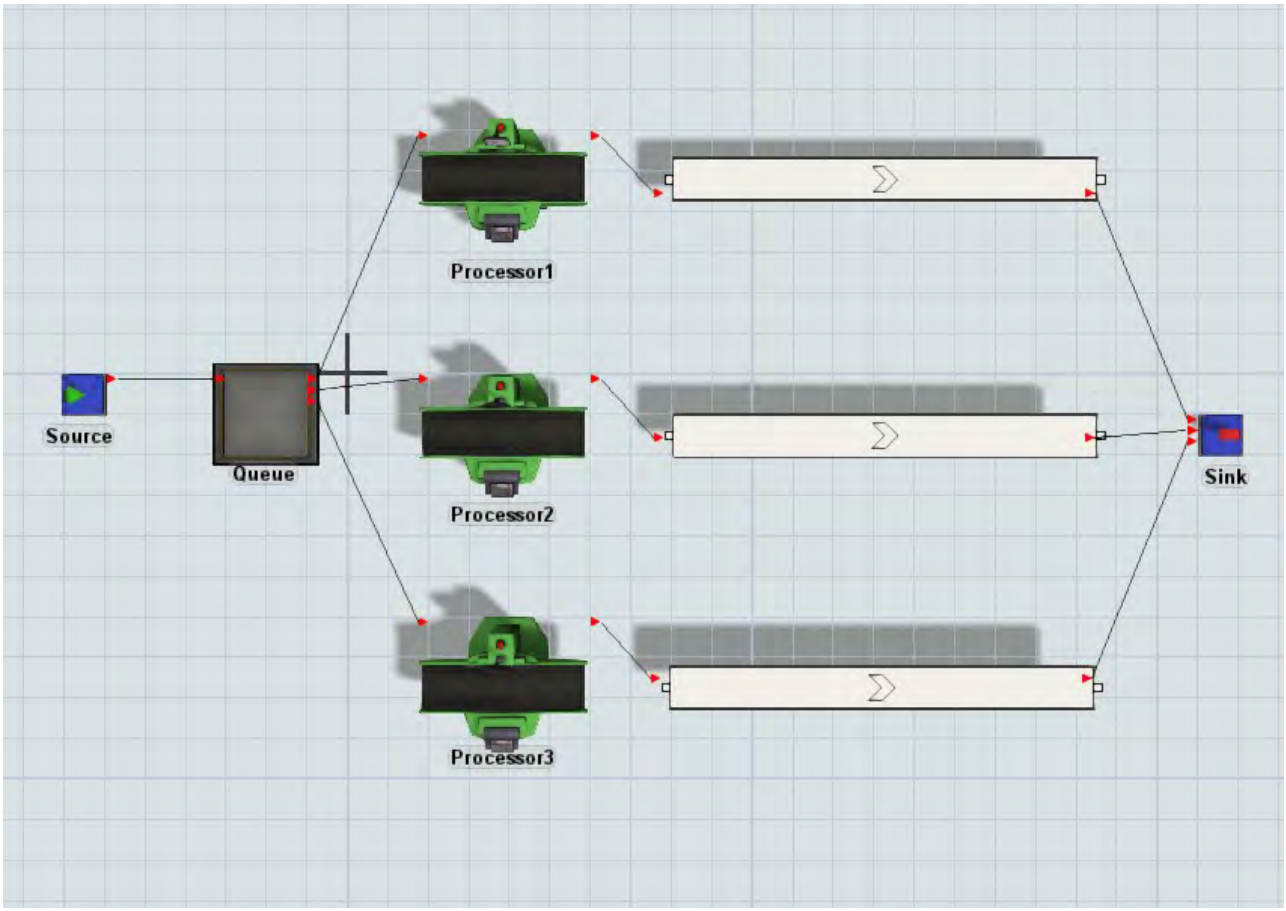


Step 2: Connect the ports

Enter the connection mode by either clicking the button or by pressing and holding the A key on the keyboard. Once in the connection mode, there are two ways to make a connection between objects. You can either click on one object and then click on another object, or you can click and drag from one object to the next. Either way, keep in mind that the flow direction of a connection is dependent on the order in which you make the connection. Flow goes from the first object to the second object in the connection.

Incidentally, connections can be broken by clicking the button or by pressing and holding the Q key on the keyboard while clicking or dragging from one object to another in the same manner as when you connected them.

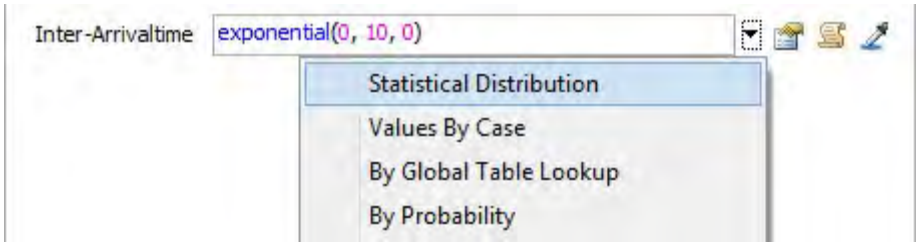
- Connect *Source* to *Queue.*
- Connect Queue to *Processor1*, *Processor2*, and *Processor3.*
- Connect *Processor1*, *Processor2* and *Processor3* each to its adjacent conveyor.
- Connect each of the three conveyors to *Sink*.

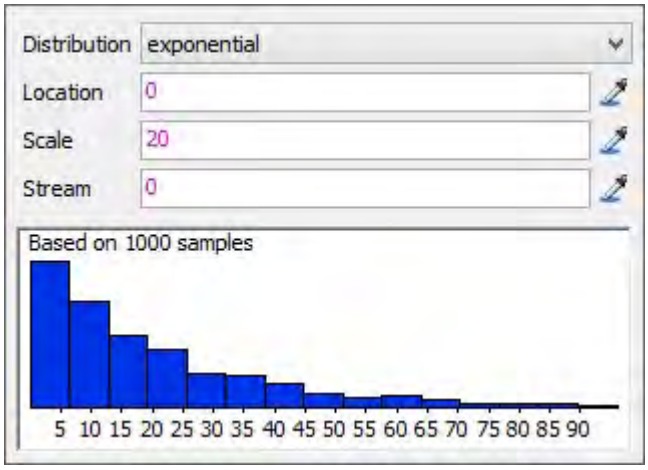


Step 3: Assign the arrival rate

In this model we need to change the Inter-Arrival time and the item type to generate 3 types of items.

- Double-click on the *Source* to open its Properties window
- On the Source tab, select Statistical Distribution from the Inter-Arrivaltime list. A statistical distribution popup will appear.
- Set Distribution to exponential.
- Set Location to 0.
- Set Scale to 20.
- Set Stream to 0.





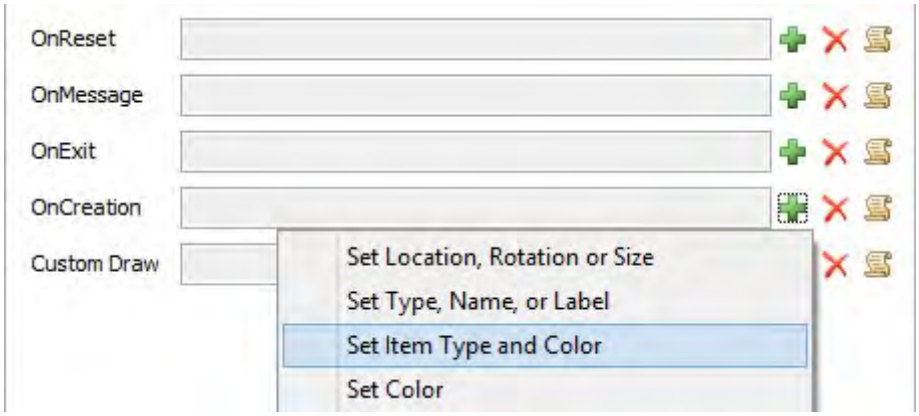
The next thing we need to do is assign a type number to the flowitems as they enter the system. This will be uniformly distributed between 1 and 3. The best way to do this would be to change the type on the OnCreation trigger of the Source, so don't close the Properties window yet.

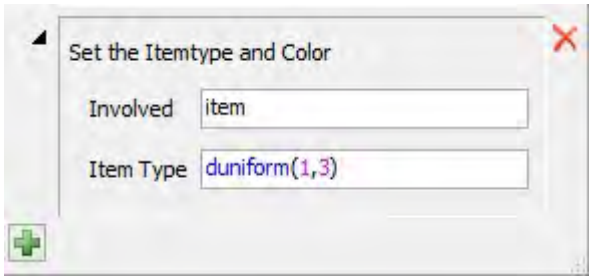
Step 4: Set Item Type and Color

- Click the Triggers tab, and add a function (click the button) to the OnCreation trigger and select the Set Item Type and Color option. The code template popup will appear.

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. The default values will be used in this example.

- Click OK to apply the changes and close the Properties window.





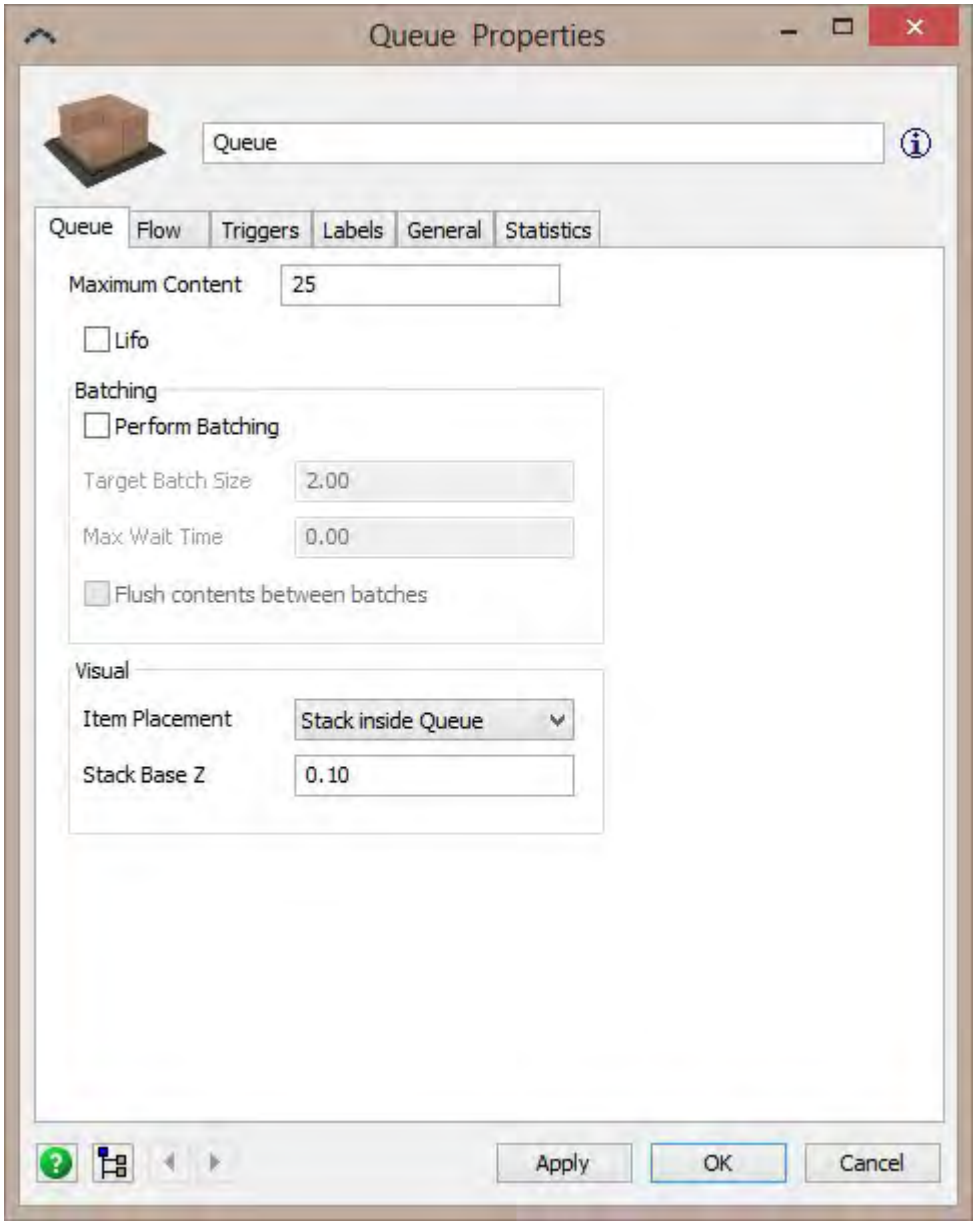
The next step will be to detail the queue. Since the queue is a place to hold flowitems until they can be processed at the processor, there are 2 things we need to do. First, we need to set the capacity of the queue to hold 25 flowitems. Second, set the flow options to send type 1 to *Processor1*, type 2 to *Processor2*, and type 3 to *Processor3*.

Step 5: Setting the Queue Capacity

You may set the Queue's Maximum Content by clicking on the Queue in the 3D view to display its properties in the Quick Properties window.

Alternatively, you can open the object's properties window:

- Double-click on the queue to open it's Properties window.
- Change the Maximum Content to 25. • Don't close the Properties window yet.

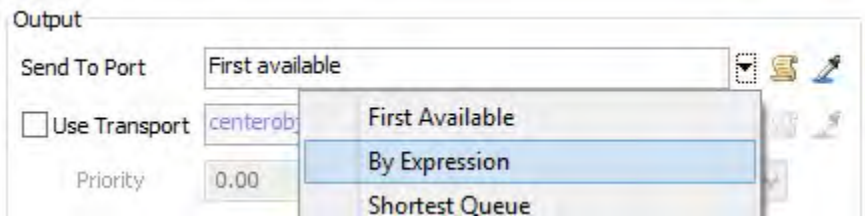


Step 6: Define the Flow for the Queue

You may define the Queue's flow by clicking on the Queue in the 3D view to display its properties in the Quick Properties window.

Alternatively, you can open the object's properties window:

- Click the Flow tab in the Properties Window to set the flow options for the queue.
- On the Send To Port list, select By Expression



Since we have assigned a type number equal to 1, 2, or 3, we can now use the type to specify the port number through which flowitems will pass. Notice that the default output port is: item.type. Leave this as it is. Processor 1 should be connected to port 1, processor 2 should be connected to port 2 and so on. Click outside of the box to apply the trigger.

Click the OK button to apply and close the queue's properties window. The

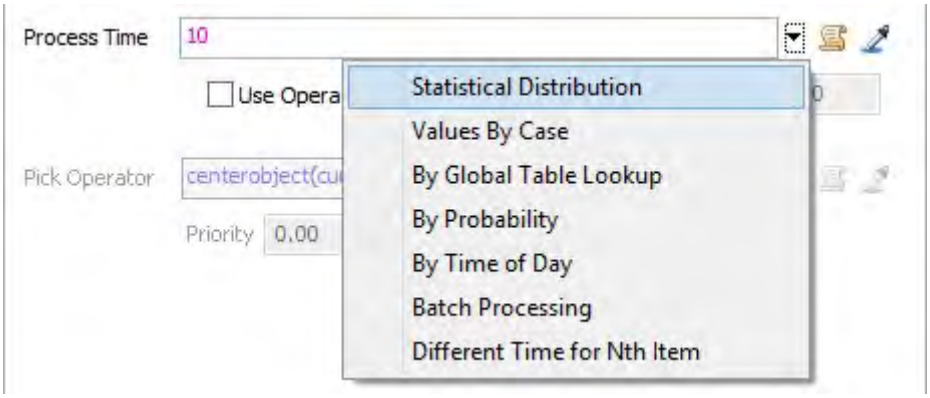
next step is to set the processor times.

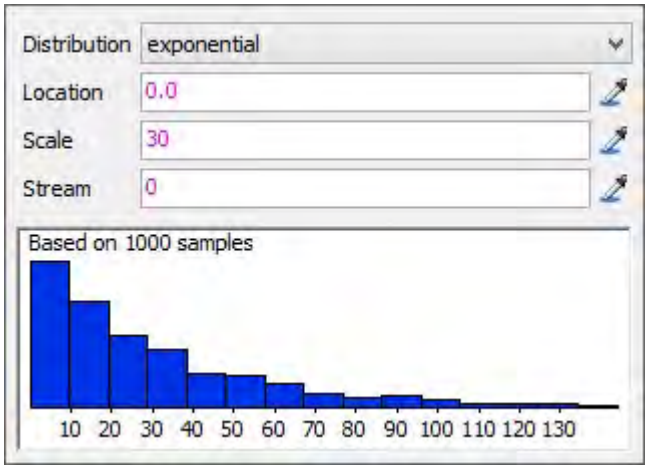
Step 7: Define the Process Time

You may define the Processor's Process Time by clicking on the Queue in the 3D view to display its properties in the Quick Properties window.

Alternatively, you can open the object's properties window:

- Double-click *Processor1* to open its Properties window.
- On the Processor tab, in the Process Time section, select Statistical Distribution from the Time list. The statistical distribution popup will appear.
- Set Distribution to exponential. • Set Location to 0
- Set Scale to 30.
- Set Stream to 0.
- Repeat this for the other 2 processors.





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

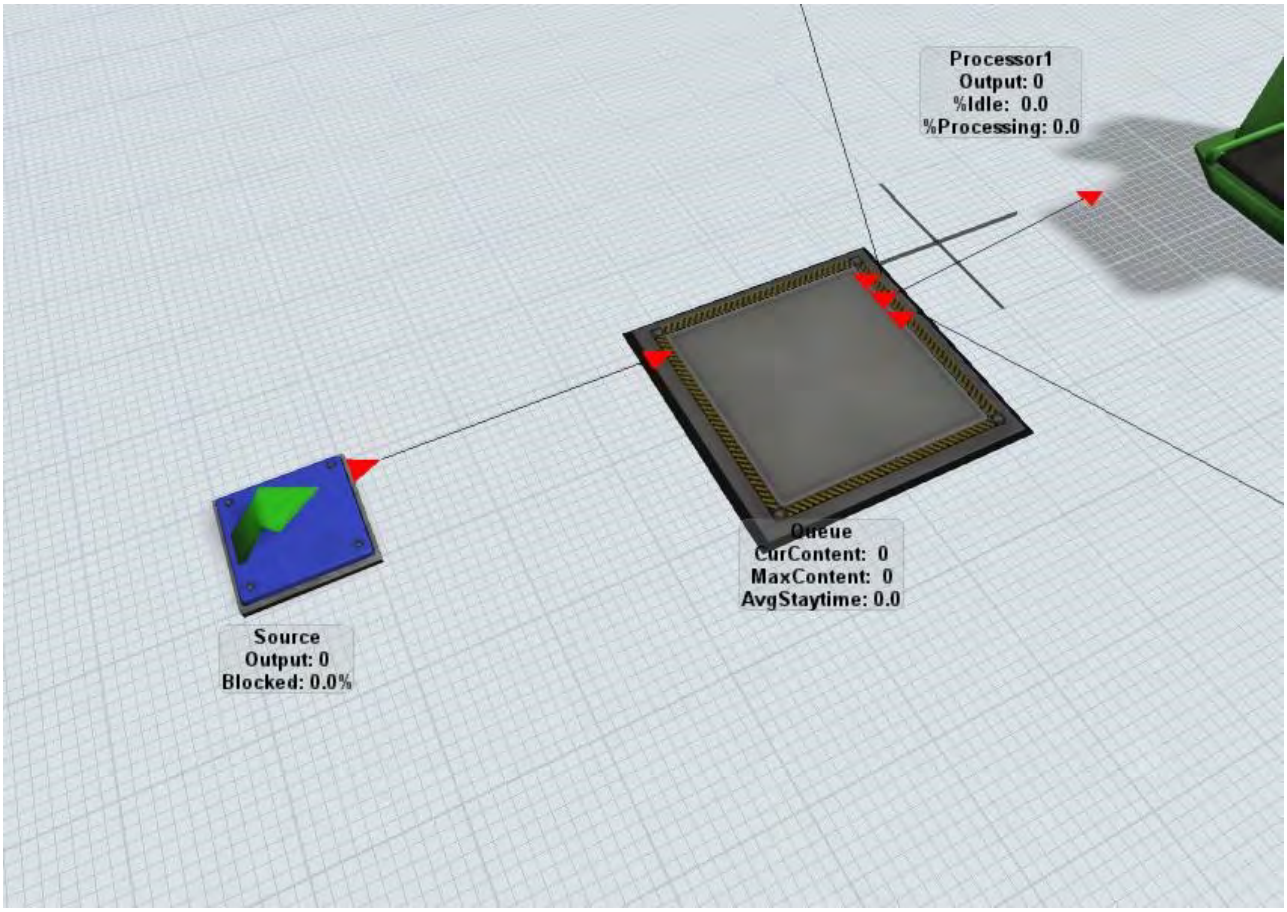
Now we are ready to run the model.

Step 8: Reset and Run the model

- Always click the Reset button to reset system and model parameters to their initial state before running a model.
- Click the Run button to start the simulation.

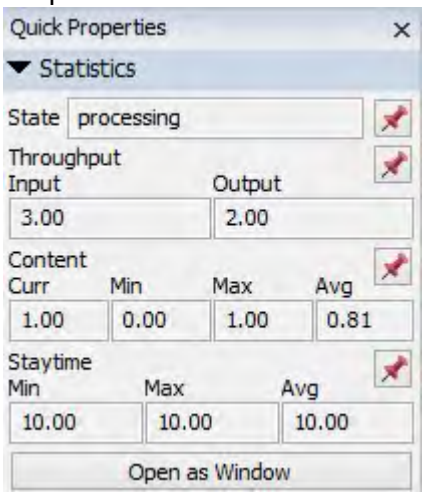
You should see flowitems entering the queue and then moving to the processors. From the processors, flowitems should move to the conveyors and then to the sink. You can change how fast the model runs by moving the Run Speed slide bar on the Simulation Run Panel.

Step 9: Viewing simple statistics



The above image shows how to view simple statistics for each object. If nothing is showing or only the names are showing, you can change the View Settings to show the statistics. To change the View Settings, click somewhere in the background of the view and in the Quick Properties window to the right, change the Show Names combo box to Show Names and Stats.

You can view more statistics of an object by clicking on the object and viewing the statistics tab in the Quick Properties.



Step 10: Save Model

Save your model by clicking the Save button on the main toolbar. The "Save FlexSim Model file" window will appear allowing you to navigate to the folder where you want to save your model. Change the "File name" to an appropriate name (lesson1.fsm) and select save. Remember that the file name extension must be .fsm.

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 Tutorial Introduction

Lesson 2 introduces the concept of adding operators and transporters to a model, and explores object properties 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 and modify object properties
- How to add a team of operators to the model
- How to add a fork truck transporter to the model
- 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.

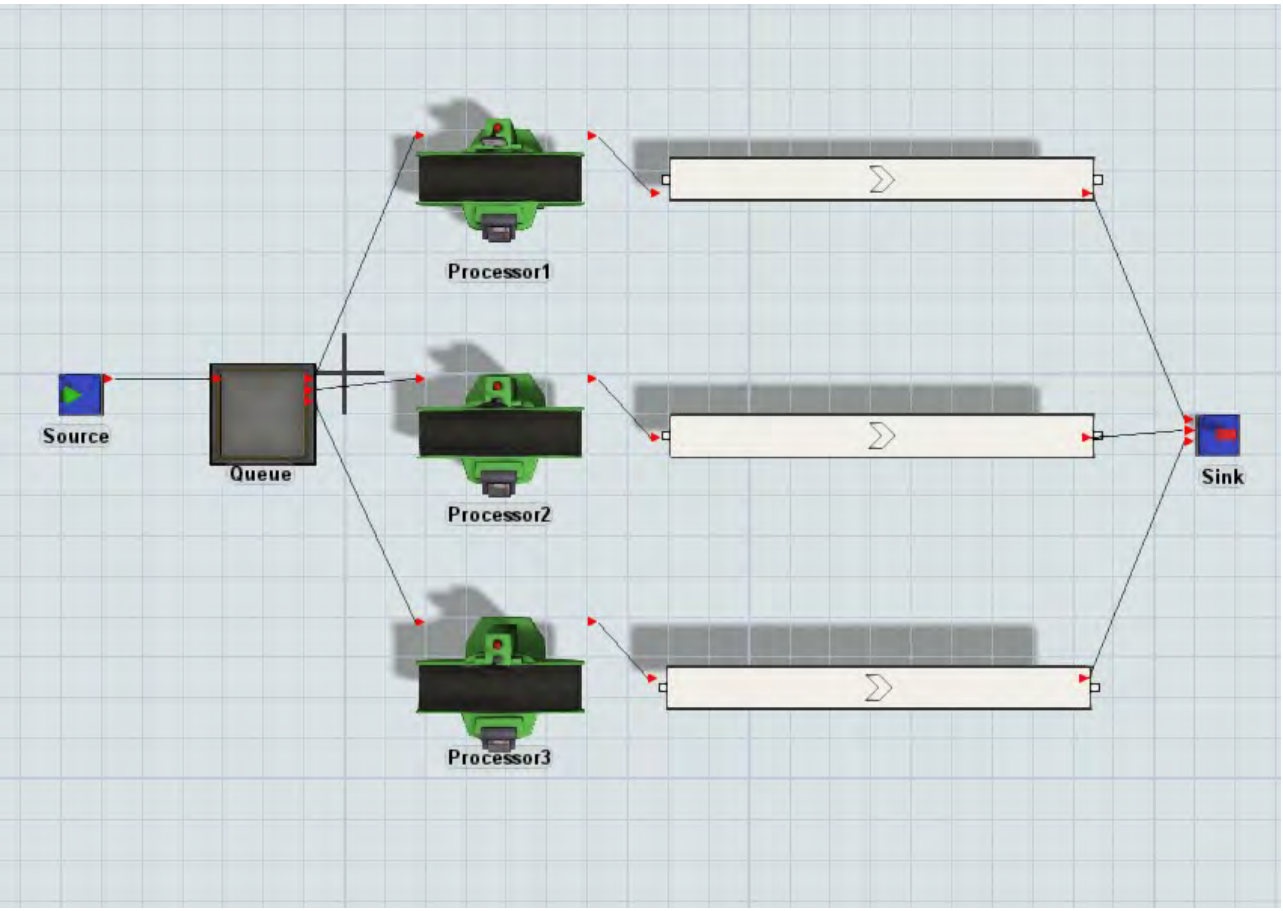
Approximate Time to Complete this Lesson

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

Model 2 Overview

In model 2 we will use a team of operators to perform the setup for the processing of the flowitems in the model. One of two operators will need to set up the processor. Once set up, the process can run without the need for the operator to be present. The operators must also carry the flowitem to the processor before the setup can begin. When the process 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. Queue Capacity: 10

Lesson 2 Tutorial Step-By-Step Model Construction

Building Model 2

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

If at any time you encounter difficulties while building this model, a fully functional tutorial model can be found at http://www.flexsim.com/tutorials

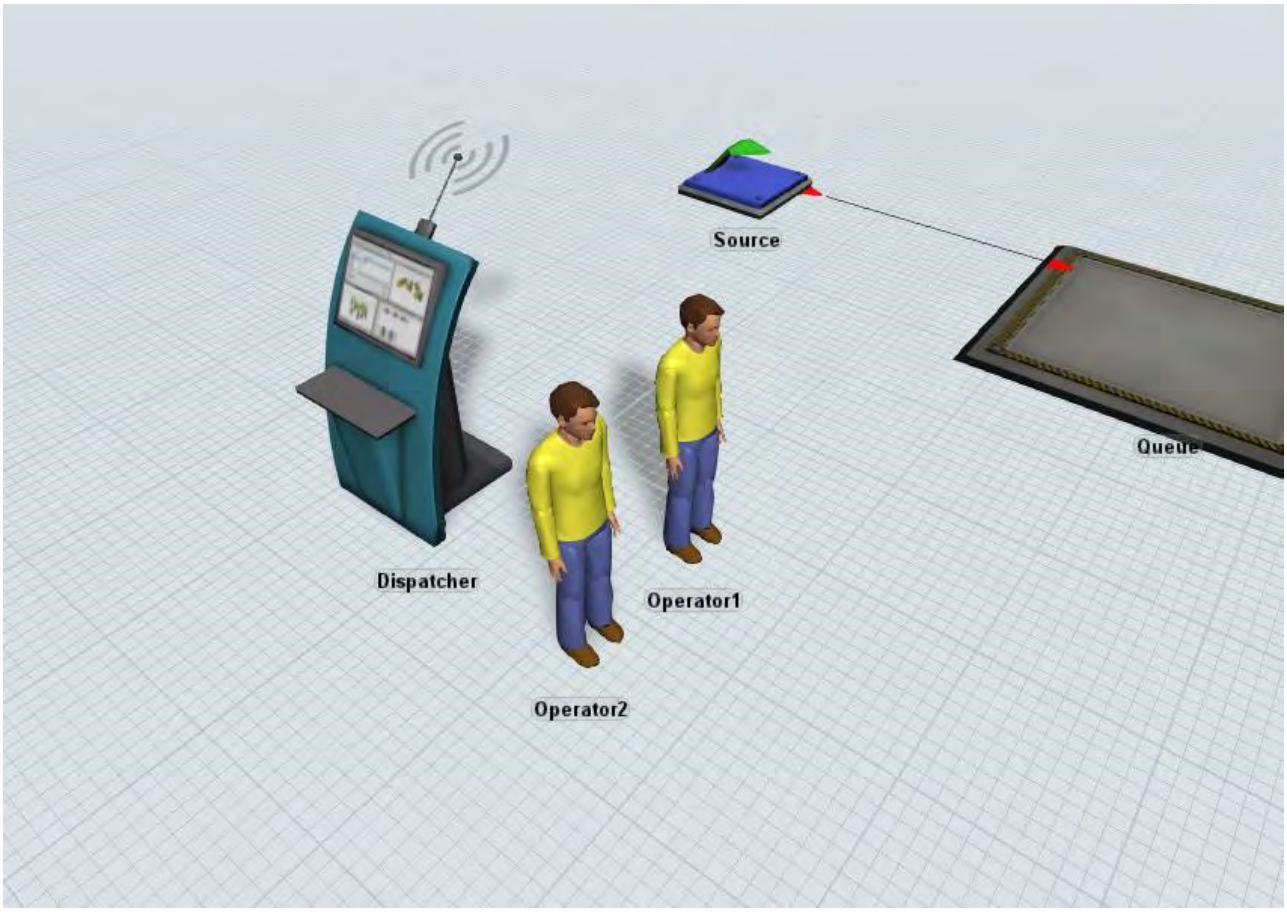
Step 1: Load model 1

- If you do not have Model 1 open, load it by clicking the  button on the toolbar. Select the file for model 1 (.fsm file) saved from lesson 1.
- We want to make our flowitems exit the Source at a faster rate for this lesson. Double-click on the *Source* to open its Properties window, and under Inter-Arrivaltime change the Scale to 12.

Step 2: Create a dispatcher and 2 operators

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.

- Create a Dispatcher and name it *Dispatcher.*
- Create 2 Operators and name them *Operator1* and *Operator2.*



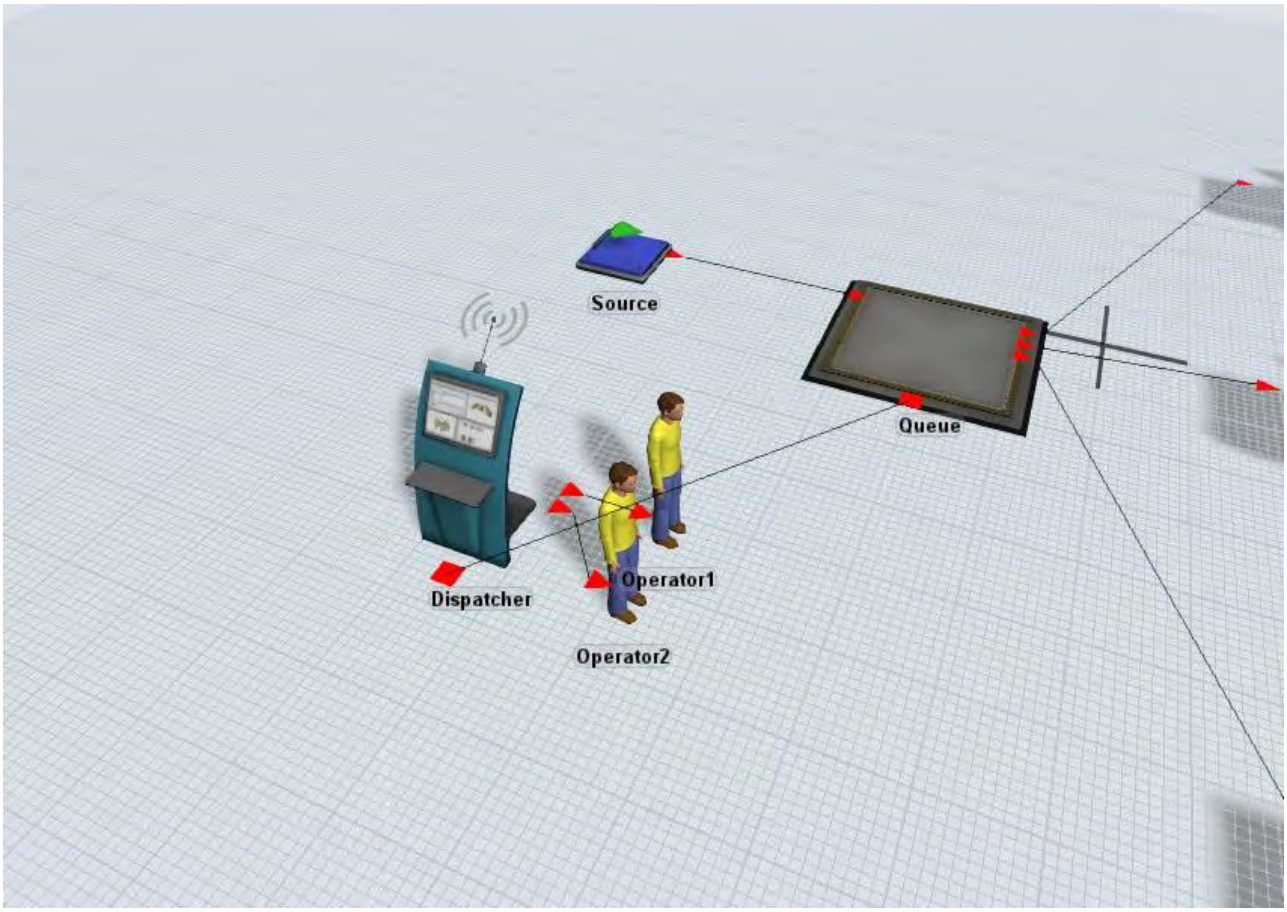
Step 3: Connect the Dispatcher and Operators

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 send them to 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. The central port is located in the center bottom position of the object. 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 button a connection from the dispatcher's central port will connect to the central port of the queue. You can also use the Connect Center Ports mode from the Mode Toolbar to do this.

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 must be done for each operator.

- Connect *Queue* to *Dispatcher* with a center connection (S key).
- Connect *Dispatcher* to *Operator1* and *Operator2* with a standard connection (A key).



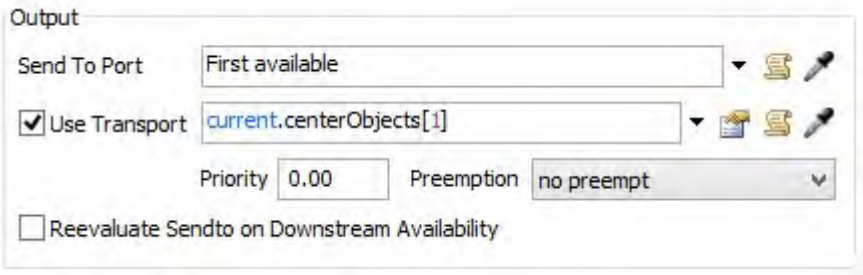
Step 4: Modify Queue Flow to Use Transport (Operators)

The next step is to modify the Queue's flow parameters so it uses the operators.

You may define the Queue's flow and transport options by clicking on the Queue in the 3D view to display its properties in the Quick Properties window.

Alternatively, you can open the object's properties window.

- Double-click *Queue* to open its Properties window.
- Click the Flow tab
- Check the Use Transport box. The Request Transport From list will become available. This picklist allows you to select which Transporter or Operator to move the item based on the center port number. In this case it is the object connected to center port 1 (the Dispatcher) that assigns the operator to the task, so the default settings will work here.
- Click OK to close the Properties window.



Step 5: Save the Model, and Test Run

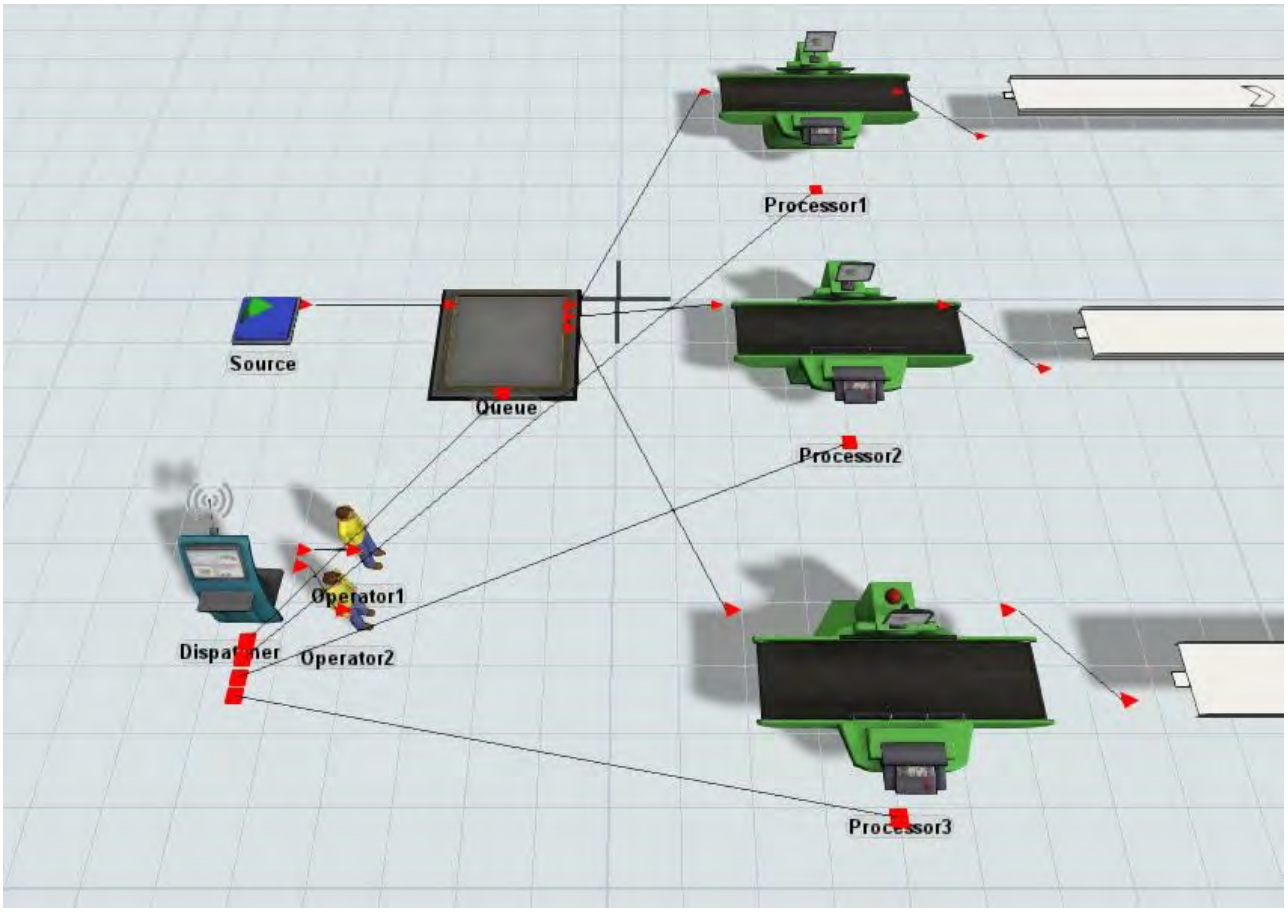
Now we should run the model to make sure that the changes we have made are working.

- Reset the model and then Save the model by clicking the button on the toolbar.
- Run the model to verify that the operators are moving the flowitems from the queue to the testers.

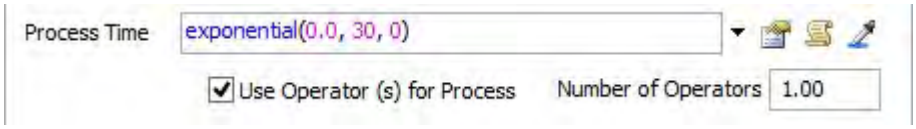
Step 6: Using Operators for the Process Time

In order for the testers to use the operators during processing, a connection must be made between the central ports of each tester to central ports of the dispatcher. Then the process must be changed to require an operator.

- Connect *Dispatcher* to *Processor1, Processor2,* and *Processor3* with center connections (S key).



- Double-click *Processor1* to open its Properties window.
- On the Processor tab, check Use Operator(s) for Process. Number of Operators and Pick Operator will become available.



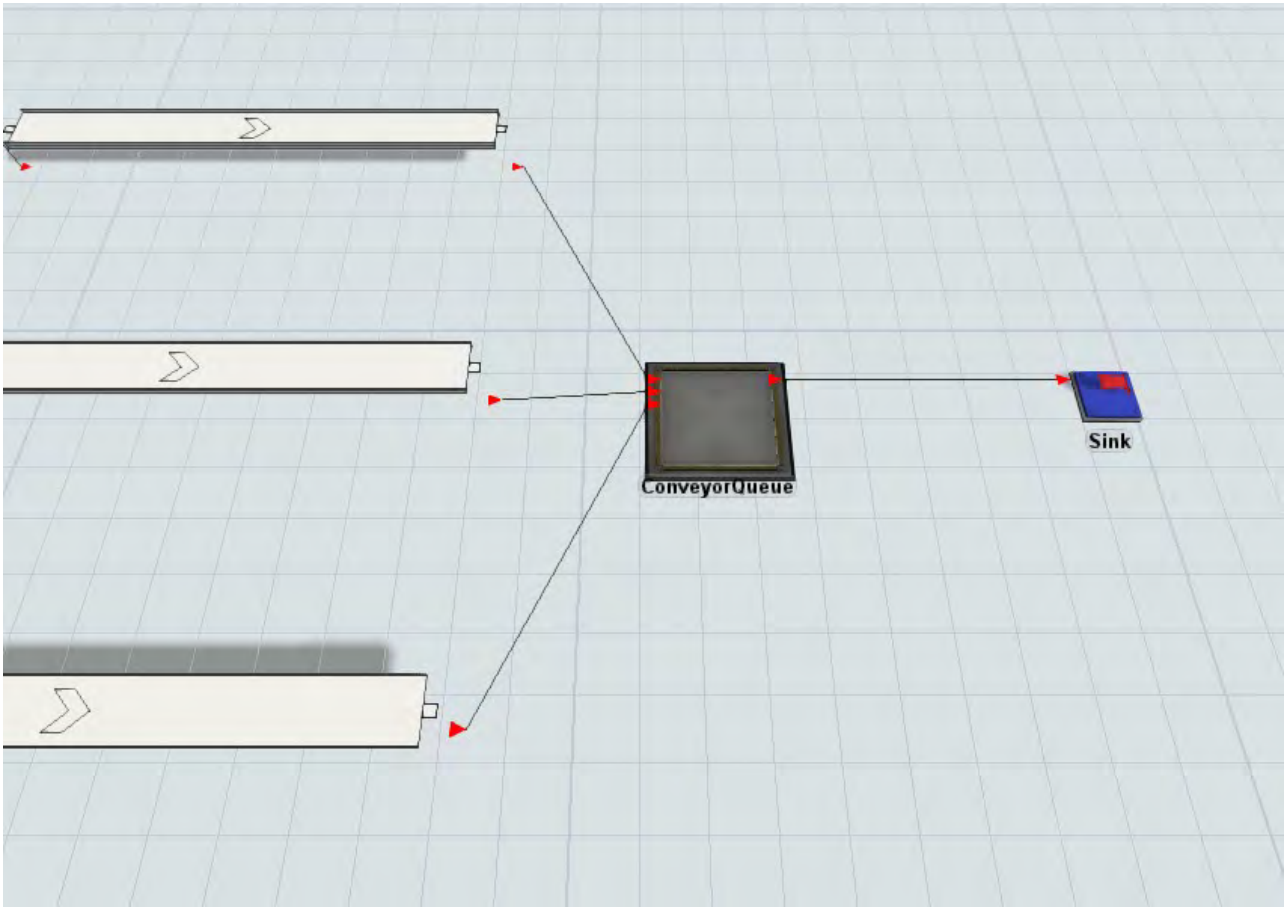
-
-

  Click OK to close the Properties window.
  Repeat this step for *Processor2* and *Processor3*.

Step 7: Disconnect the Conveyors from 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.

- Disconnect the three conveyors from *Sink* (Q key).
- Create a Queue, place it to the right of the conveyors, and name it *ConveyorQueue*.
- Connect the three conveyors to *ConveyorQueue* (A key).
- Connect *ConveyorQueue* to *Sink* (A key).



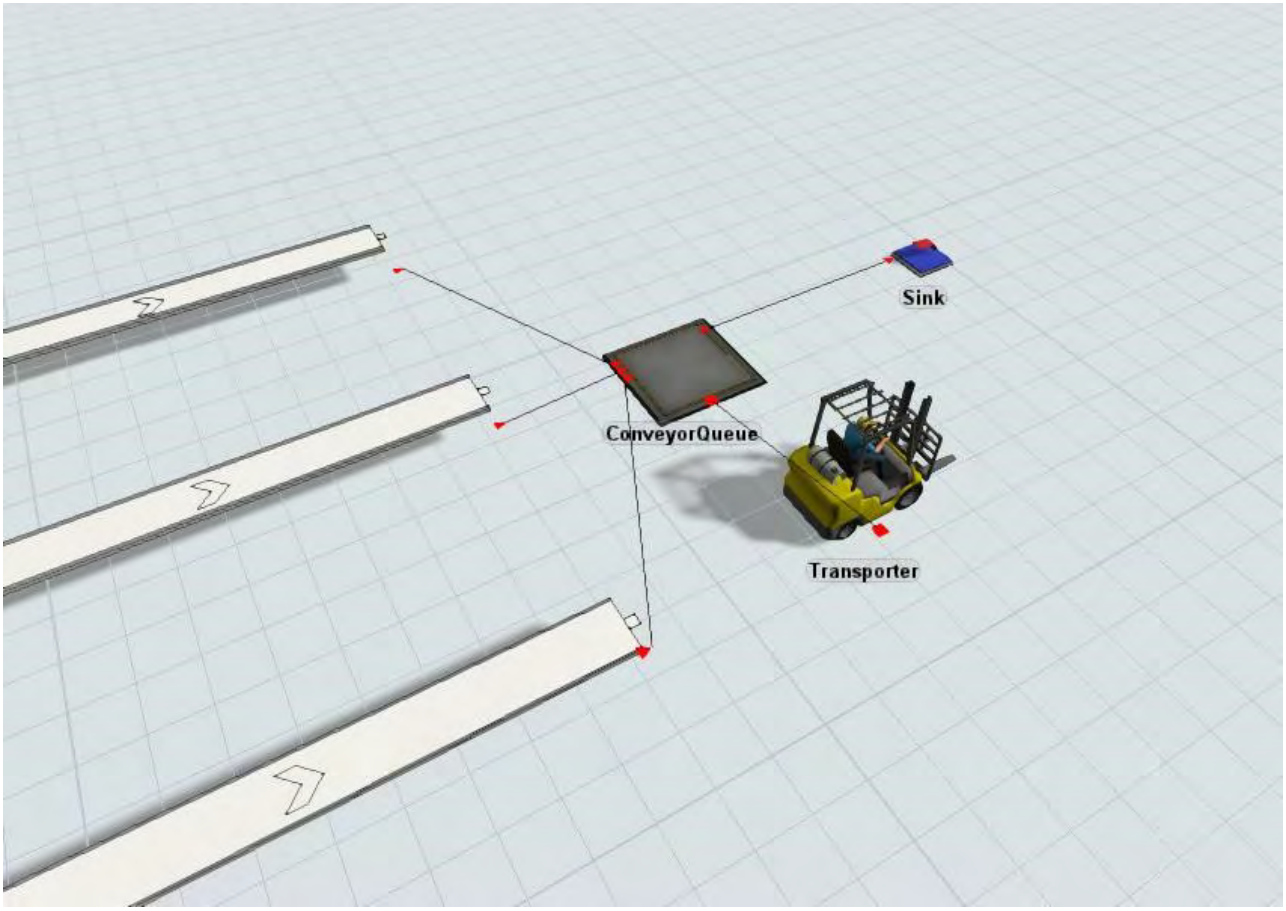
Now that the layout has been revised and the port connections created, the fork truck can be added.

Step 8: Adding the Transporter

•
•

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

- Move *Sink* to the right about 10 units to simulate travel distance.
  Create a Transporter, place it near *ConveyorQueue*, and name it *Transporter*.
  Connect *ConveyorQueue* to *Transporter* with a center connection (S key).



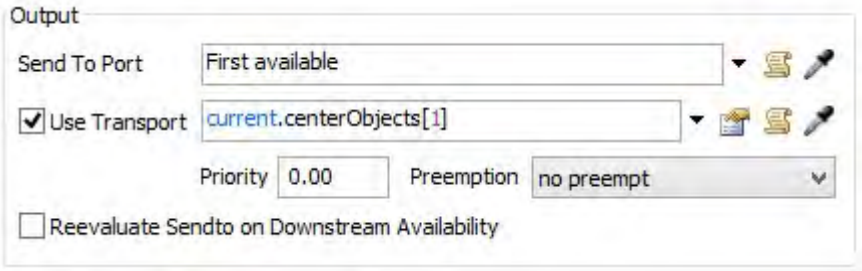
Step 9: Adjust the Queue's Flow Parameters to Use the Transporter

You may define the Queue's flow and transport options by clicking on the Queue in the 3D view to display its properties in the Quick Properties window.

Alternatively, you can open the object's properties window.

- Double-click *ConveyorQueue* to open its Properties window.
- Click the Flow tab and check Use Transport. The central port 1 of the queue is already connected so there is no need for any adjustments.

•
•

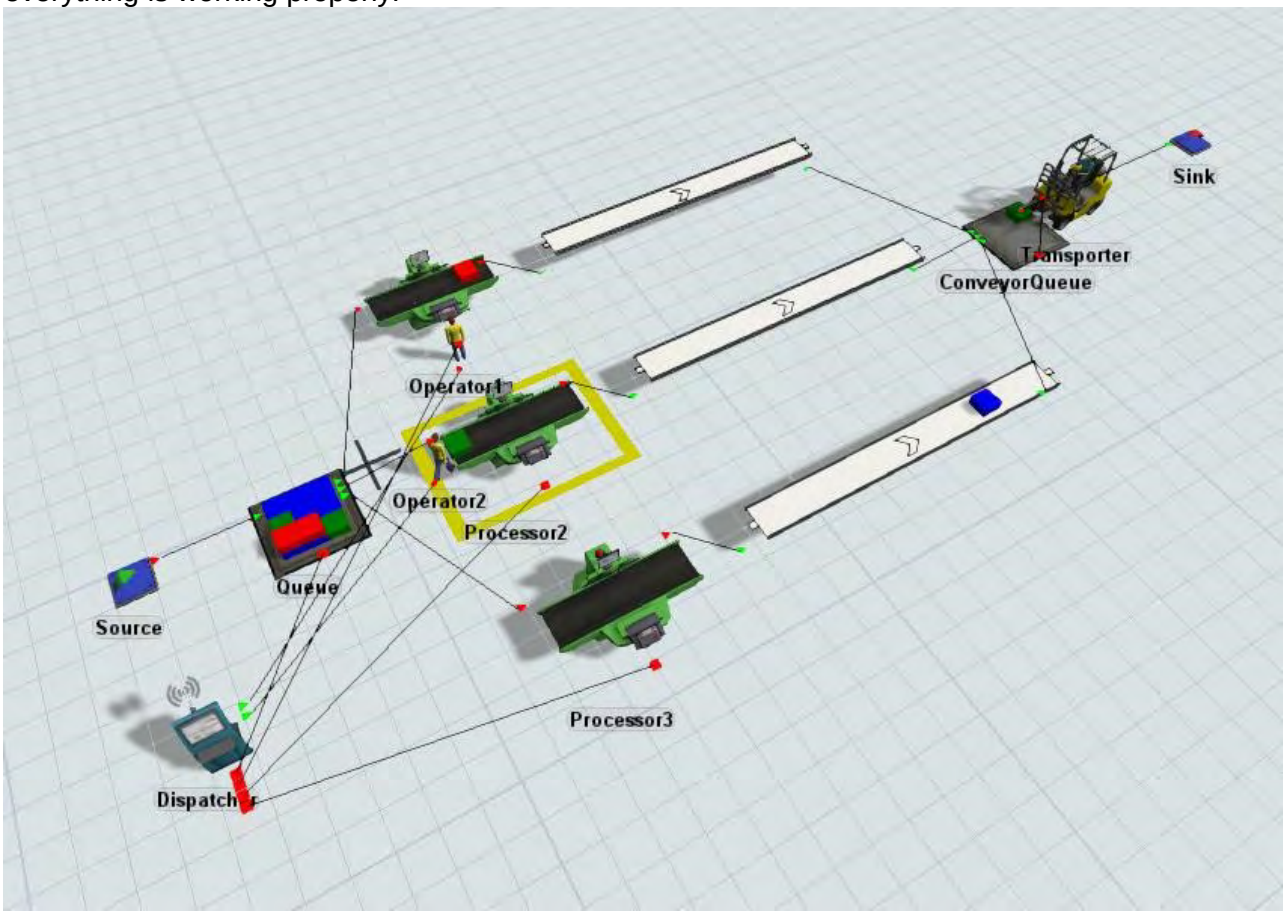


Click OK to close the Properties window.
Reset and Save the model.

Step 10: Run the Model

- Run the model.

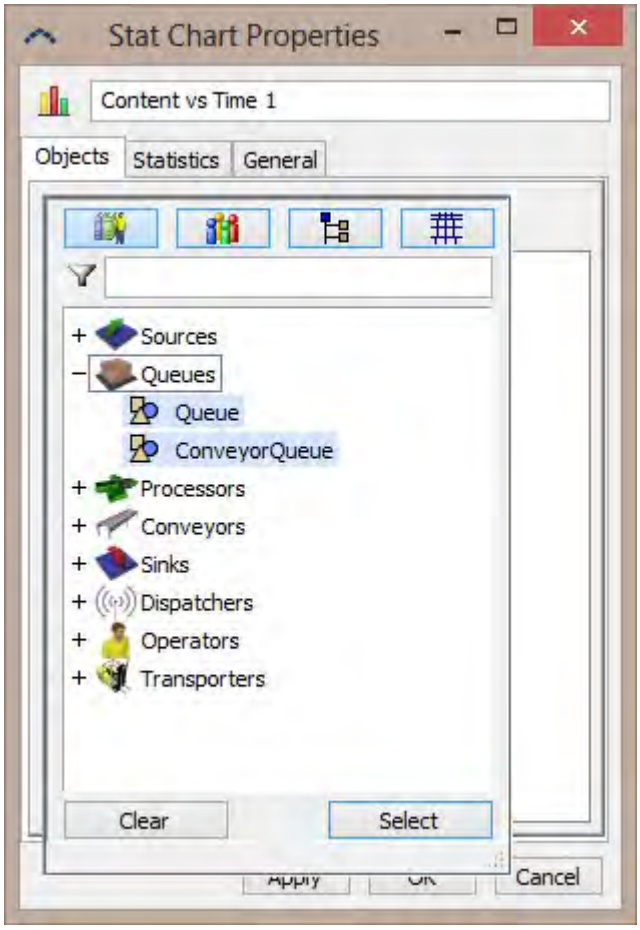
This is the rewarding part of building the model. It's time to check the model to see if it is working the way you want. While the model is running, you can use the animation to visually inspect the model to see if everything is working properly.



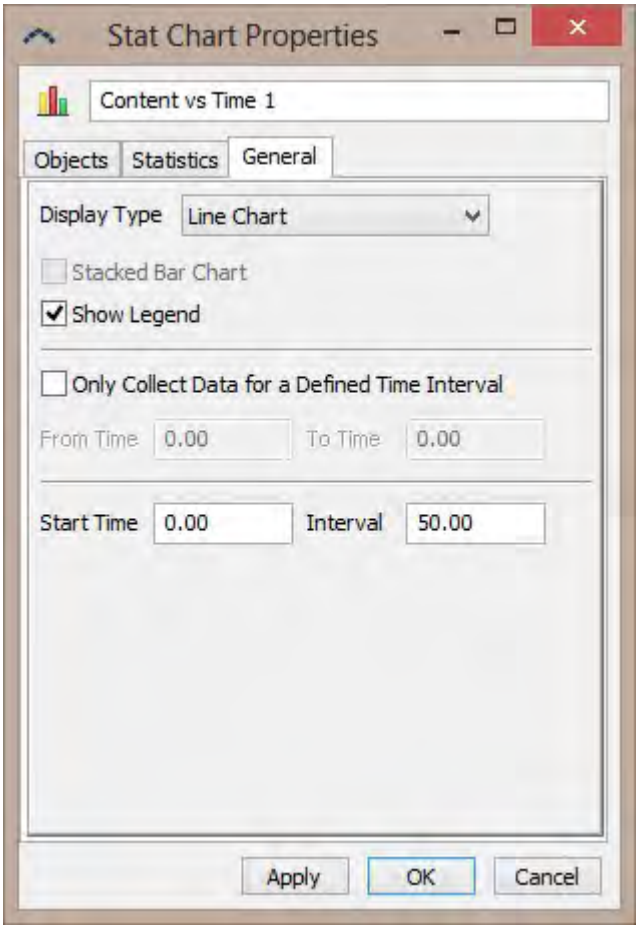
You should see the operators moving back and forth between the queue and the processors. The transporter should be moving flowitems from the queue to the sink. You will notice that when a processor is waiting for an operator to perform the setup, a yellow square is shown under the processor.

Step 11: Analyzing the Output

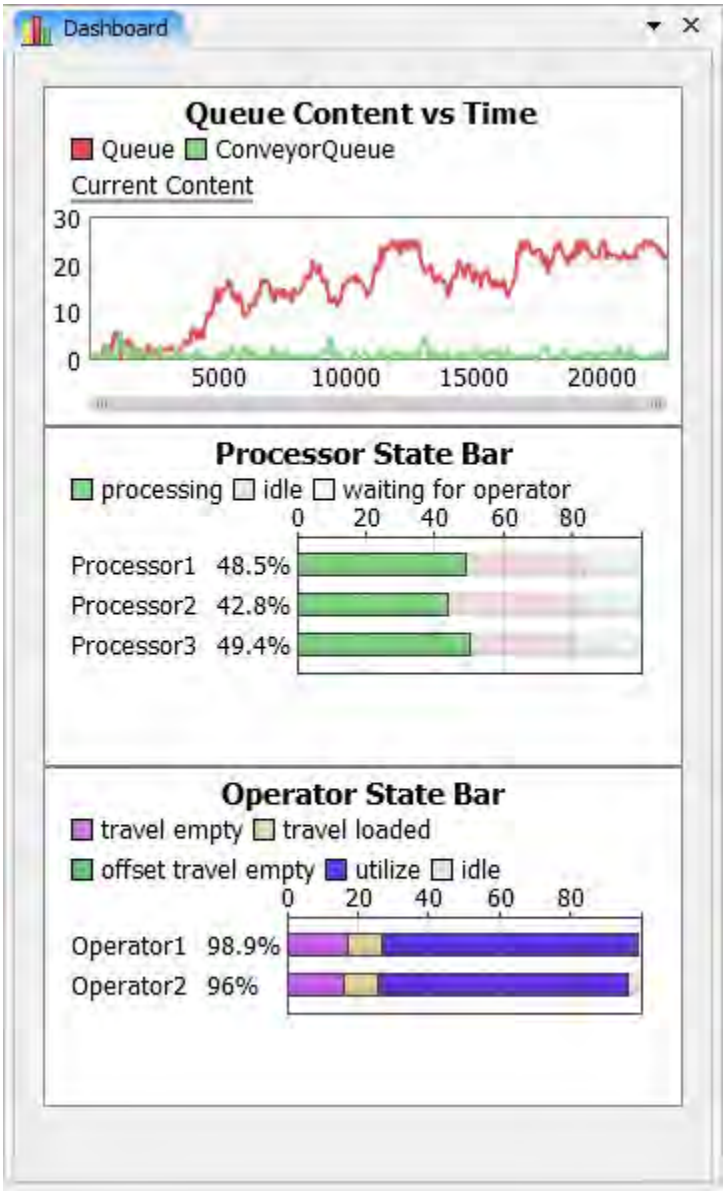
- Select Dashboard > Add a dashboard from the FlexSim Toolbar. The dashboard window will appear.
- Add a line graph by clicking the  button. Select Content vs Time. The object selection window will appear.
- Click the to add objects to the graph. Add *Queue* and *ConveyorQueue*. Click Select .
- Change the chart name to Queue Content vs Time and click OK. A blank chart should appear in the dashboard.



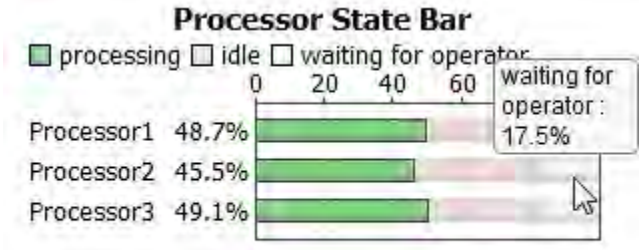
- Add a legend by double-clicking the chart in the dashboard. Select the General tab and check the Show Legend checkbox. Click OK to return to the dashboard.



- Now add a bar chart by clicking the button and selecting the State Bar option.
- Add all of the processors to the chart.
- Change the chart name to Processor State Bar and click OK. Another blank chart should appear in the dashboard.
- Add another state bar chart to the dashboard for the operators. Follow the same steps as above, but add all the operators to the chart, instead of processors. Change the chart name to Operator State Bar and click OK.
- Reset and Run the model. The graphs will dynamically update.

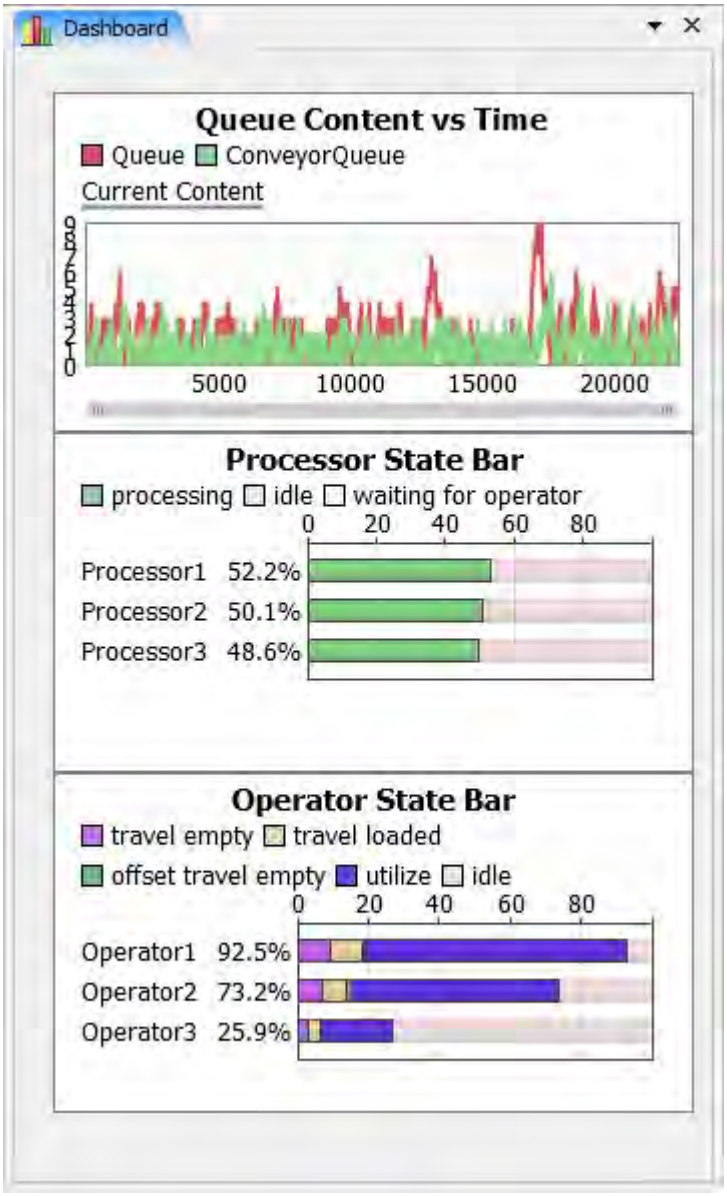


- Hold the mouse over one of the bars in the Processor State Bar graph. A ToolTip will appear and give more information.



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.

- Create an Operator, place near the other two, and name it *Operator3*.
- Connect *Dispatcher* to *Operator3* (A key).
- Double-click the *Operator State Bar* chart and add *Operator3*.
- Reset, Save, and Run the model.



With one more operator, the processor wait time goes down and that the *Queue* stays a a much lower content level.

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 Tutorial 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 charts, graphs, and 3D text to the model you finished in lesson 2.

What You Will Learn

- How to add a content graph for the Queue
- How to add a histogram to show the wait time for the Queue
- How to add a 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 Dashboard objects.

Approximate Time to Complete this Lesson

This lesson should take about 20-30 minutes to complete. Click here for the Step-By-Step Tutorial.

Lesson 2 Extra Mile Tutorial Step-By-Step Model Construction

Building Model 2 Extra Mile

To start building model 2 extra mile you will need to load model 2 which you saved from the last lesson. If at any time you encounter difficulties while building this model, a fully functional tutorial model can be found at http://www.flexsim.com/tutorials

Step 1: Load Model 2

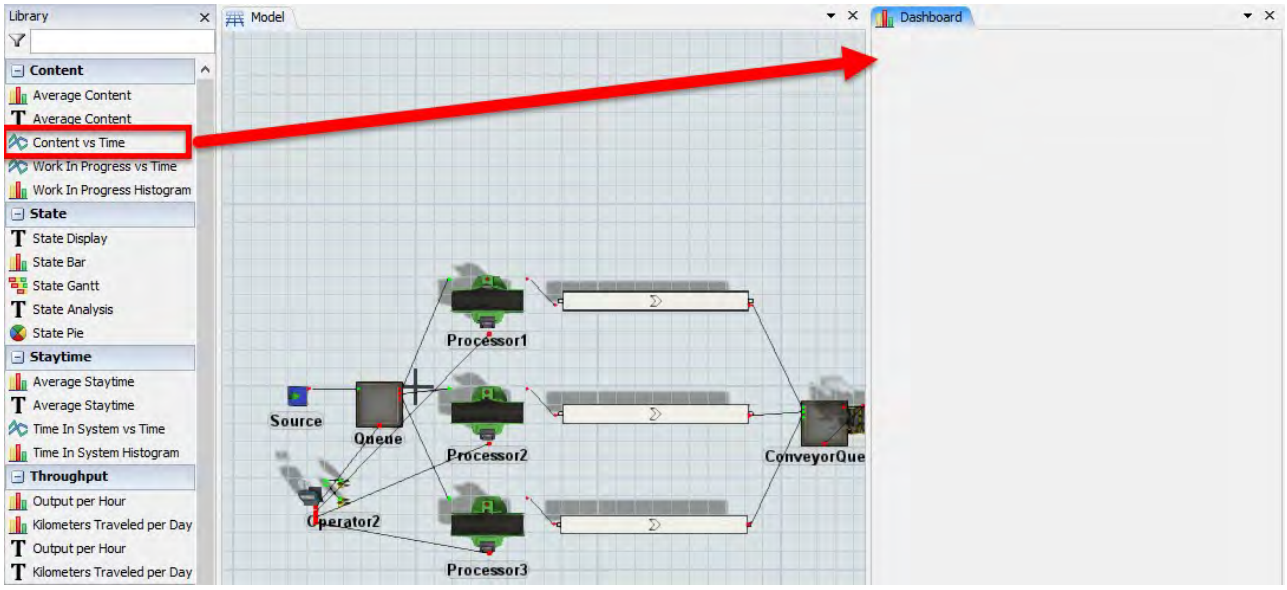
- Load Model 2 if it is not already open.

Step 2: Save Model as "Model 2 Extra Mile"

- Go to the menu option File > Save Model As... to save your model under a new name.

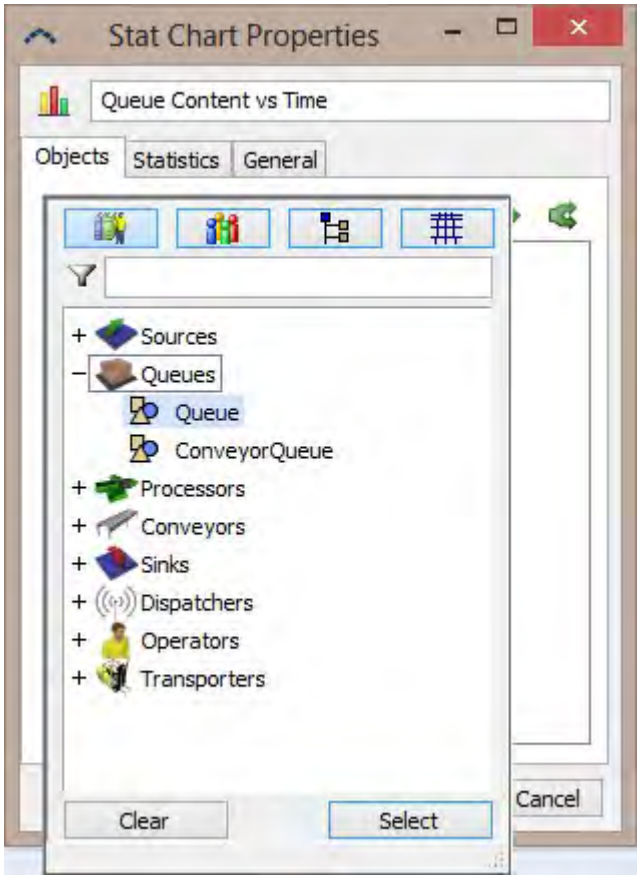
Step 3: Add a Dashboard to Show the Content of the Queue

- Create a Dashboard by clicking the Dashboard button on the FlexSim Toolbar. Drag the "Content vs Time" object from the library into the dashboard pane.

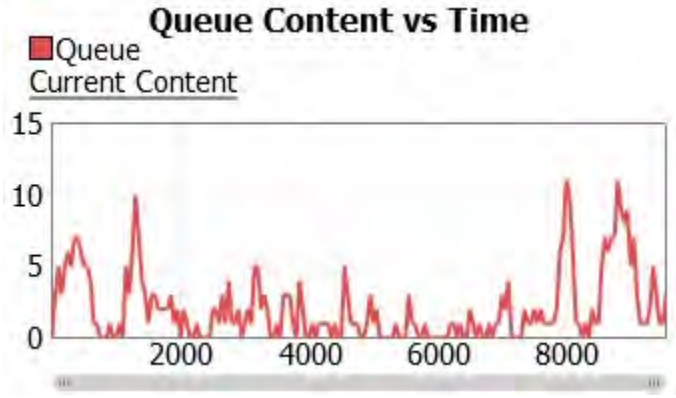


Step 4: Adjust the Parameters of the Dashboard to show a Content Graph of the Queue

- In the Dashboard properties, Push the Green Plus on the Objects tab. You will now be able to select the Queue to show its Content vs Time.
- Push the plus button next to Queues, select Queue. Push the Select button below to complete your selection.
- Change the name of the Dashboard to Queue Content vs Time • Click OK to apply the changes and close the window.

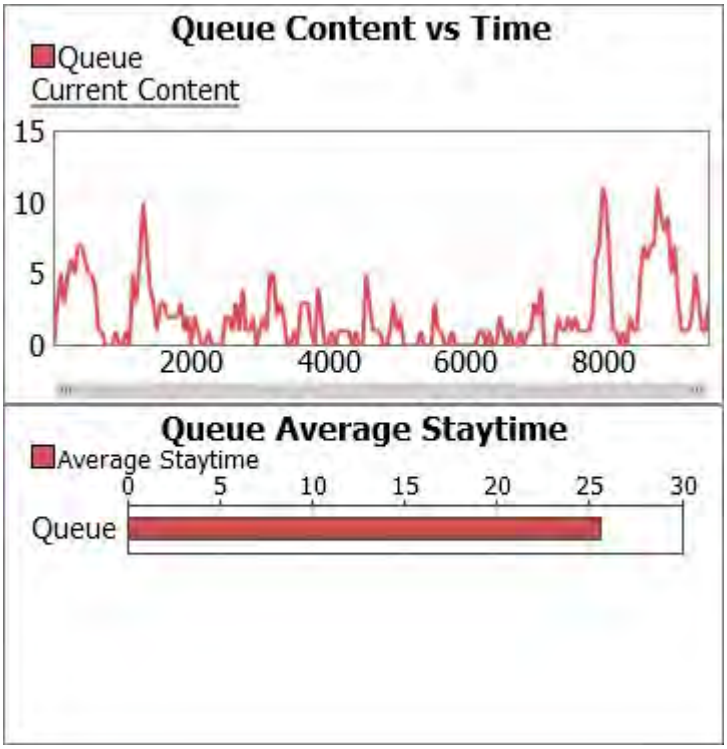


Alternatively, you can click on the  and then click on the Queue in the 3D view to add it to your dashboard.



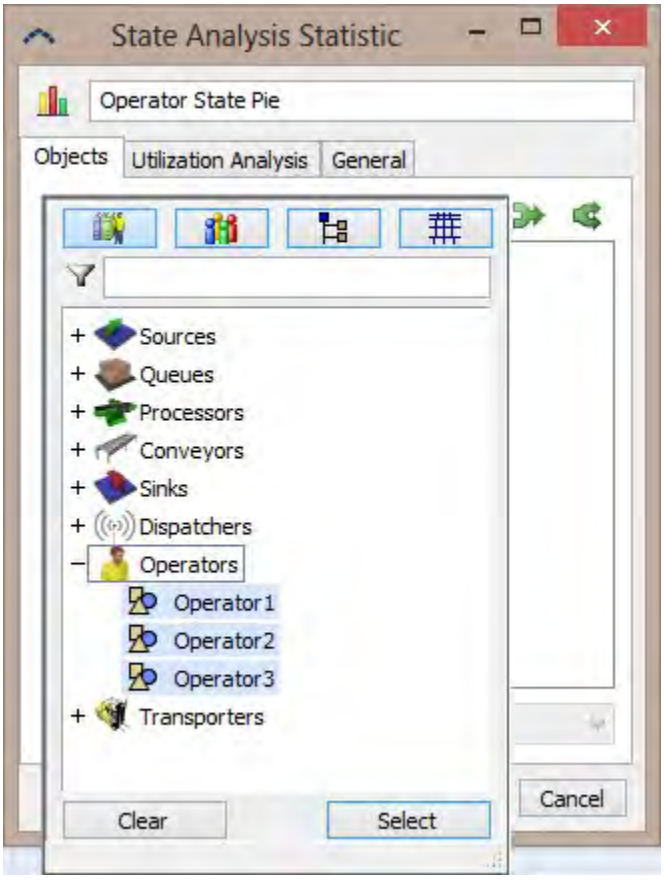
Step 6: Add a Dashboard to show the Average Staytime of the Queue

- Following the same steps from steps 4, add a new Dashboard Chart and place it below the content graph. The only difference will be that you will drag an Average Staytime Bar Graph instead of "Content vs Time" from the Library and the Graph Title will be Queue Average Staytime.

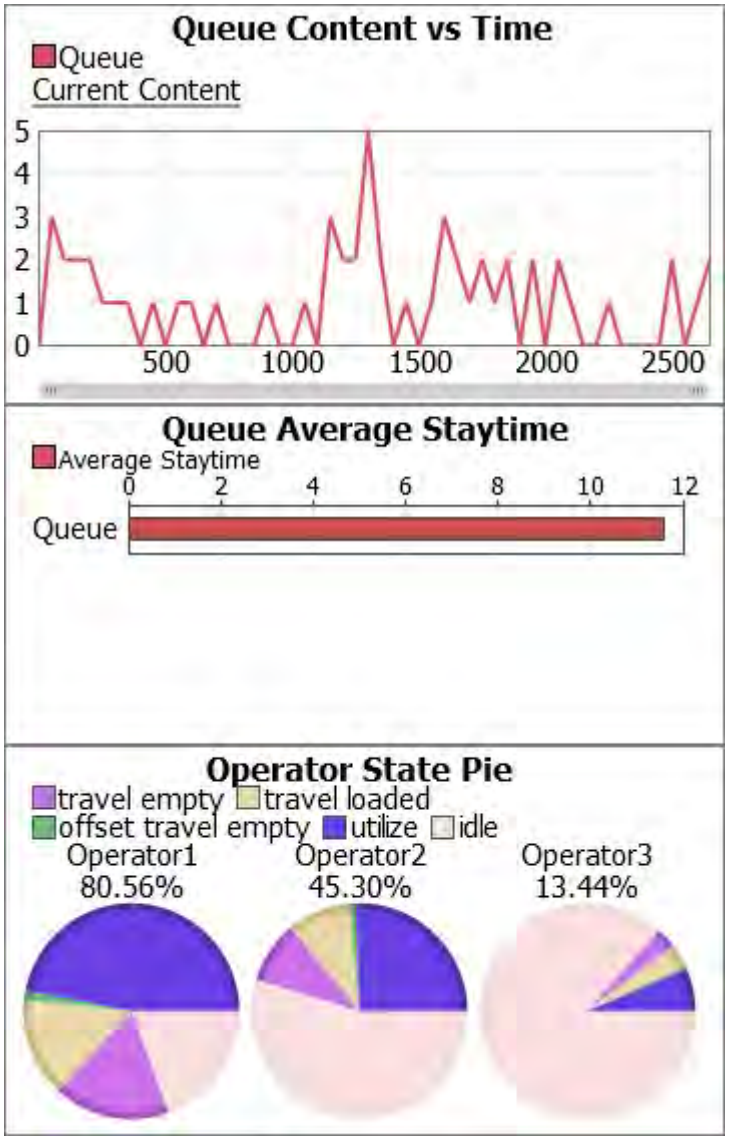


Step 7: Add a State Pie Chart for each Operator

- Following the same steps from step 4, add a new Dashboard Chart for each Operator. Select State Pieinstead of "Content vs Time" from the Library and set the Graph Title to be Operator State Pie.



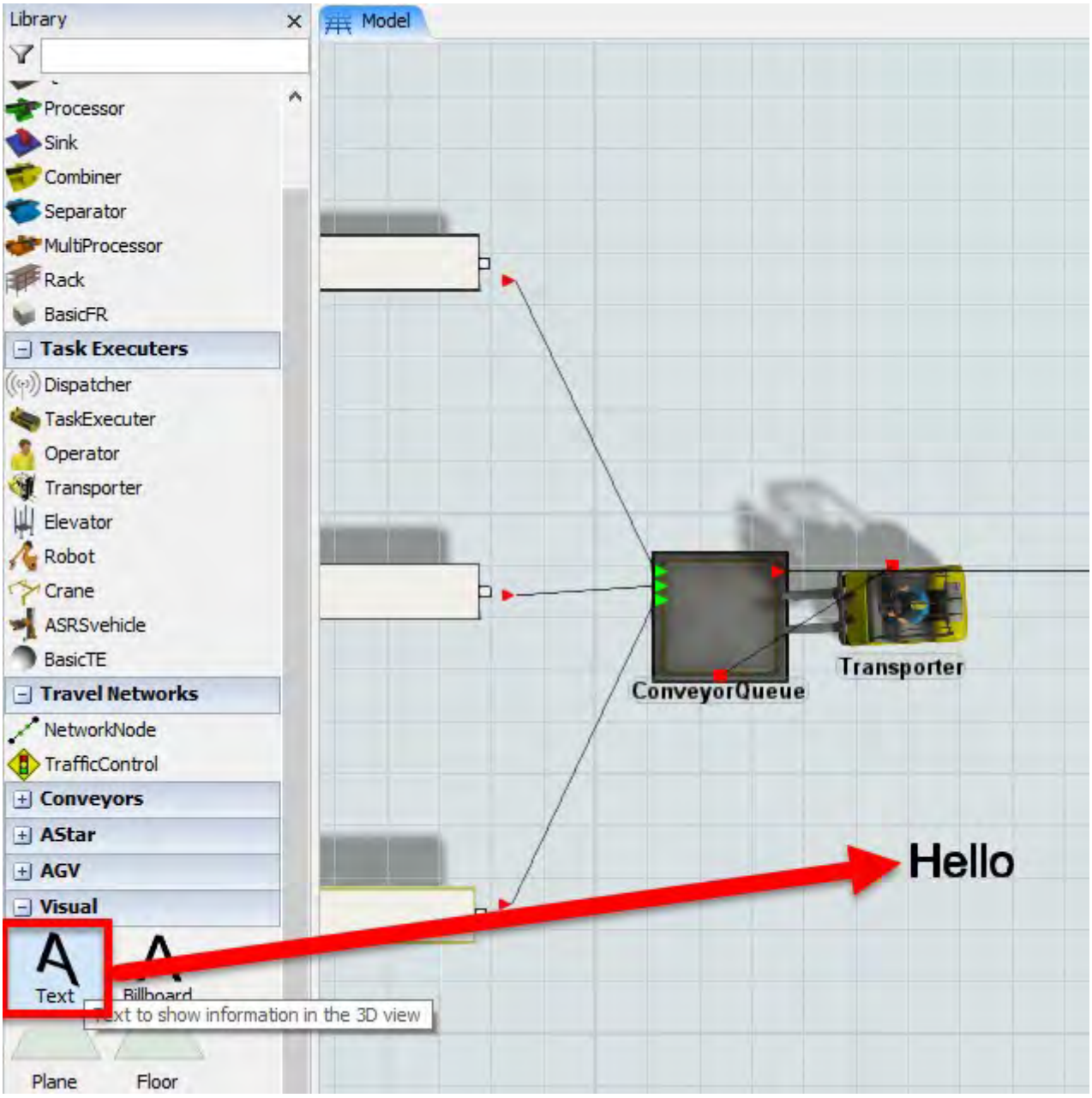
When reset and run, the pie charts should look something like this:



Step 8: Adding 3D Text to the Model

There is another way to add information to the model that can show performance measures. You can place 3D text at strategic points in the layout to show what is happening while the model is running. This is done using a VisualTool. In this model we will add 3D text to show the average wait time of flowitems in the "Conveyor Queue".

- From the library under the Visual group, add a VisualTool by dragging a Text object into the model. Place it by ConveyorQueue, and name it *Text*.

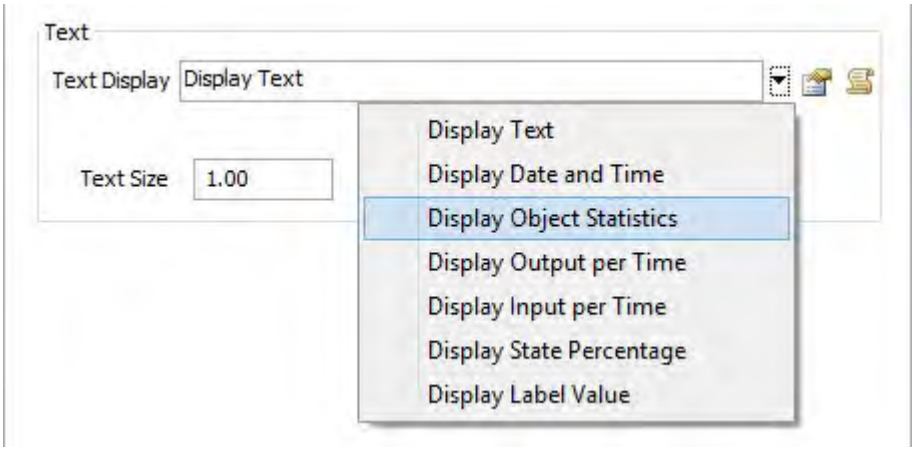


- Double-click the VisualTool to bring up its Properties window (see Figure 2-45).
- In the Text Display list select the Display Object Statistics option. The code template window will appear. Change the parameters text to be as follows:

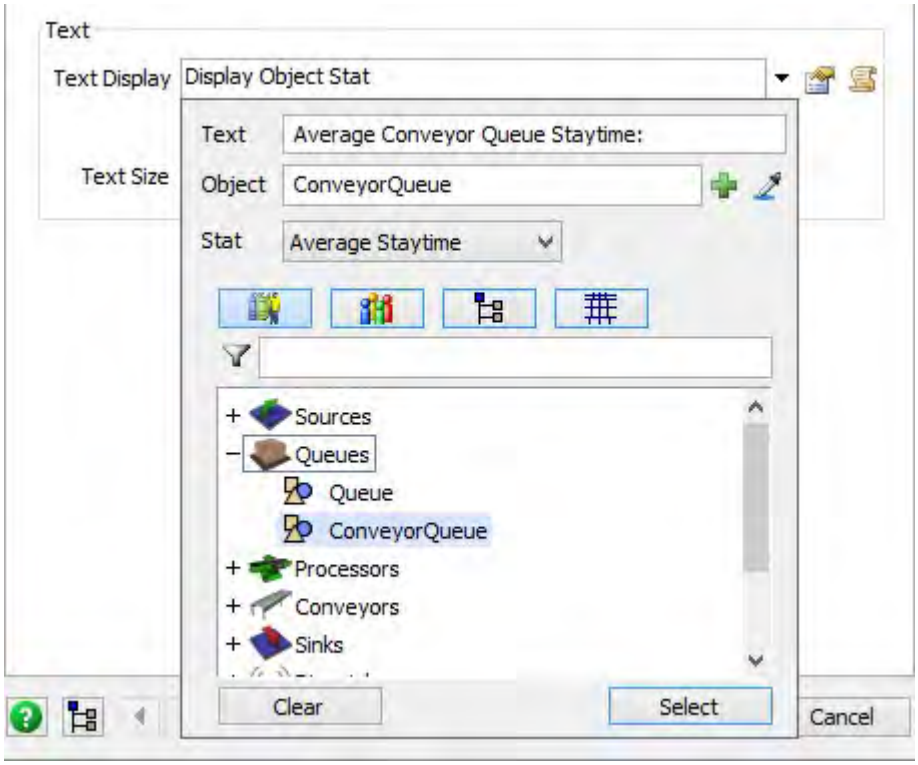
Text: "Average Conveyor Queue Staytime: "

Object: ConveyorQueue

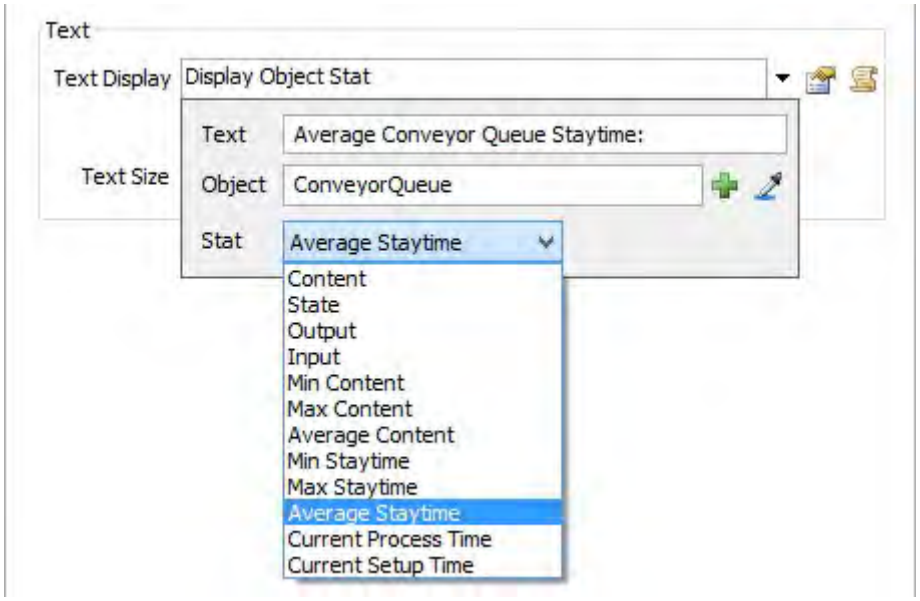
Stat: Average Staytime



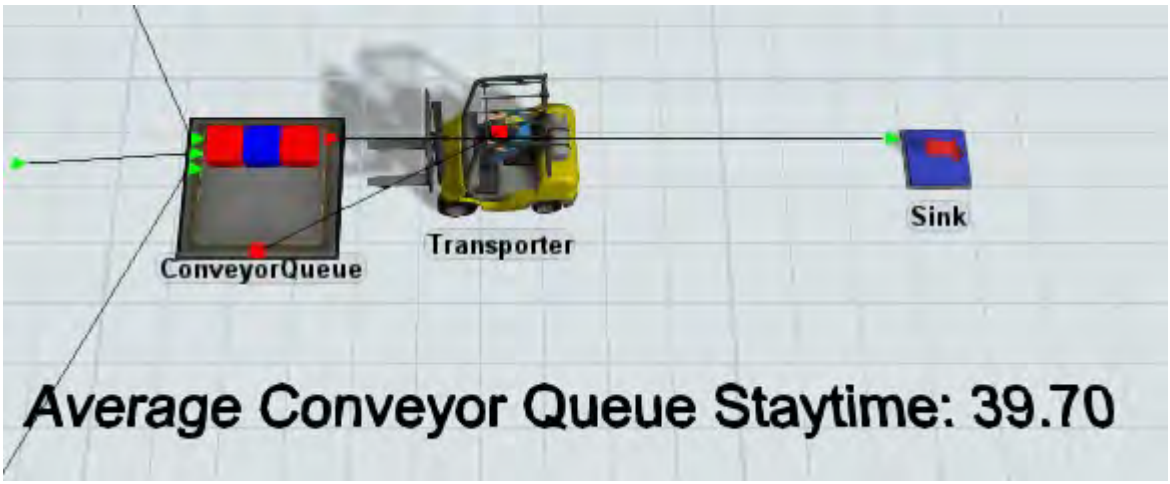
- For the Object: value, click the Green Plus sign, then click on the plus sign next to Queues, and select Average Staytime. Then press Select.



- For the Stat: value, click the Dropdown Menu, and select Average Staytime.

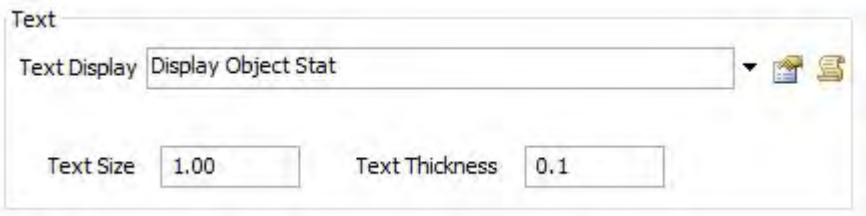


NOTE: To click on the VisualTool, click directly on the 3D text that is showing.

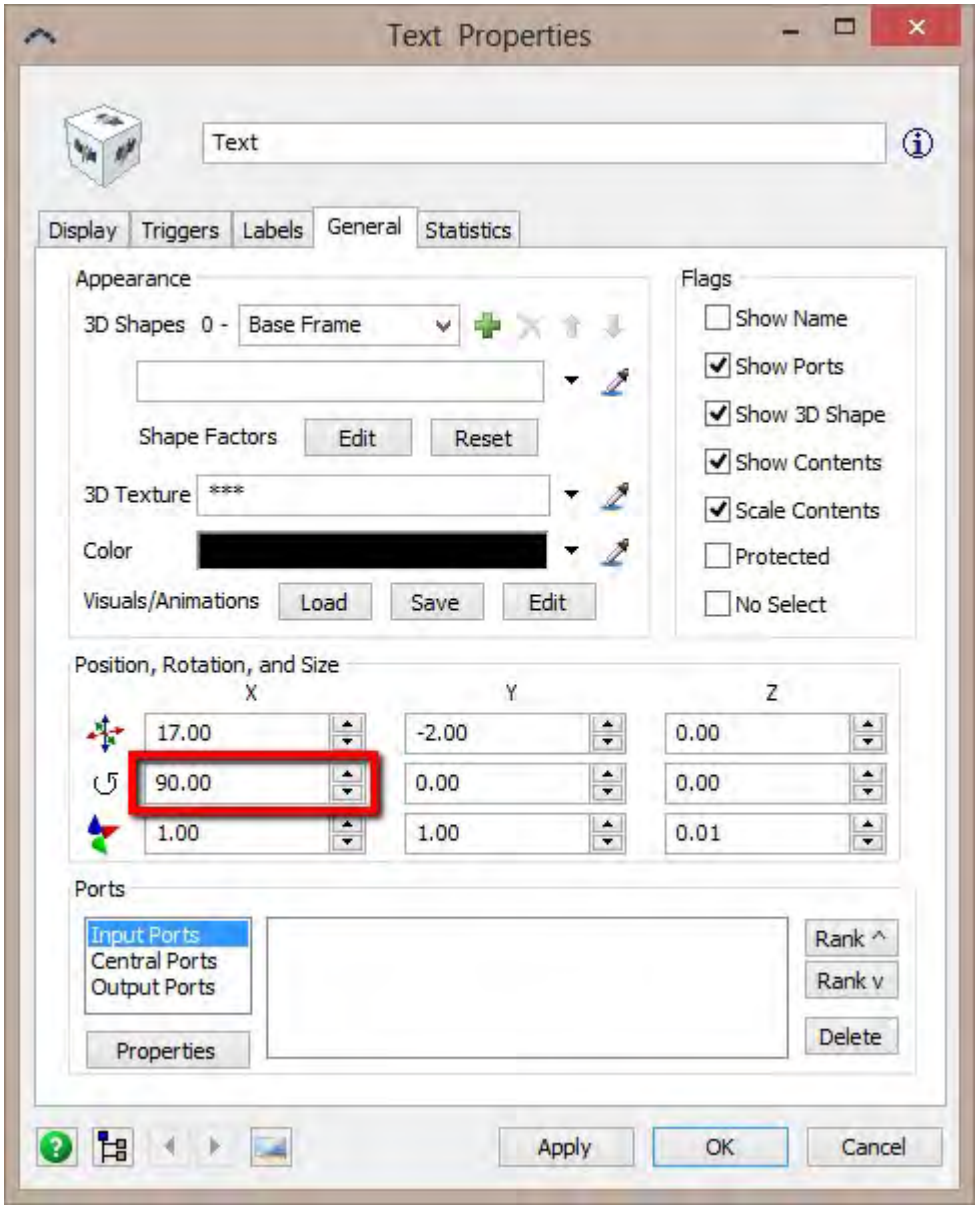


At this point you may want to adjust the display of the text. The text size is set to 1 by default, and you may want to make it smaller. You may also want to have the text hover over the queue. To make the text smaller, type the desired size in the Properites window of the VisualTool. You may also adjust the thickness to give the text a 3D appearance.

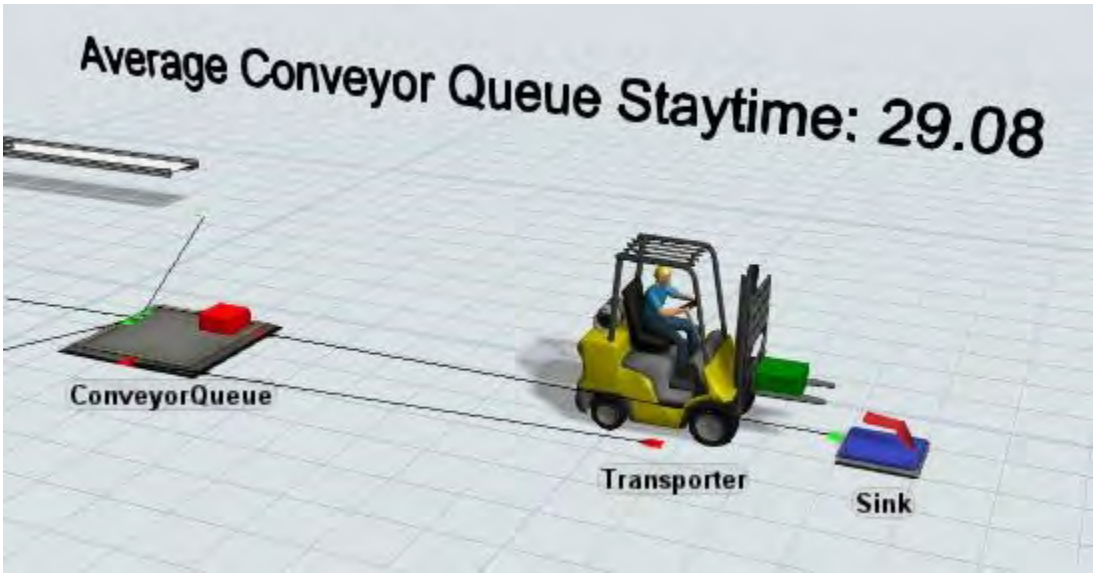
- Double-click the VisualTool to open its Properties window. On the Display tab, change Text Thickness to 0.1.



- Click the General tab.
- Change Rotation X to 90.
- Click OK to close the Properties window and apply the changes.

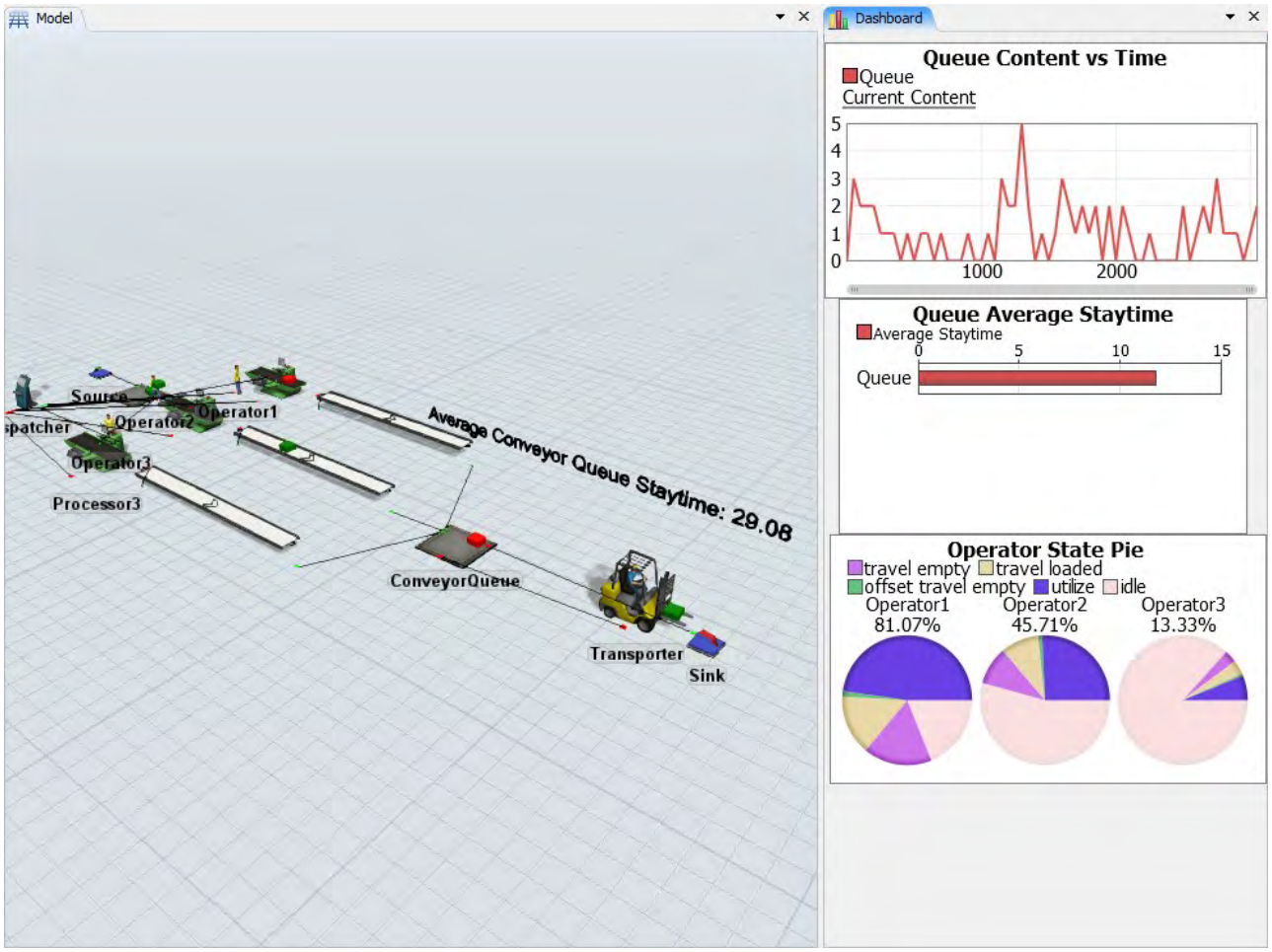


The text will now be rotated in the model. Use you mouse to select and position the text as desired. Remember that the z-position of the text can be controlled by selecting the text with both the left and right mouse buttons and moving the mouse forward and back, or selecting the text and then rolling the mouse wheel to move the text up or down.



Step 9: Reset, Save and Run

- 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.



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 Tutorial 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, which allows you to do multiple runs 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 properties 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 properties will still be identified in the step-by-step description, but screen shots may not be provided.

Note: If you are using the Evaluation version of FlexSim, you will not be able to complete this model. This lesson exceeds the number of allowed objects in the Evaluation version.

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 and NetworkNode objects as well as Spline Points.

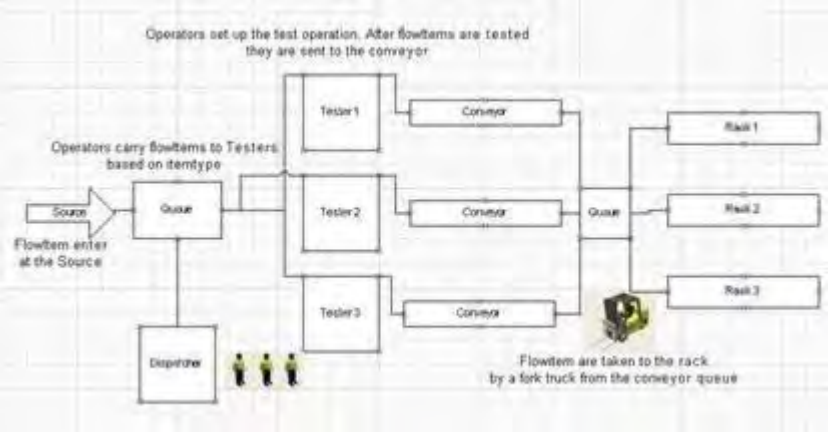
Approximate Time to Complete this Lesson

This lesson should take about 45-60 minutes to complete

Model 3 Overview

In model 3 the sink will be replaced with 3 racks that will be used to store the completed flowitems prior to shipping. 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 type 1 flowitems will be sent to rack 2, all type 2 flowitems will be sent to rack 3, and all type 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.



Model 3 Data

Add curved sections to the first and third conveyors 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:

- Type 1 to rack 2
- Type 2 to rack 3
- Type 3 to rack 1

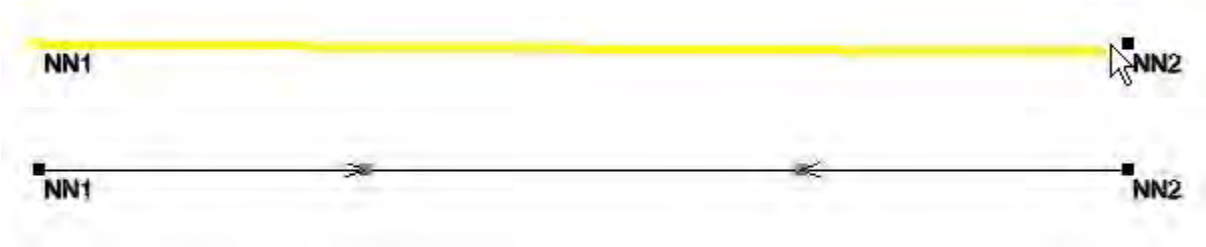
Set up a path network for the fork truck to travel on between the conveyor queue and the racks. Set up a flypath for a fly-through model presentation.

New Concepts

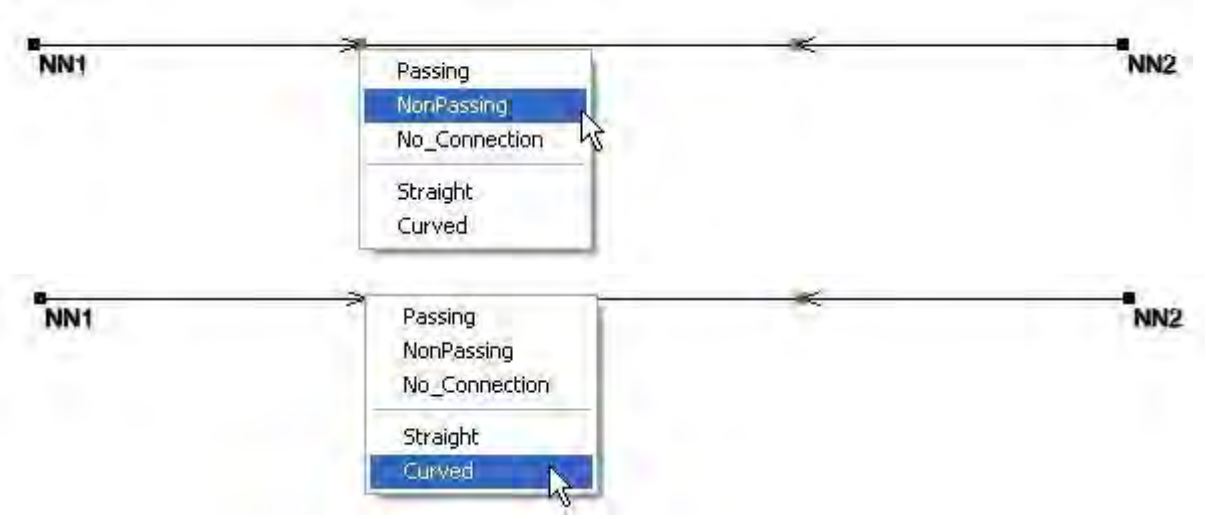
Spline Control Points

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. There are also two green boxes with arrows at about 1/3 and 2/3 of the way down the line.

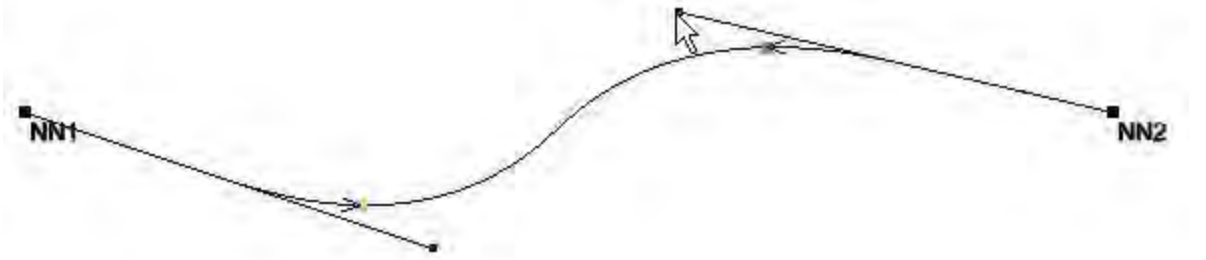


These green boxes indicate the attributes of the path going in the indicated direction. Green means it is a passing path, yellow means it is a nonpassing path, and red means it is a "no connection" or in other words it is a one-way path going the other way. To switch between these colors, you can right click on the node and select an option. 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 Menu.

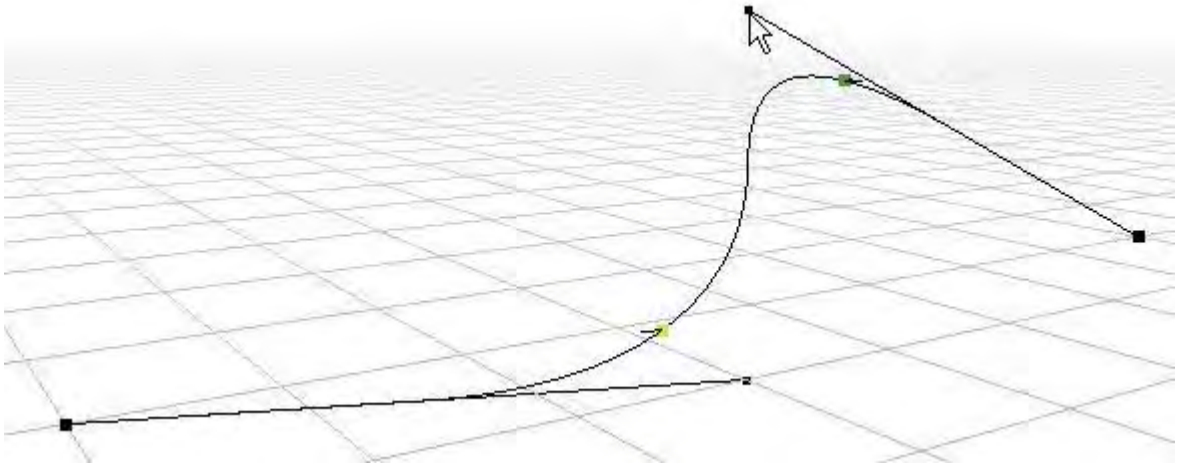


Spline Control Point parameters

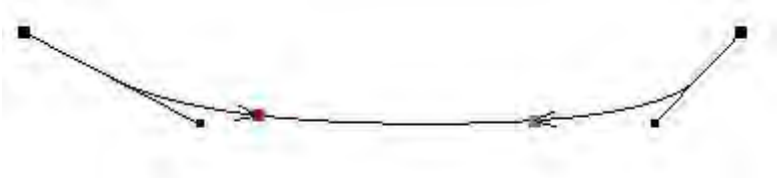
Once you have created a curved path, move the small control points with the mouse.



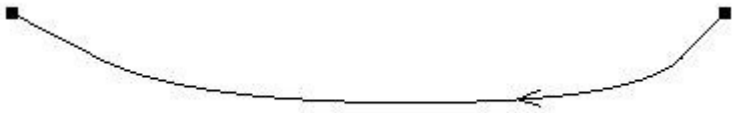
To change the Z height of the spline control point, click on it and roll the mouse wheel up and down.



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.



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 NetworkNodes in the path network. Multiple "X" clicks will toggle between several different visual modes for the network.



Click here for the Step-By-Step Tutorial.

Lesson 3 Tutorial Step-By-Step Model Construction

Building Model 3

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

If at any time you encounter difficulties while building this model, a fully functional tutorial model can be found at http://www.flexsim.com/tutorials

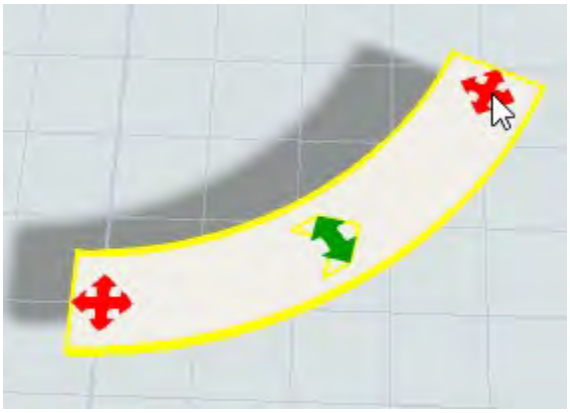
Step 1: Load Model 2

- Load model 2 if it is not already open.

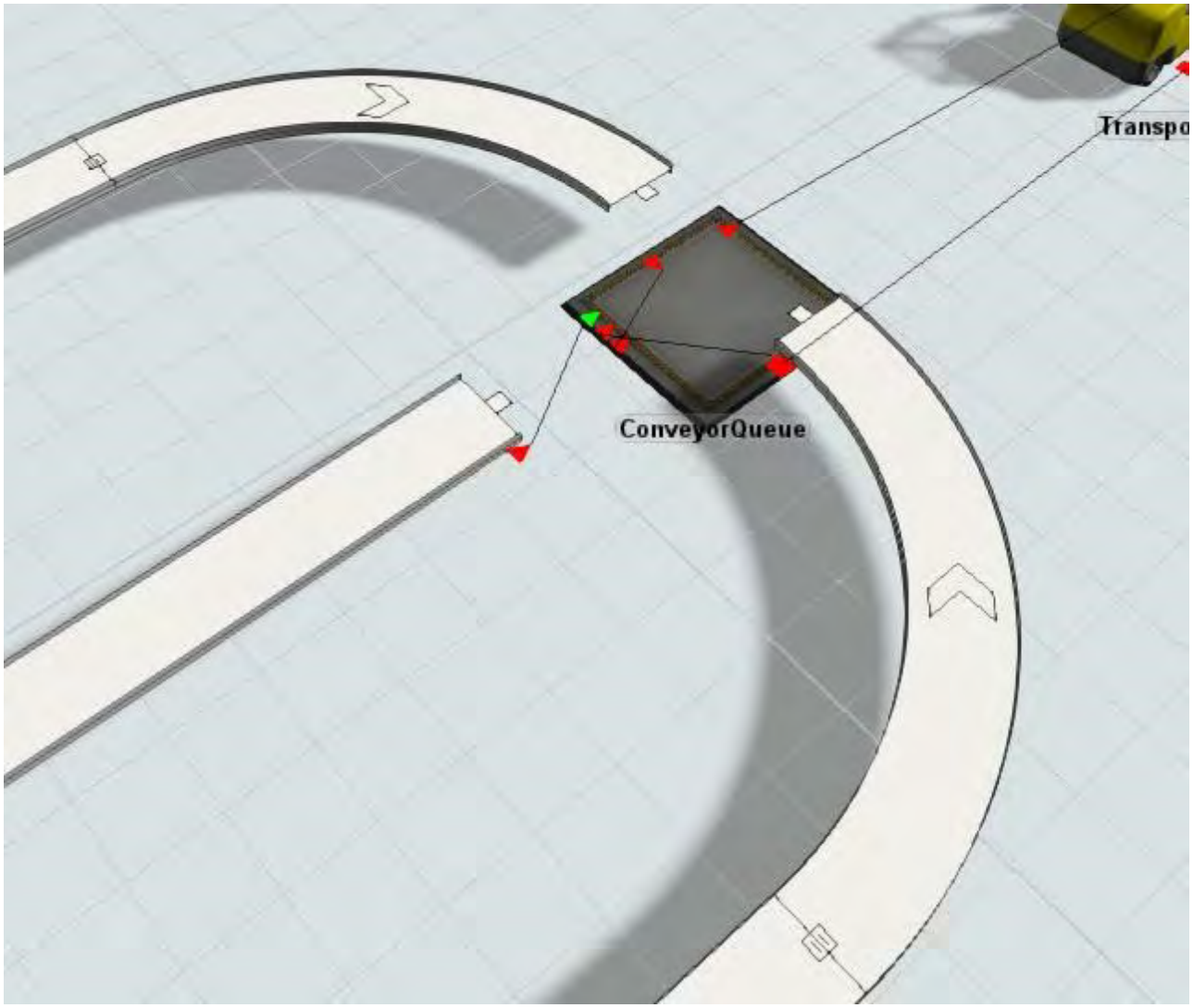
Step 2: Add curved conveyor sections from the outside conveyors that feed items closer to ConveyorQueue.

Now we will add curved sections to the two outside conveyors.

- Disconnect the top and bottom conveyors from ConveyorQueue with a 'Q' disconnect.
- From the Library's Conveyors group, drag two Curved Conveyors into the model and place them next to the top and bottom conveyors respectively.
- Adjust the direction of the curve for the top curved conveyor. Click on the top conveyor, then click and drag the red four-way arrow on the conveyor's end point. Drag it around so that the conveyor makes a 90 degree turn right instead of left. You can also drag the green arrow to adjust the radius of the curved conveyor.



- Move each curved conveyor so that its start point is aligned with the end point of the top and bottom conveyors respectively. The conveyors should snap to the end of the straight sections. When you release the cursor, a transfer will be created between the two conveyors. This is a square drawn at the intersection point, signifying that it is a transfer point between the two conveyors.
- Recreate the connection from the new curved conveyors to ConveyorQueue with an 'A' connection.



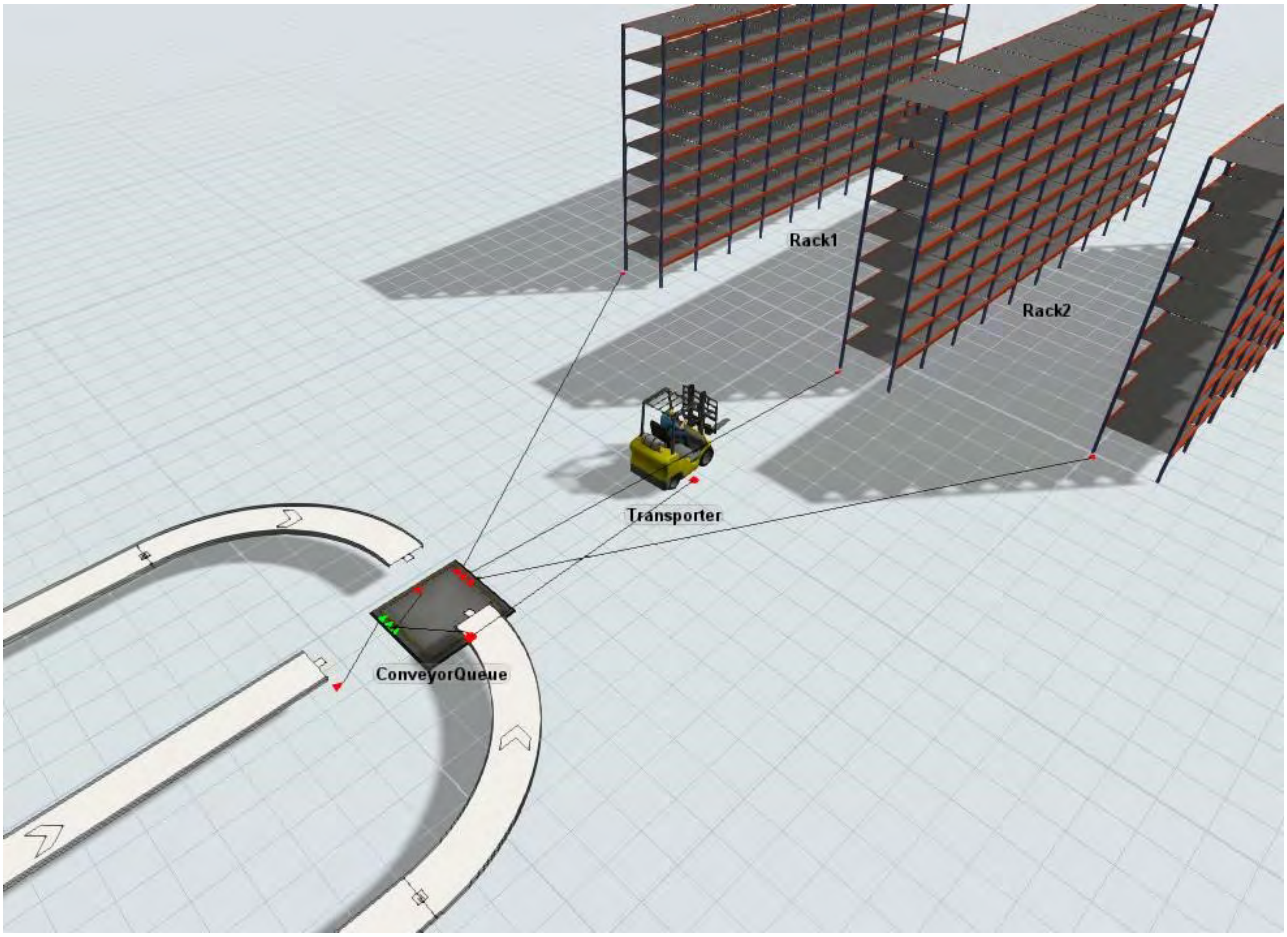
Step 3: Delete the sink

- Highlight the *Sink* and press the Delete key.

When an object is deleted, all port connections to and from that object are deleted as well. Be aware that this may affect the port numbering of those objects that were connected to the deleted object.

Step 4: Create three Racks

- Create three Racks, place them to the right of *ConveyorQueue*, and name them *Rack1, Rack2,* and *Rack3*. Place the racks far enough away from the queue to allow the fork truck some travel distance to reach the racks.
- Connect *ConveyorQueue* to *Rack1, Rack2,* and *Rack3* (A key).

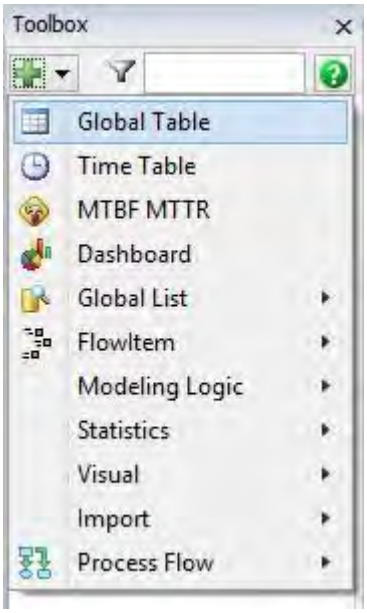


Step 5: Create a Global Table to Control Flowitem Routing

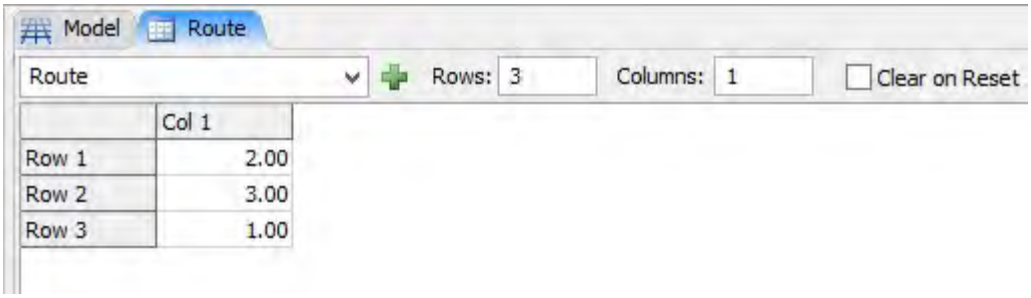
The next step is to set up a global table that will be used to reference which rack each flowitem will be sent to (or more accurately stated, which output port of the conveyor queue the flowitems will be sent through). It is assumed that output port 1 was connected to *Rack1*, port 2 to *Rack2*, and port 3 to *Rack3*. If the connections are not in the correct order, you can modify them through the queue's properties window on the General tab in the Ports section.

We will send all item type 1s to *Rack2*, all item type 2s to *Rack3*, and all item type 3s to *Rack1*. Here are the steps to setting up a global table:

- Add a new Global Table from the Toolbox (View > Toolbox or click on the Tools button in the main toolbar).



- Change the Name to Route.
- Set Rows to 3 and Columns to 1.
- Double click on the row names (Row 1, Row 2, Row 3) and 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 to.
- Click the Close button to apply the changes and close the table.



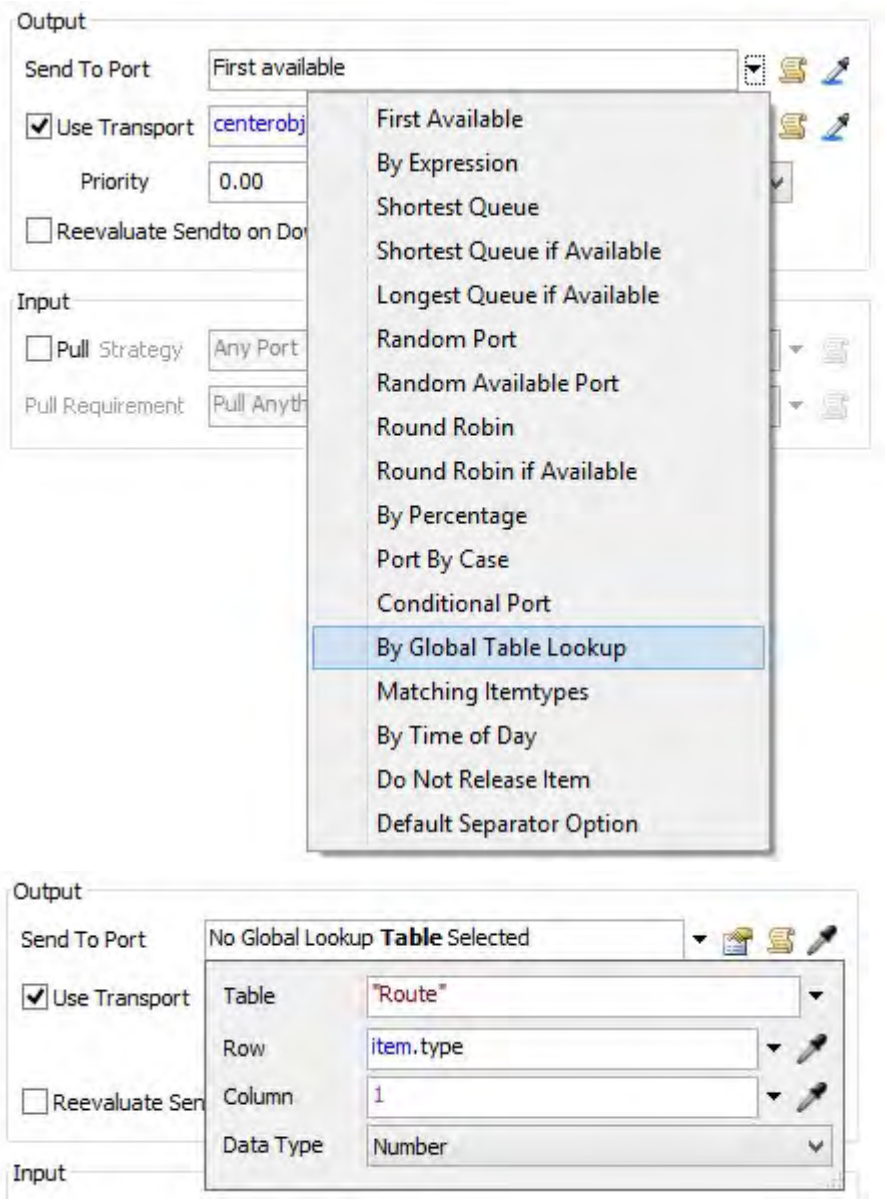
Now that the global table is defined, we can adjust the "Send To Port" option on the queue.

Step 6: Adjusting the Send To Port Option on the ConveyorQueue

You may define the ConveyorQueue's flow and transport options by clicking on the Queue in the 3D view to display its properties in the Quick Properties window.

Alternatively, you can open the object's properties window.

- Double-click on *ConveyorQueue* to bring up its Properties window.
- Click the Flow tab. Select the option By Global Table Lookup from the Send To Port list. The code template window will appear. Edit the options to read as follows:



- Click the OK button to close the Properties window.

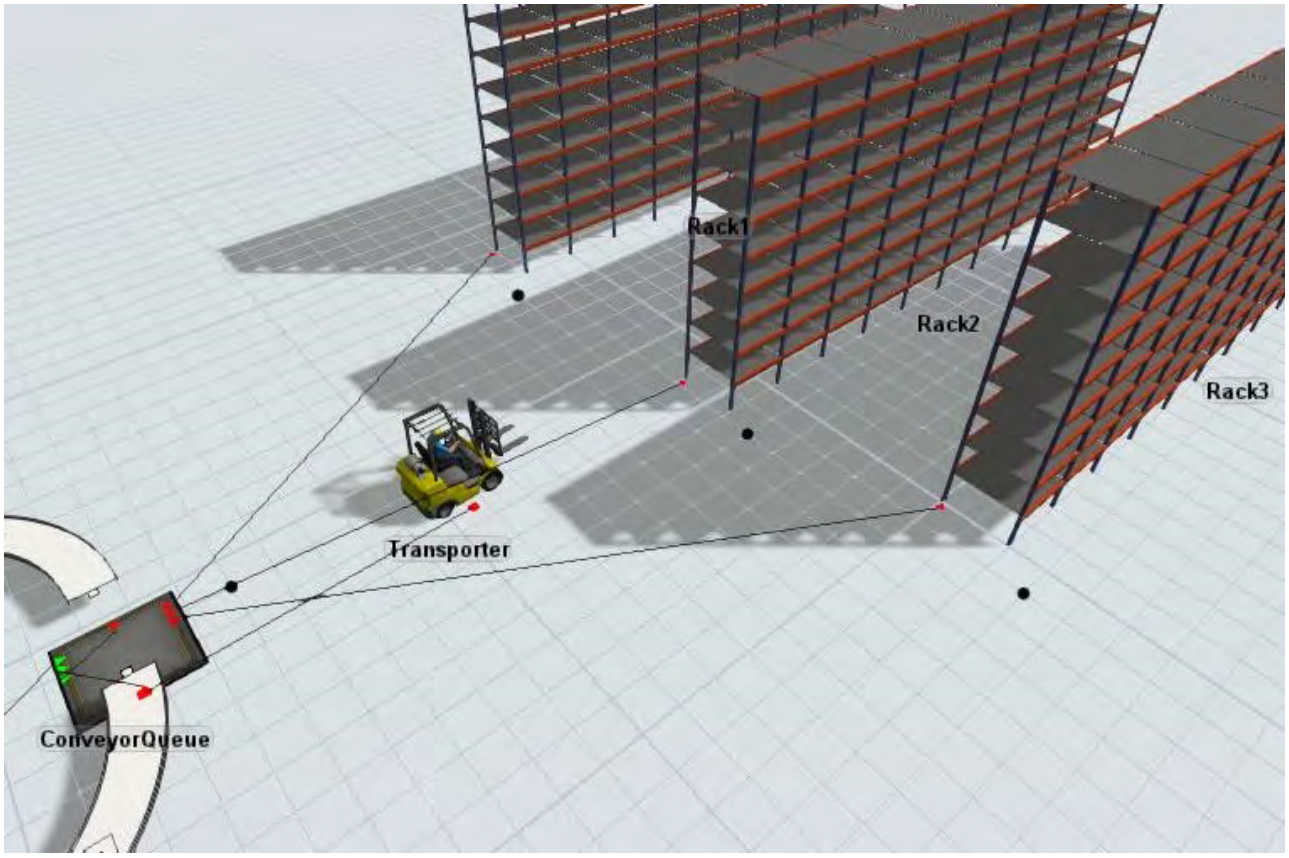
Step 7: Reset, Save, and Run

At this point it would be wise to Reset, Save the model, and then Run the model to verify that the changes are working correctly. The model should run with the fork truck transporting flowitems to the racks based on the item type definition in the global table.

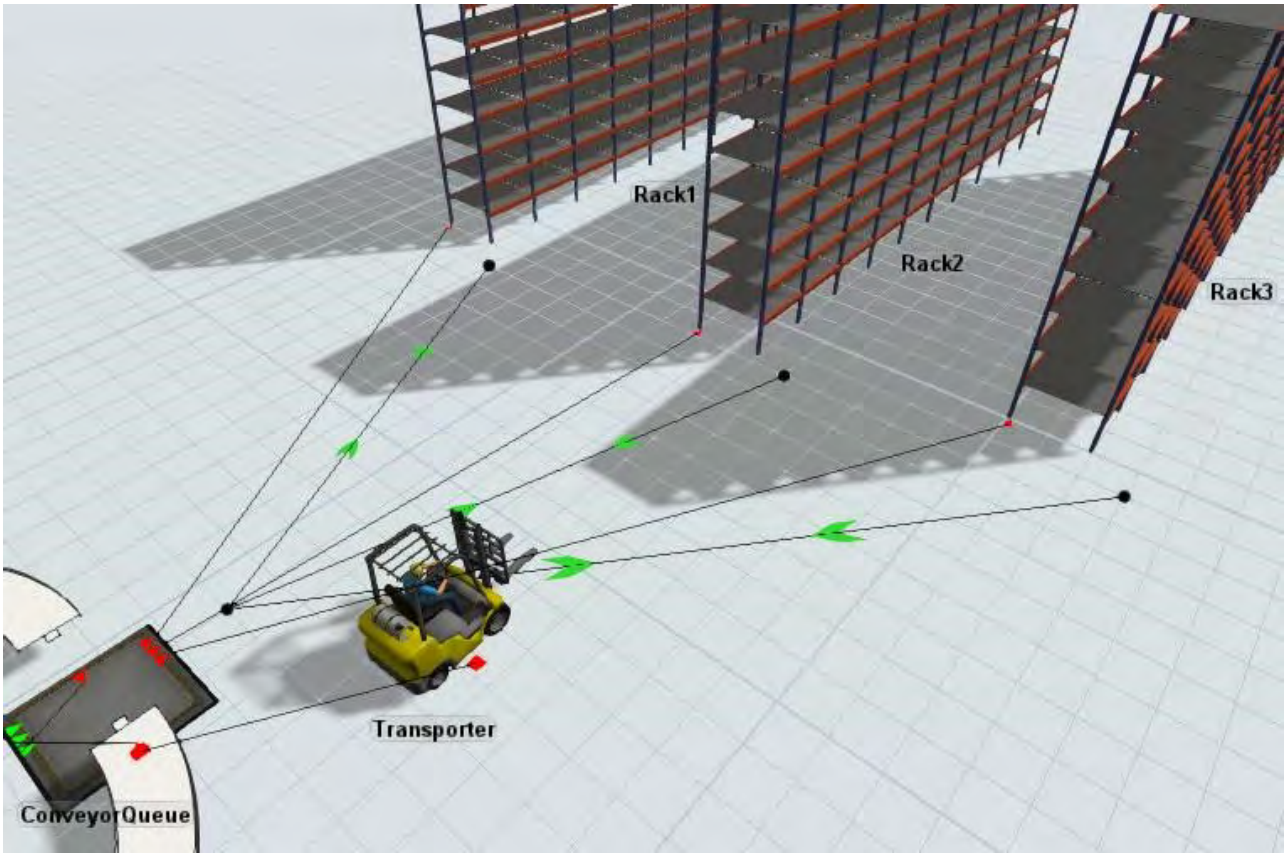
Step 8: Adding NetworkNodes to develop a path for the Fork Truck

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.

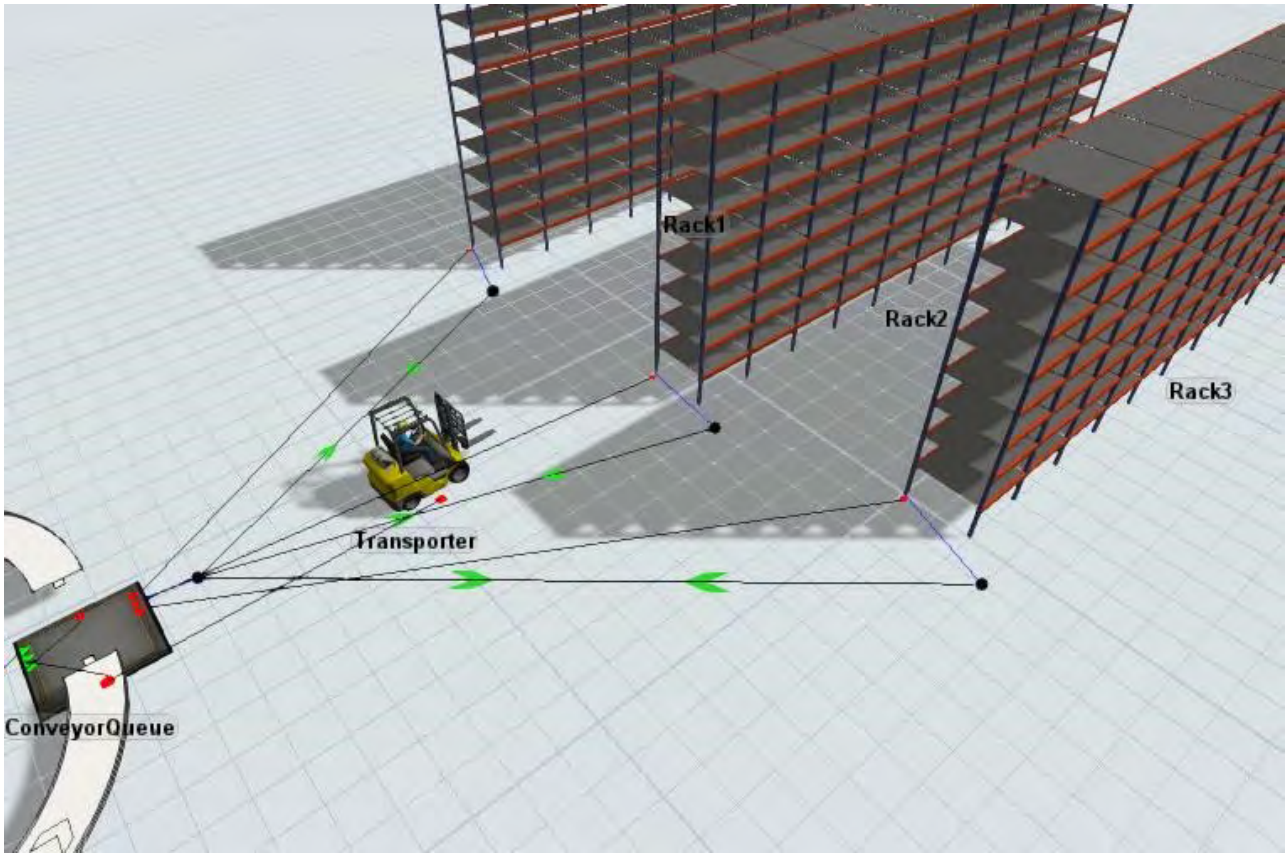
- Create NetworkNodes by dragging them from the library and into the model. Place them near the ConveyorQueue and each of the racks, and name them *NN1, NN2, NN3,* and *NN4.* The nodes will become the pick-up points and drop-off points in the model. You may add additional nodes between these nodes, but it is not necessary.



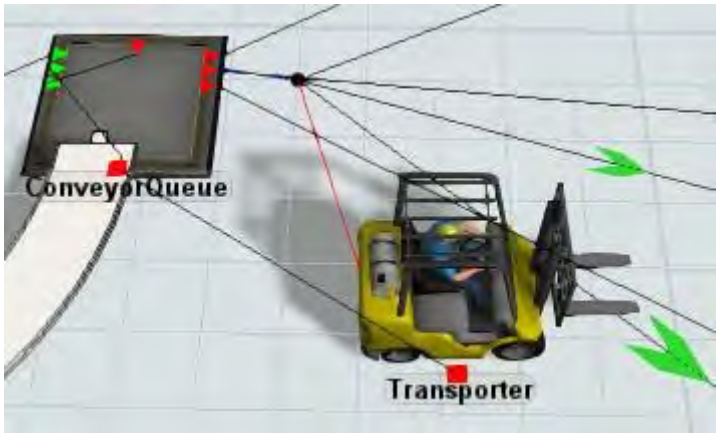
- Connect *NN1* to *NN2, NN3,* and *NN4* (A key). A line will appear after the connection is made with two green boxes along it, indicating that travel is possible in both directions between the two nodes.



- Connect each NetworkNode to the corresponding object (*NN1* to *ConveyorQueue*, *NN2* to *Rack1*, etc.) with the A key. A thin blue line will appear when the connection is made correctly. (If you cannot see the blue line, you may need to move the NetworkNodes)



- 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. Connect the *Transporter* to *NN1* (A key). This node now also becomes the starting point for the fork truck when you reset the model.

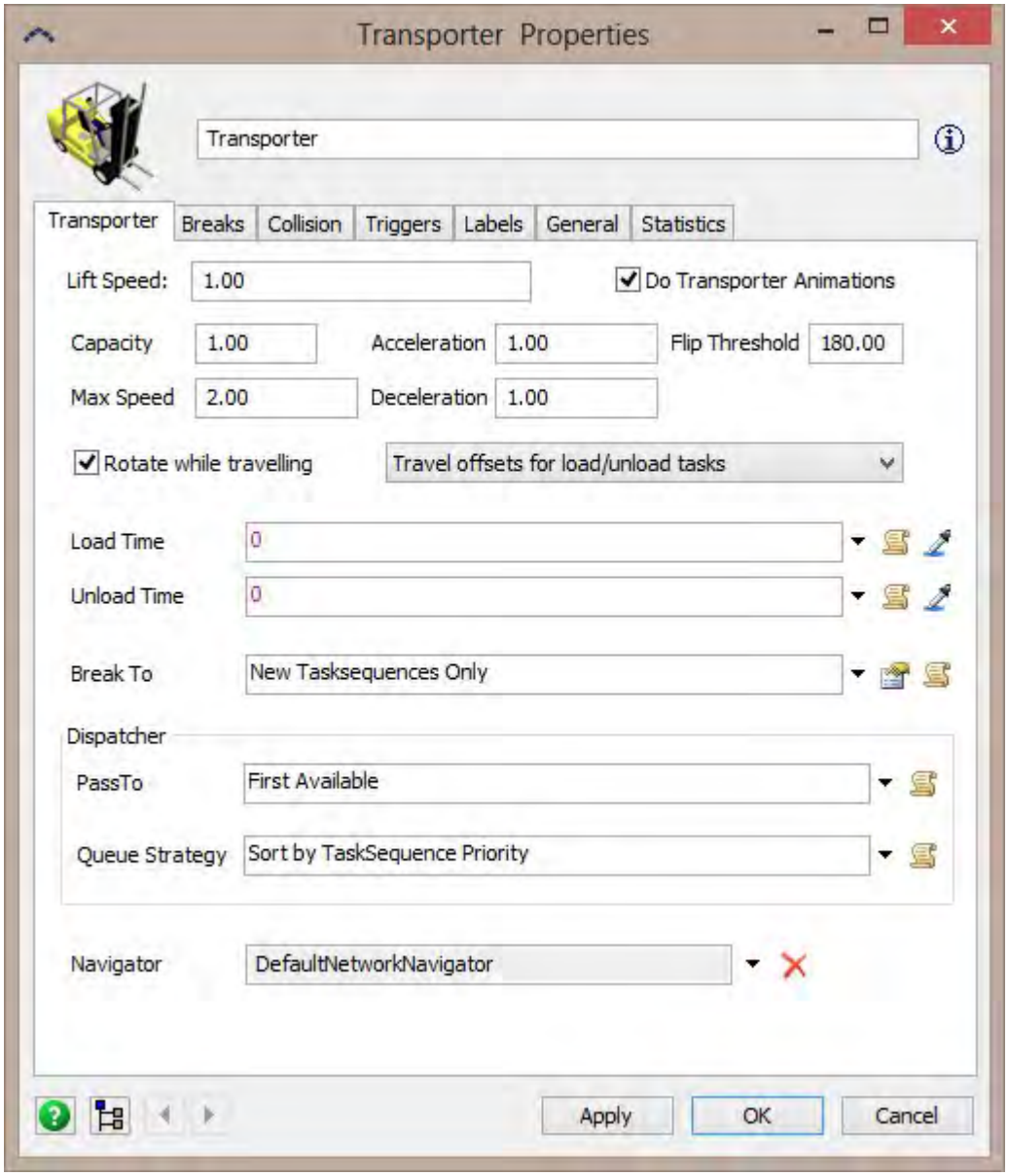


Step 9: Reset, save, and run the model

Now it would be a good idea to Reset, Save, and then Run the model to make sure the fork truck is using the network paths.

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 properties.

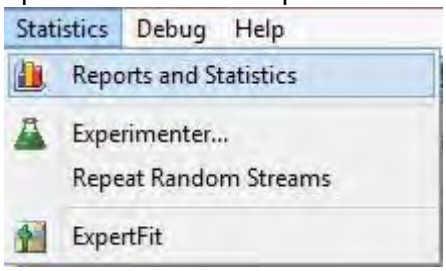


Offsets are used by the fork truck to locate where the flowitem needs to be picked up or dropped off in the object. This allows the fork truck to travel into the queue and pick up the box, and travel to the specific cell in the rack to drop off the box. To force the fork truck to stay at the NetworkNode and not to travel off the path network, select "Do not travel offsets for load/unload tasks" from the drop down picklist found below the field entitled Deceleration.

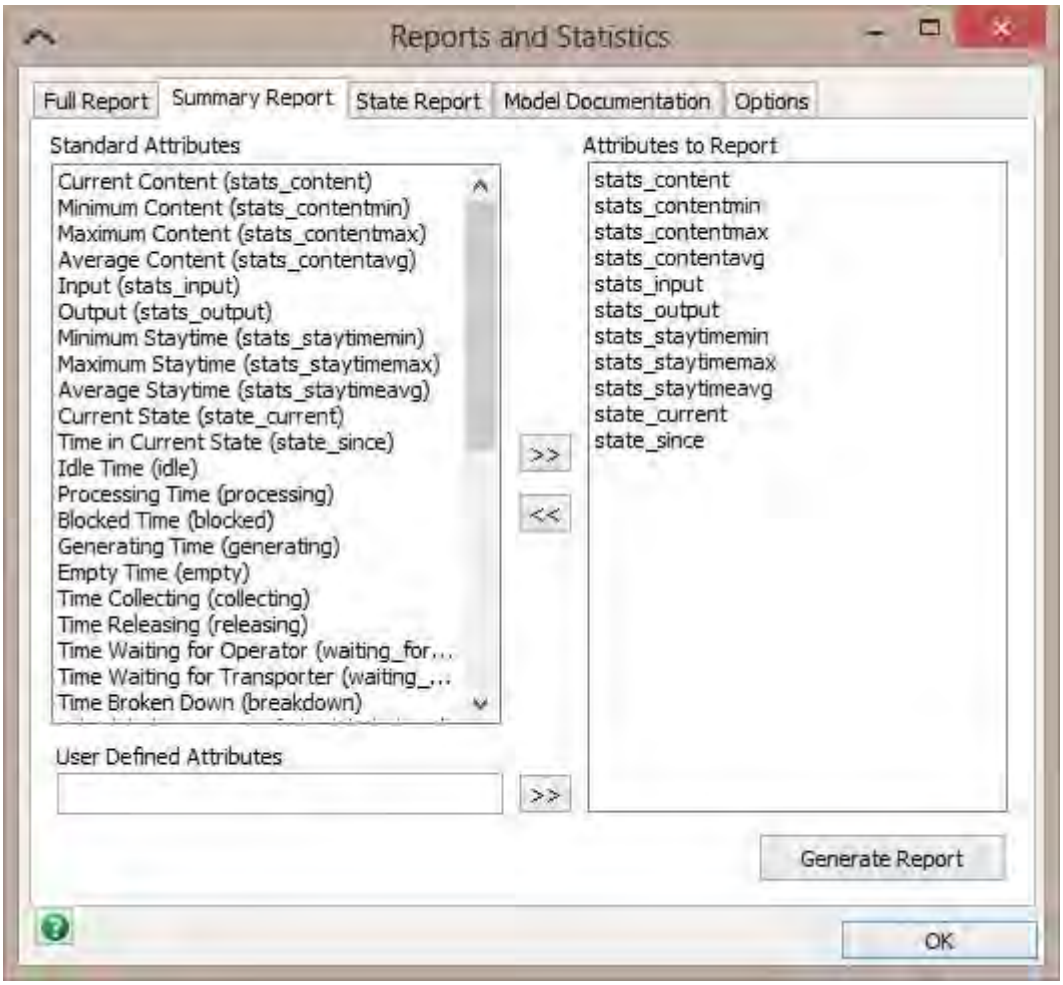
Path networks automatically use Dijkstra's algorithm to determine the shortest distance to travel between any two nodes in the network.

Step 10: Using reports to view output results

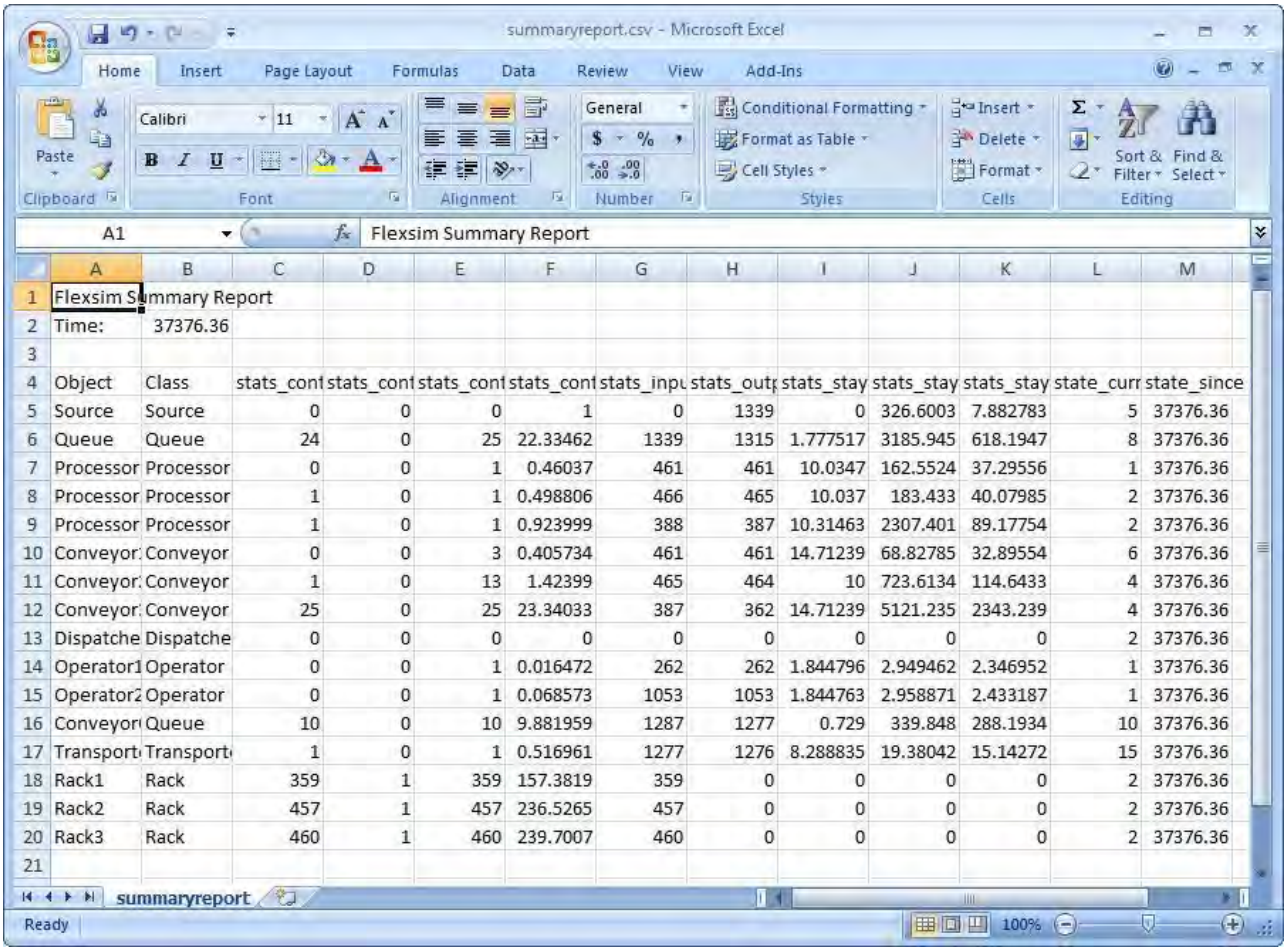
To view summary results of the simulation after having run the model for a period of time, select the main menu option Statistics > Reports and Statistics.



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

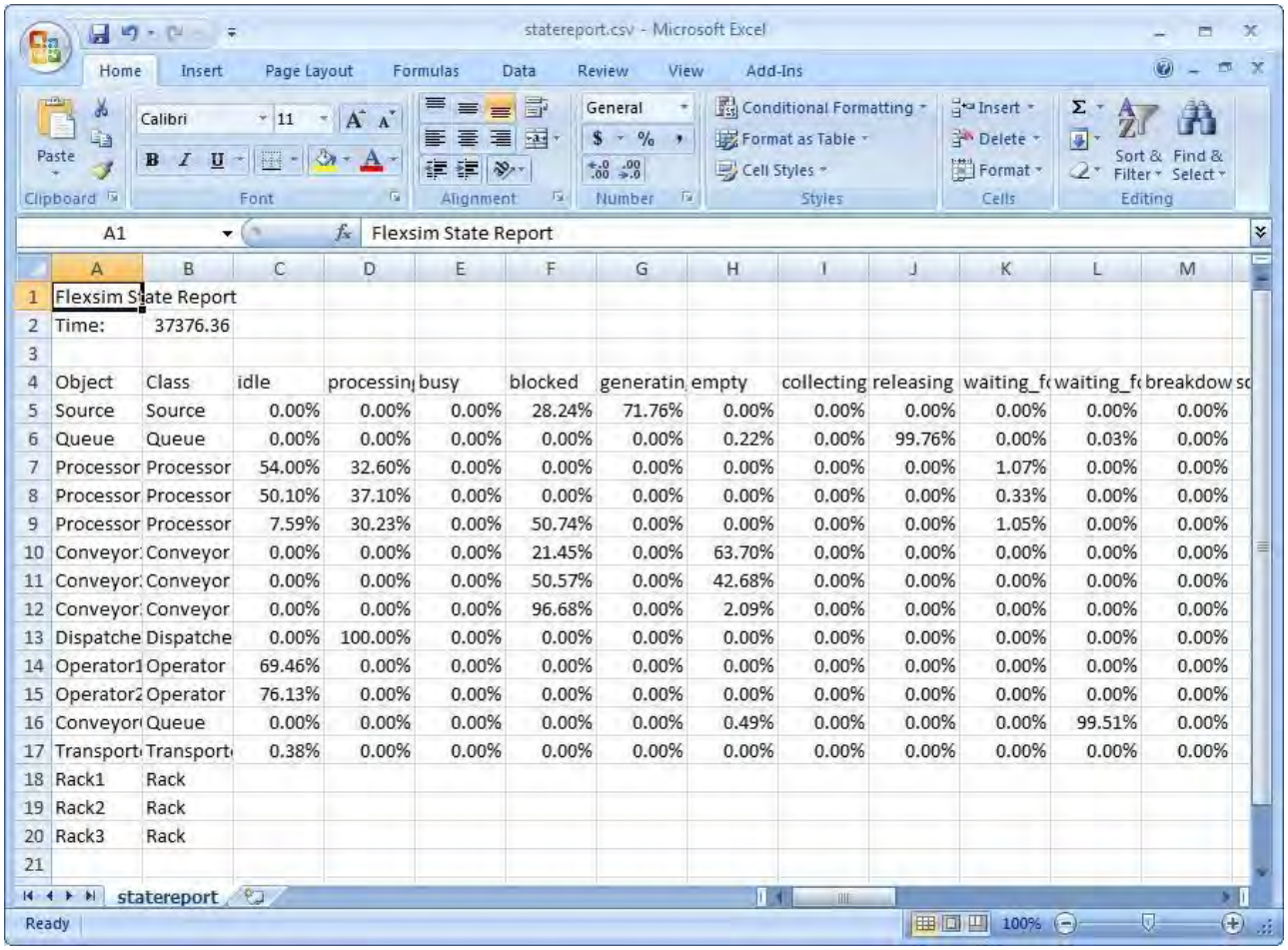


To generate a basic report, press the Generate Report button. If you have any other attributes you would like reported, you can add them using the interface provided. The report will be exported to a csv file and automatically displayed in Excel or whichever default program is set up to open csv files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



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Flexsim Summary Report												
2	Time:	37376.36											
3													
4	Object	Class	stats_cont	stats_cont	stats_cont	stats_cont	stats_inpu	stats_outp	stats_stay	stats_stay	stats_stay	state_curr	state_since
5	Source	Source	0	0	0	1	0	1339	0	326.6003	7.882783	5	37376.36
6	Queue	Queue	24	0	25	22.33462	1339	1315	1.777517	3185.945	618.1947	8	37376.36
7	Processor	Processor	0	0	1	0.46037	461	461	10.0347	162.5524	37.29556	1	37376.36
8	Processor	Processor	1	0	1	0.498806	466	465	10.037	183.433	40.07985	2	37376.36
9	Processor	Processor	1	0	1	0.923999	388	387	10.31463	2307.401	89.17754	2	37376.36
10	Conveyor:	Conveyor	0	0	3	0.405734	461	461	14.71239	68.82785	32.89554	6	37376.36
11	Conveyor:	Conveyor	1	0	13	1.42399	465	464	10	723.6134	114.6433	4	37376.36
12	Conveyor:	Conveyor	25	0	25	23.34033	387	362	14.71239	5121.235	2343.239	4	37376.36
13	Dispatche	Dispatche	0	0	0	0	0	0	0	0	0	2	37376.36
14	Operator1	Operator	0	0	1	0.016472	262	262	1.844796	2.949462	2.346952	1	37376.36
15	Operator2	Operator	0	0	1	0.068573	1053	1053	1.844763	2.958871	2.433187	1	37376.36
16	Conveyor(	Queue	10	0	10	9.881959	1287	1277	0.729	339.848	288.1934	10	37376.36
17	Transport	Transport	1	0	1	0.516961	1277	1276	8.288835	19.38042	15.14272	15	37376.36
18	Rack1	Rack	359	1	359	157.3819	359	0	0	0	0	2	37376.36
19	Rack2	Rack	457	1	457	236.5265	457	0	0	0	0	2	37376.36
20	Rack3	Rack	460	1	460	239.7007	460	0	0	0	0	2	37376.36
21													

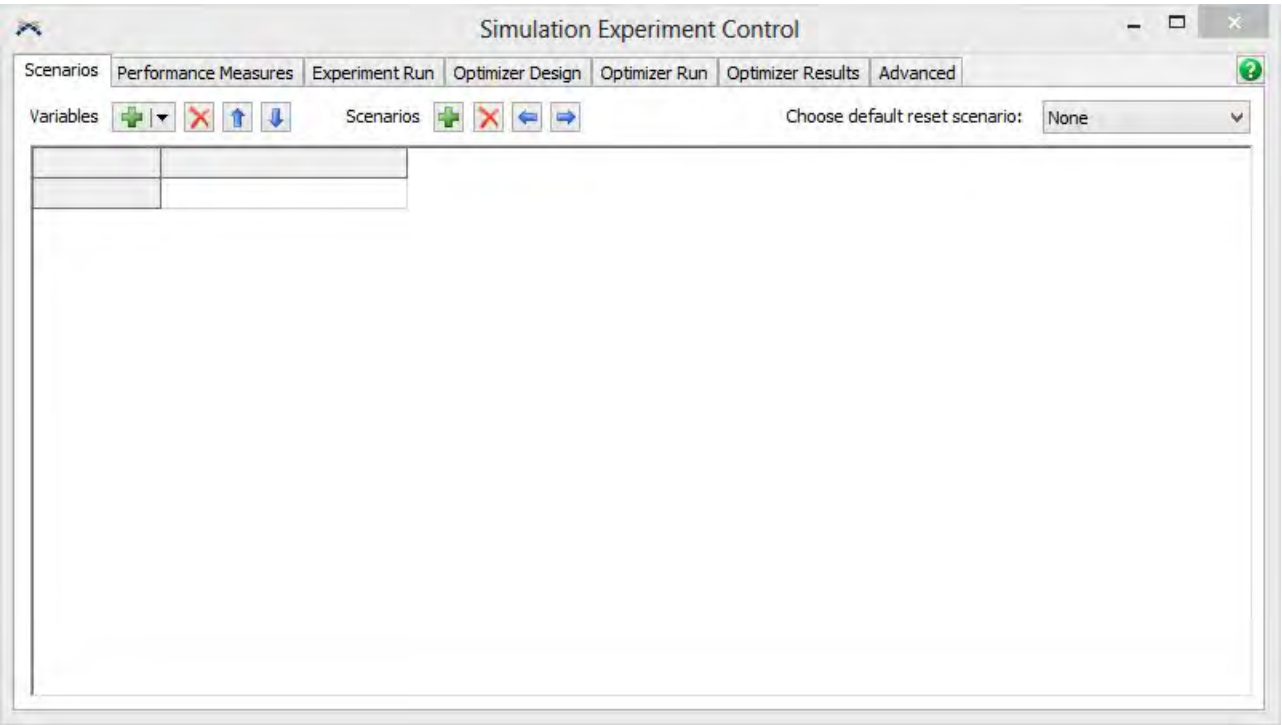
To create a state report, go to the State Report tab of the Reports and Statistics dialog window, and press Generate Report.



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Flexsim State Report												
2	Time:	37376.36											
3													
4	Object	Class	idle	processin	busy	blocked	generatin	empty	collecting	releasing	waiting_f	waiting_f	breakdow
5	Source	Source	0.00%	0.00%	0.00%	28.24%	71.76%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
6	Queue	Queue	0.00%	0.00%	0.00%	0.00%	0.00%	0.22%	0.00%	99.76%	0.00%	0.03%	0.00%
7	Processor	Processor	54.00%	32.60%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	1.07%	0.00%	0.00%
8	Processor	Processor	50.10%	37.10%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.33%	0.00%	0.00%
9	Processor	Processor	7.59%	30.23%	0.00%	50.74%	0.00%	0.00%	0.00%	0.00%	1.05%	0.00%	0.00%
10	Conveyor	Conveyor	0.00%	0.00%	0.00%	21.45%	0.00%	63.70%	0.00%	0.00%	0.00%	0.00%	0.00%
11	Conveyor	Conveyor	0.00%	0.00%	0.00%	50.57%	0.00%	42.68%	0.00%	0.00%	0.00%	0.00%	0.00%
12	Conveyor	Conveyor	0.00%	0.00%	0.00%	96.68%	0.00%	2.09%	0.00%	0.00%	0.00%	0.00%	0.00%
13	Dispatche	Dispatche	0.00%	100.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
14	Operator1	Operator	69.46%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
15	Operator2	Operator	76.13%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
16	Conveyor	Queue	0.00%	0.00%	0.00%	0.00%	0.00%	0.49%	0.00%	0.00%	0.00%	99.51%	0.00%
17	Transport	Transport	0.38%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
18	Rack1	Rack											
19	Rack2	Rack											
20	Rack3	Rack											
21													

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.



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!

Labels Tutorial Introduction

This tutorial will introduce you to using Labels in your model. Labels can be used to store information in your objects and items that can be accessed at anytime. As a couple of examples, labels can be useful to specify the downstream flow of your items, or they can be used to store financial data about an object.

What You Will Learn

- How to create and access labels
- How you can use labels to change the way your model functions

New Features

- You will be introduced to Pull Requirements

- You will be introduced to the FlowItem Bin

Approximate Time to Complete this Lesson

This lesson should take about 20- 30 minutes to complete.

Model Overview

In this model we will create a simple model that creates items with three different types. We will then track how many items we create of each type and then upon processing the item, we will modify the size of our items.

Click here for the Step-By-Step Tutorial.

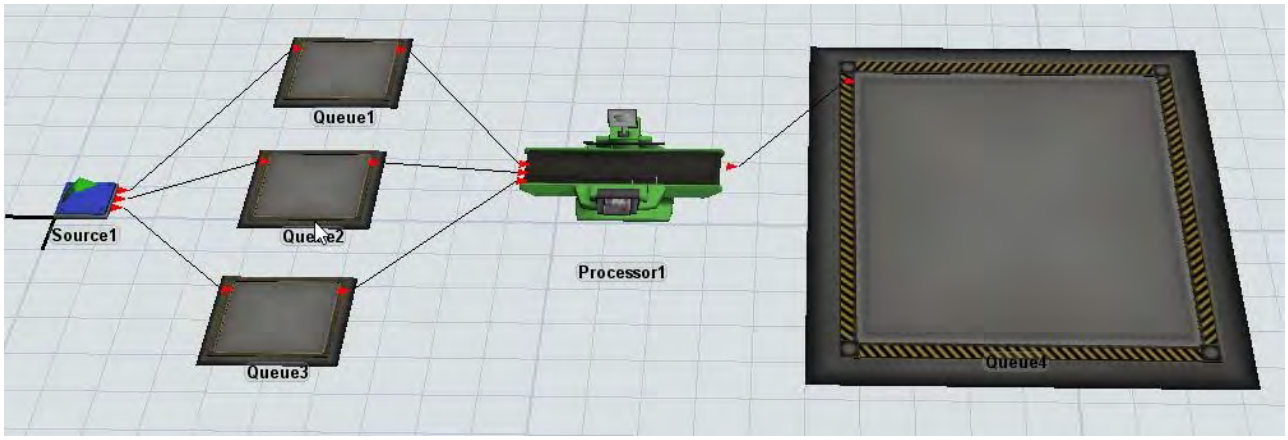
Labels Tutorial Step-By-Step Model Construction

Building Labels Model

 button on the toolbar. Click OK on the Model Units window – we will Begin a new model by clicking the use the default units for our model.

Step 1: Create the Objects

- Drag objects from your Library Icon Grid onto the 3D View to create the model shown below.



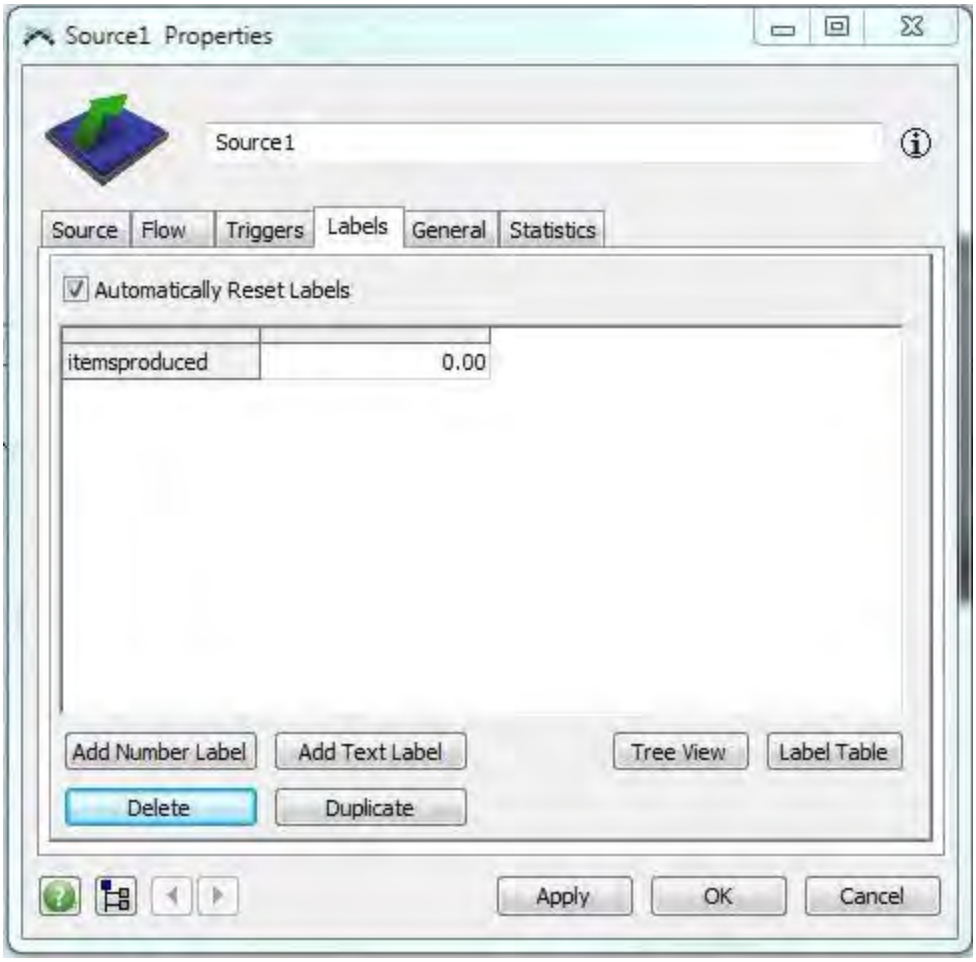
Connect all of the objects as shown:

- Connect *Source1* to *Queue1 Queue2* and *Queue3*
- Connect *Queue1*, *Queue2* and *Queue3* to *Processor1*
- Connect *Processor1* to *Queue4*

Step 2: Setup the Source Properties

All objects can store their own Labels. The Labels tab will allow you to create, view and manipulate the labels on that object.

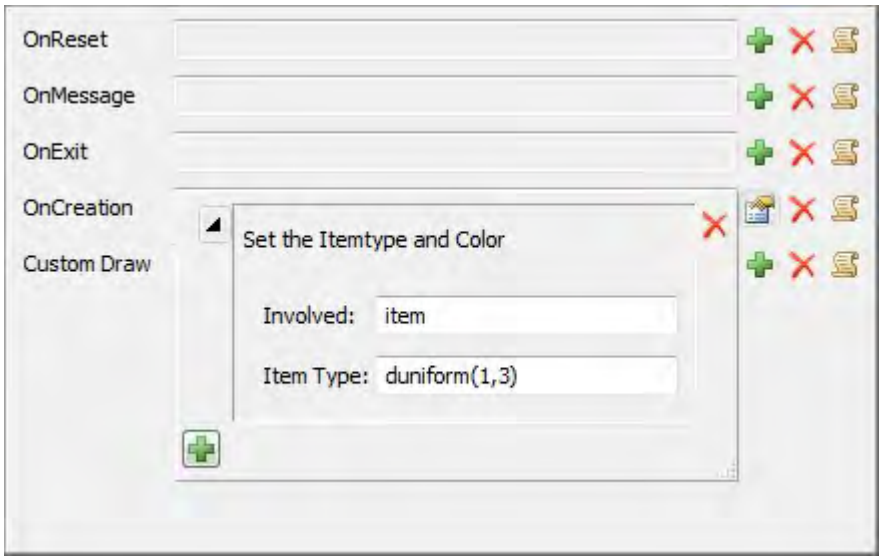
- Double-click on the *Source* to open its properties window.
- Go to the Labels tab.
- Check Automatically Reset Labels. This will cause the labels to reset to the value you give it when you reset the model.
- Click Add Number Label and double click *"newlabelname"* to rename the label *"itemsproduced"*.



The *Source* will create items with 3 item types. Each item type will be assigned a specific color.

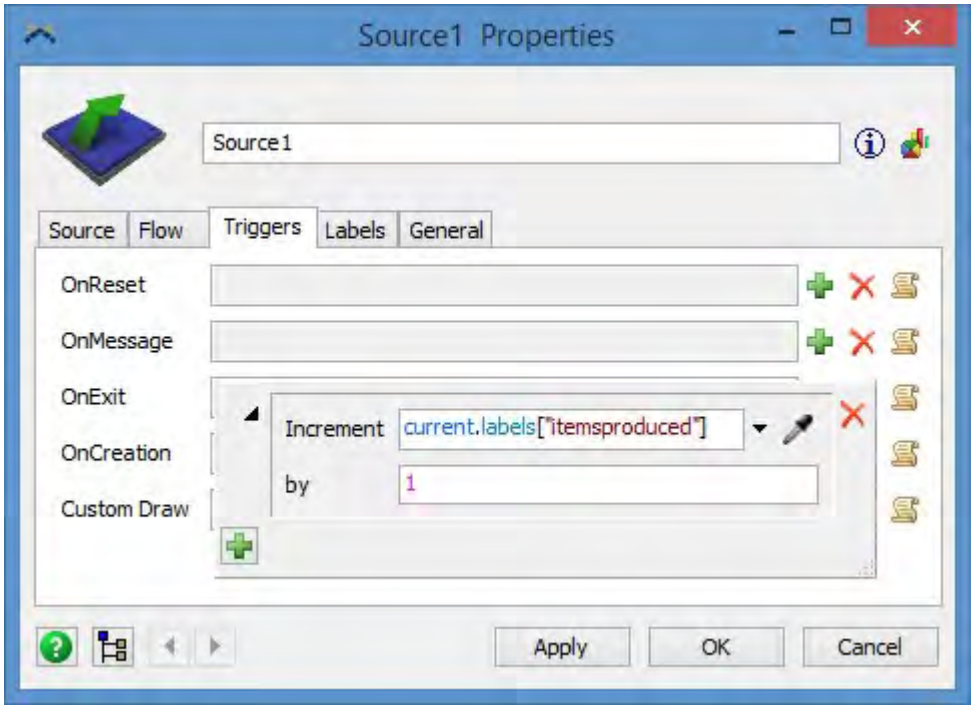
- Go to the Triggers tab.
- Click the add button for the OnCreation trigger, and select the *Set Item Type and Color* pick option.

- We will be using 3 item types, so keep the Involved and Type at their default values.



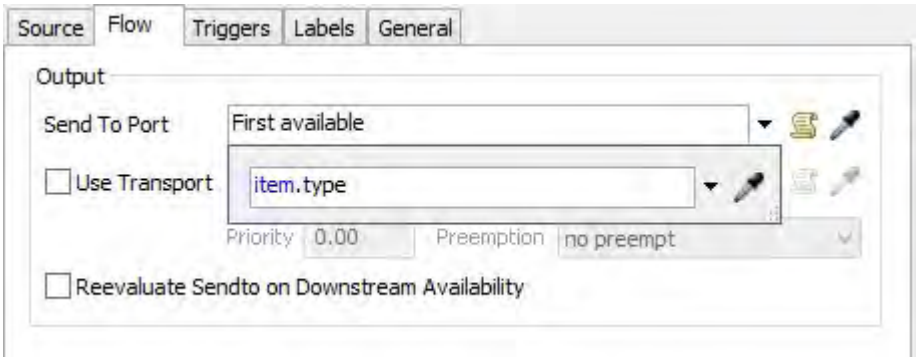
This will add a *type* label to each item with a random number between 1 and 3.
We also want to increment the *"itemsproduced"* label to keep track of how many items have been produced by the source.

- Go to the Triggers tab.
- Click the add button for the OnExit trigger, and select the Increment Value pick option.
- In the edit box for Increment replace *item.labels["labelname"]* with *current.labels["itemsproduced"]*.
- Leave the By at *1*. This will increment the value of the *"itemsproduced"* label by 1 on each time an item leaves the Source.



We will also assign items of each itemtype to go to a specific queue based on the itemtype.

- Go to the Flow tab.
- Choose By Expression in the Send to Port option. Leave it at the default *item.type*.



- Click Apply to apply the changes you made to the *Source*.

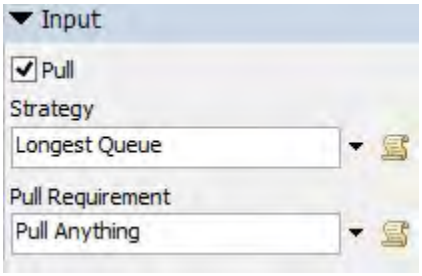
Reset and run your model. If you go back to the Labels tab of the source and watch your *"itemproduced"* label, it will update the value as each new flow item is created.

Close the properties window of the Source. It may ask you if you want to "Update Label Reset Values?". Click No. (You want the labels to reset to zero when you reset the model.)

Step 3: Setup the Processor Properties

We want the Processor to grab items from the Queue that has the most items waiting in, or the Longest Queue. Since each of the Queues do not know how many items are in the other Queues, it is works best to have the Processor determine which Queue it will pull an item from.

- Click on the *Processor* to open its properties in the Quick Properties window.
- Click the arrow next to the Input section to expand it.
- Check the Pull option and set the Pull Strategy to *Longest Queue*.

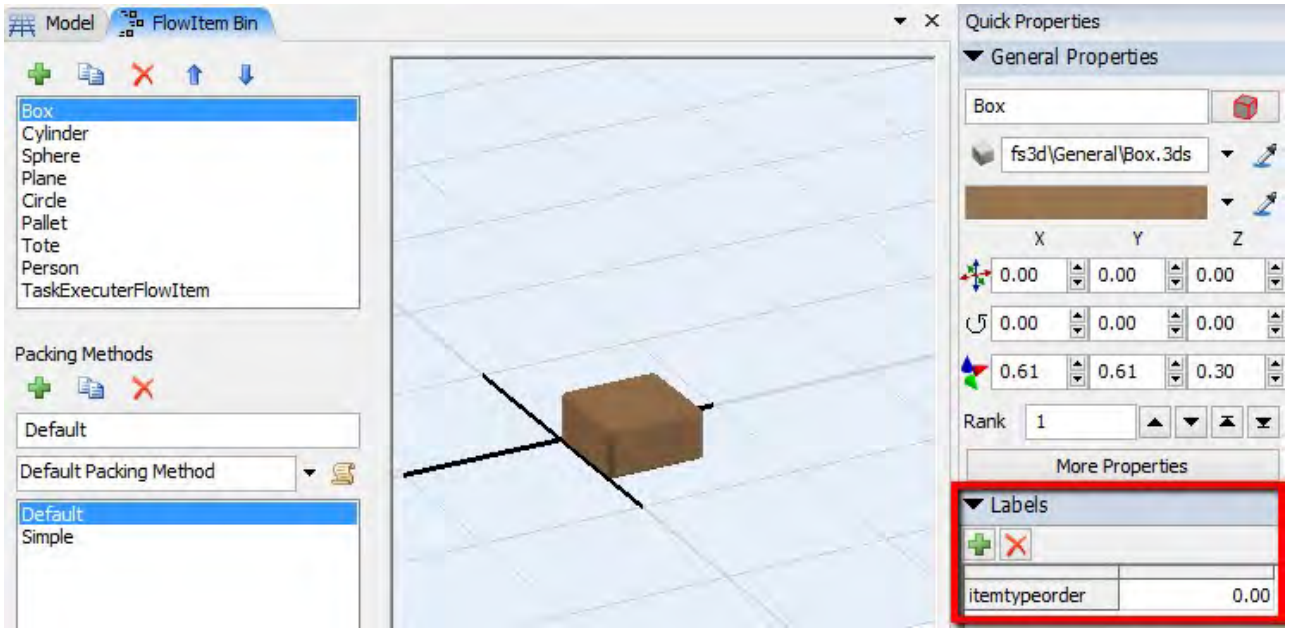


Step 4: Customizing Flowitems

We are going to store information on all the flowitems so that they can be accessed throughout the model.

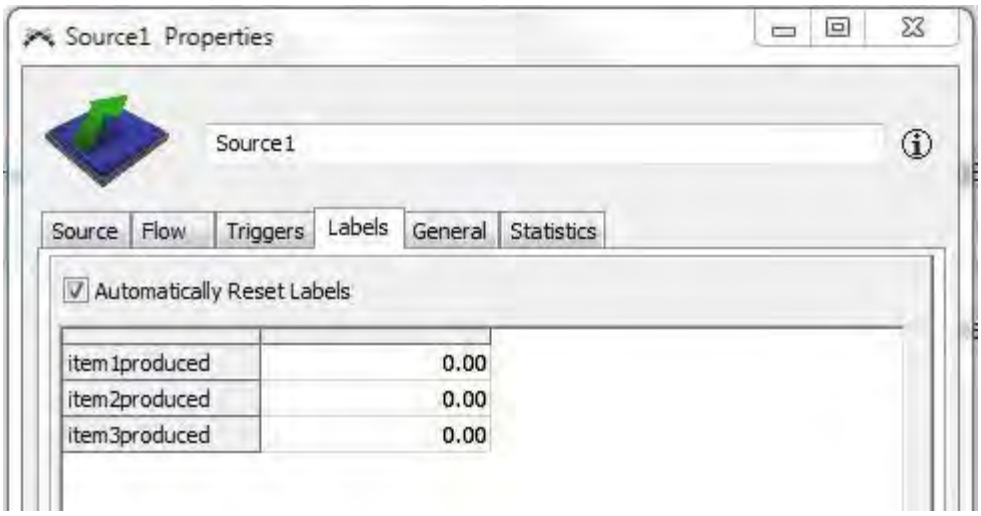
- Open the Flowitem Bin. (You can get to the FlowItem Bin either through the User Toolbar or through the Toolbox).
- With Box selected click the under the Labels section of the Quick Properties window and add a number label.

- Double click *"LabelName"* to rename the label *"itemtypeorder"*.



We will now have the source increment the *"itemtypeorder"* label on each flowitem for each specific type of item. This will allow us to know in what order the flowitem was created.

- Open the properties window for the *Source*.
- Go to the Labels tab.
- Change the label *"itemsproduced"* to *"item1produced"*.
- Select the *"item1produced"* label and hit the Duplicate button twice.
- Rename the two new labels to *"item2produced"* and *"item3produced"*.



In order to increment each these labels correctly, we need to customize the OnExit trigger. This will require editing the code manually, but don't worry – writing code is easy!

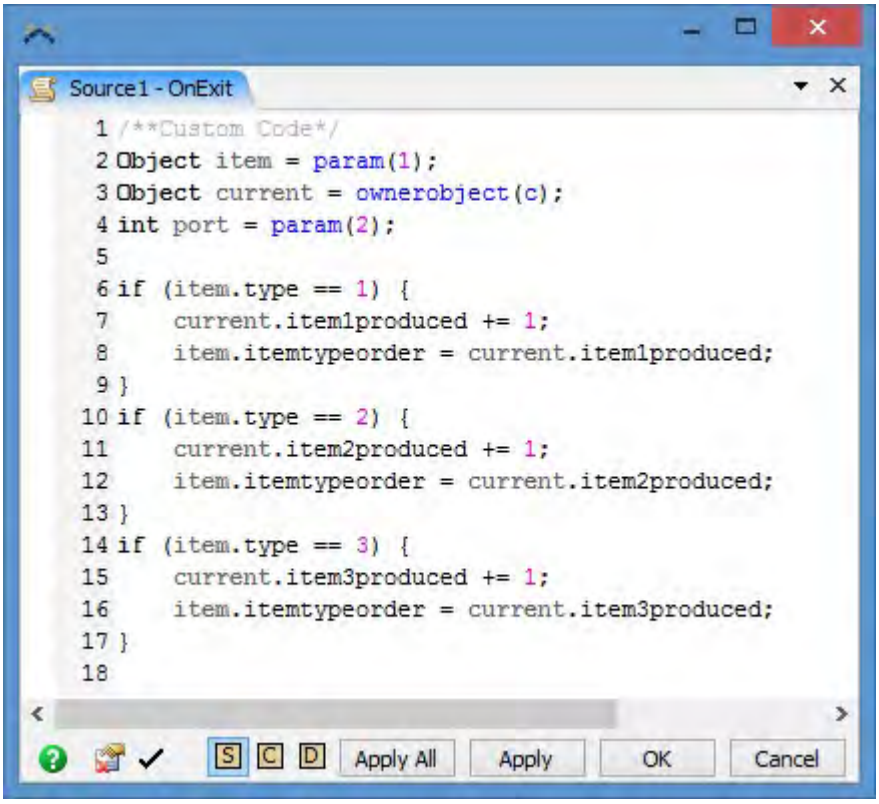
- Go into the Triggers tab, and click on the button, located to the left corner of the OnExit Trigger, to open the Code Editor.

- Click on the remove template button to get rid of the comments.
- Feel free to delete the "PickOption Start/End" lines.

- We want this *"increment"* function to increment each label individually. Write an "if" statement matching each item type to its own label. We will also set the new label's value to the flowitem.

```
if (item.type == 1) {  current.item1produced += 1;
    item.itemtypeorder = current.item1produced; }
```

- Write the same statements for types 2 and 3. The code should look like this when you are done:



- Click OK to apply and close the Code window.
- Click OK to apply and close the Properties window.

Step 5: Process the Flowitems

In this step, we will change the way the *Processor* processes the boxes.

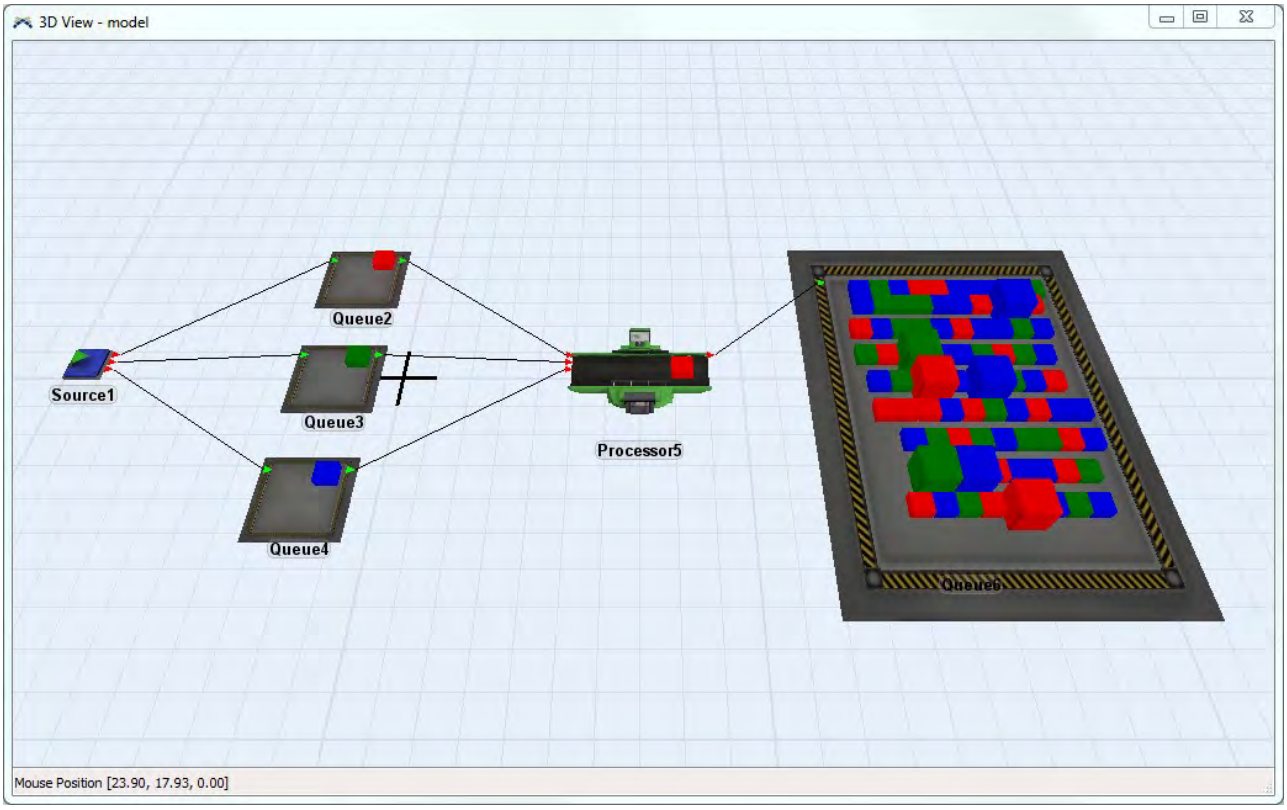
- Open the *Processor's* Properties window.
- Go to the *Triggers* tab.
- Click the code edit button next to the *OnProcessFinish* trigger.
- Write an if statement that will only allow every tenth item of each type to be set to a specific size:

```
if (item.itemtypeorder % 10 == 0) {    item.setSize(1, 1, 1);
}
```

Note:You don't need to do this two more times, because it applies to any flowitem with the *"itemtypeorder"* label.

- Click OK on the Edit Code window as well as the Properties window.

Reset and Run your model. Your model should begin to look something like this:



This completes the the Labels tutorial. Congratulations!

Global Modeling Tools Tutorial Introduction

This lesson will introduce the basic concepts of global modeling tools that can be used throughout your model. These properties can simplify views and enable dynamic changes in your model. We will also look at ways to use a Combiner and Separator in order to better model situations that may happen in real life. Note: For more information on global modeling tools, see the Modeling Tools section.

What You Will Learn

- How to create and access Global Tables
- How to create and access Global Variables and Global Macros
- How to write a simple User Command for use in your model • How to use a Combiner and Separator object

New Objects

In this lesson you will be introduced to the Combiner and Separator objects.

Conveyor Module Required

This lesson requires the conveyor module. If you do not have this module, you can download and install if from the main menu under Help > Online Content.

Approximate Time to Complete this Lesson

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

Model Overview

In this model we're going to look at some different ways of accomplishing tasks as opposed to how they have been done in previous lessons. Both methods give the same result, but some users prefer one way over the other.

We'll also take a look at Combiners and Separators and how they can be used in your models. Click here for the Step-By-Step Tutorial.

Global Modeling Tools Tutorial Step-By-Step Model Construction

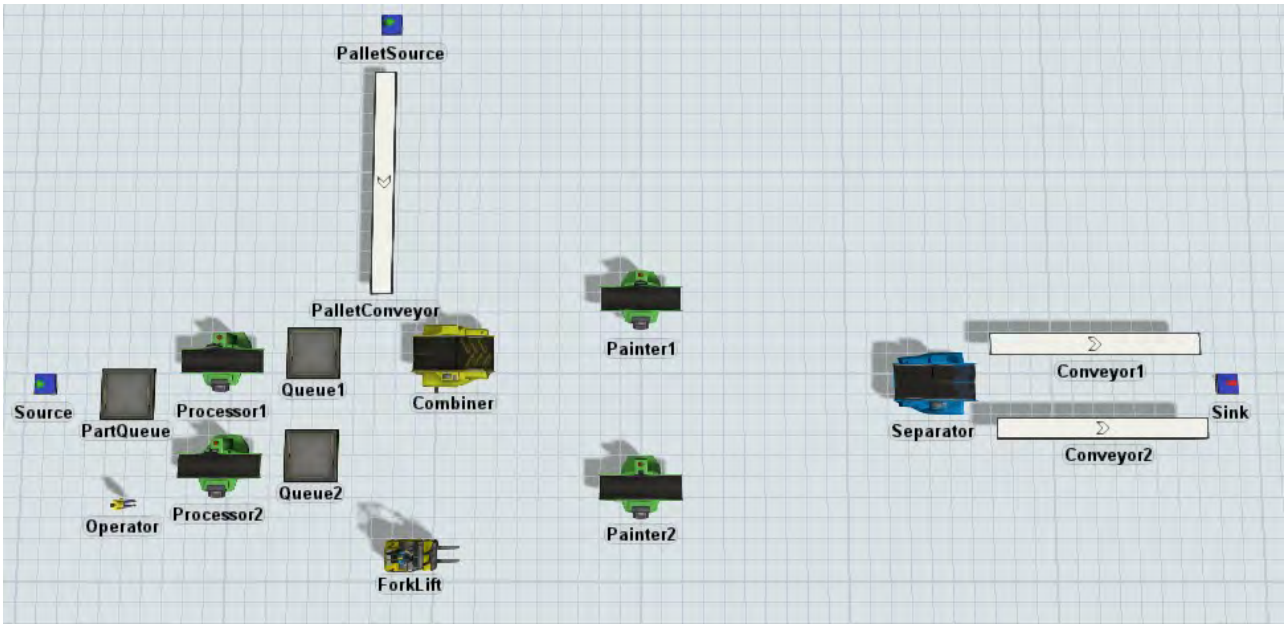
Building Global Modeling Tools Model

 button on the toolbar. Click OK on the Model Units window; we will Begin a new model by clicking the use the default units for our model.

If at any time you encounter difficulties while building this model, a fully functional tutorial model can be found at http://www.flexsim.com/tutorials

Step 1: Create the Objects

- Drag objects from your Library Icon Grid onto the 3D View to create the model shown below.
- Rename the objects as shown as we will refer to these objects by name throughout the tutorial.



Step 2: Connect the ports

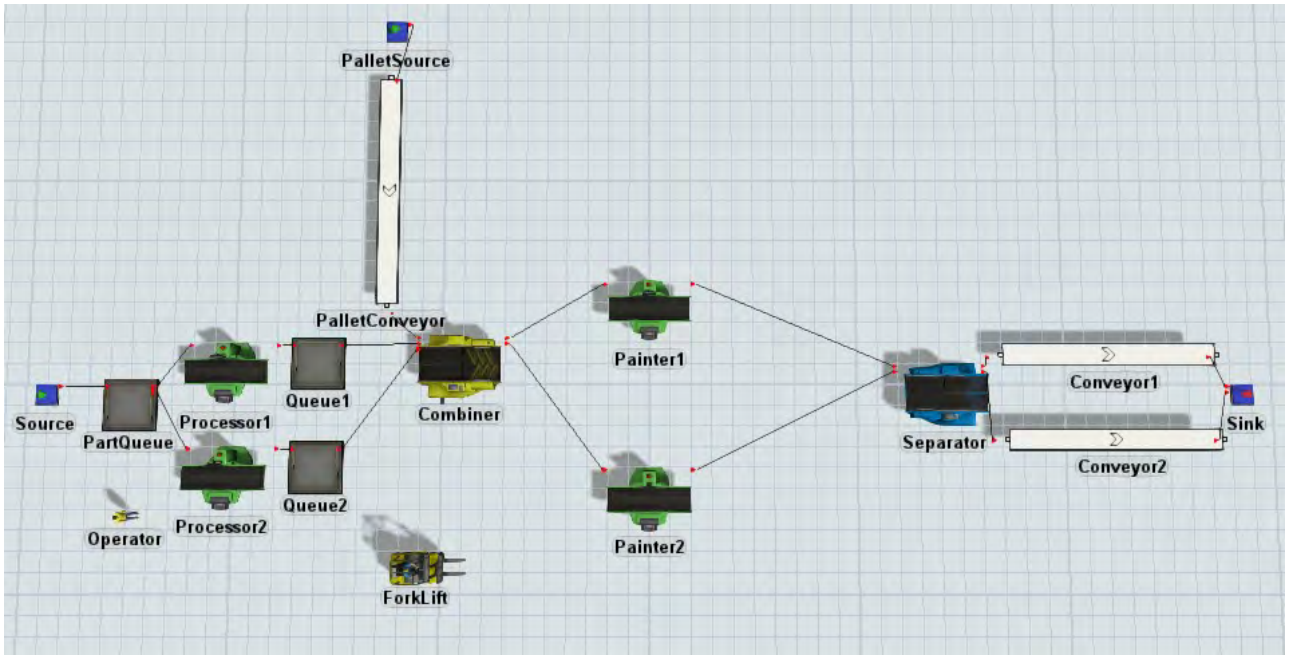
Connect all of the objects as shown below.

- Connect *Source* to *PartQueue.*
- Connect *PartQueue* to *Processor1* and *Processor2*
- Connect *Processor1* to *Queue1* and *Processor2* to *Queue2*
- Connect *PalletSource* to *PalletConveyor*
- Connect *PalletConveyor* to *Combiner*

Note: It is required to have *PalletConveyor* attached to the 1st input port of *Combiner*.

- Connect *Queue1* and *Queue2* to *Combiner*
- Connect *Combiner* to *Painter1* and *Painter2*
- Connect *Painter1* and *Painter2* to *Separator*
- Connect *Separator* to *Conveyor1* and *Conveyor2*
- Connect *Conveyor1* and *Conveyor2* to *Sink*

Notice that we did not make any center connections to *Operator* or *Forklift*. We will tie those in using Global Variables.



Step 3: Set the Source Properties

Set up the source to create two item types.

- Leave the default Inter-Arrival Time of *Source* at *exponential(0, 10, 0)*.
- Set the OnCreation trigger to Set Item Type and Color with a Type of *duniform(1,2)*.
- Click OK to apply and close the Properties window.

Step 4: Set the PalletSource Properties

Our pallets are going to act as dummy objects that will hold our other parts as they go through the *Combiner* and *Painter*. Setting the FlowItem Class to a pallet will cause the *Combiner* to neatly stack our other flowitems on top of the pallet item.

Our pallet is also going to be used to determine what items are combined in the *Combiner*. This is specified by setting the item type of each pallet. This will be explained later on.

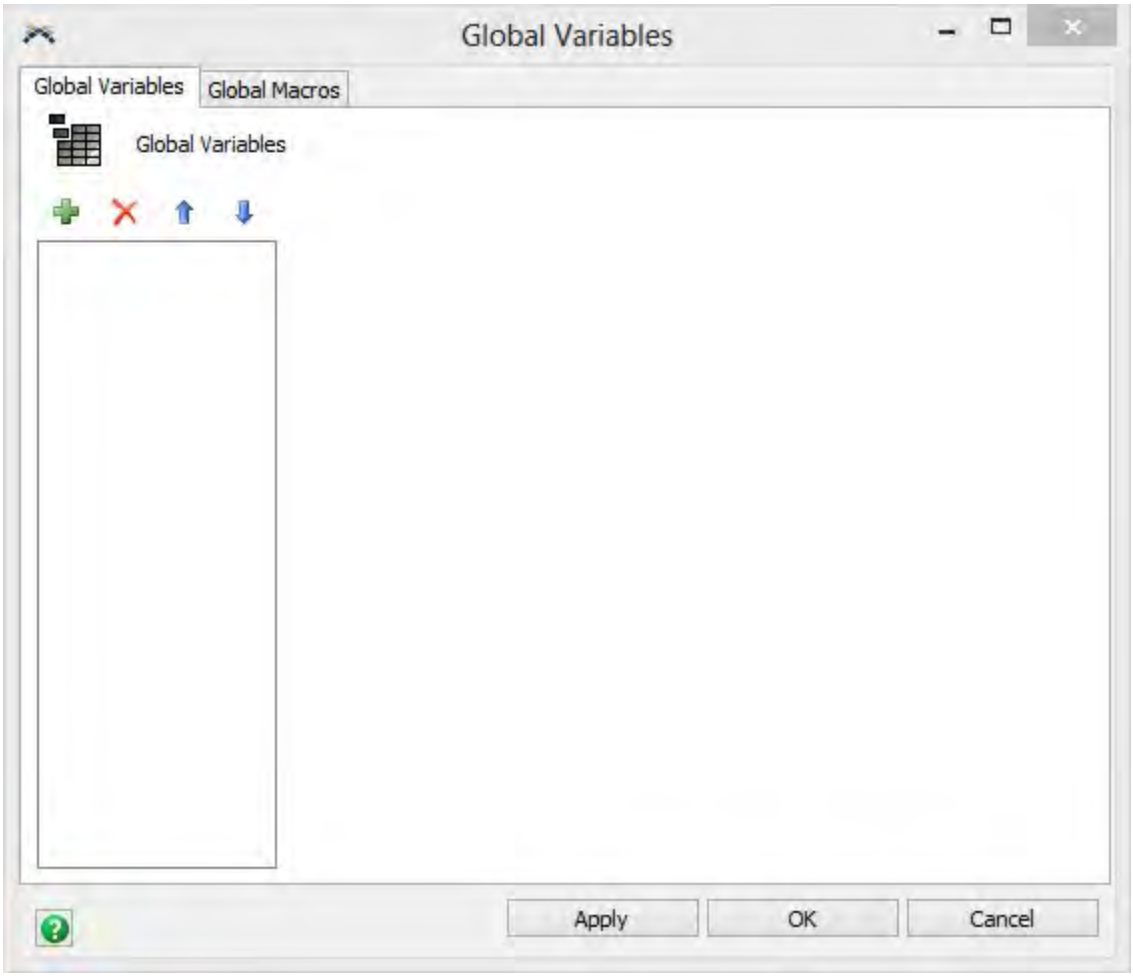
- Set the FlowItem Class of *PalletSource* to Pallet (this can be done through the Source's properties under the Source tab or through the Quick Properties).
- To have an infinite supply of pallets, set the Inter-Arrival Time to 0.
- Go to the Triggers tab and add a trigger to the OnCreation of *Set Type, Name or Label*.
- Change the Type to *duniform(1,6)*.
- Click OK to apply and close the Properties window.

At this point, feel free to try your model out. You should see your flowitems run through the entire model, combining a pallet with one item from *Queue1* and one item from *Queue2* at the *Combiner*, and then separating those items at the *Separator*.

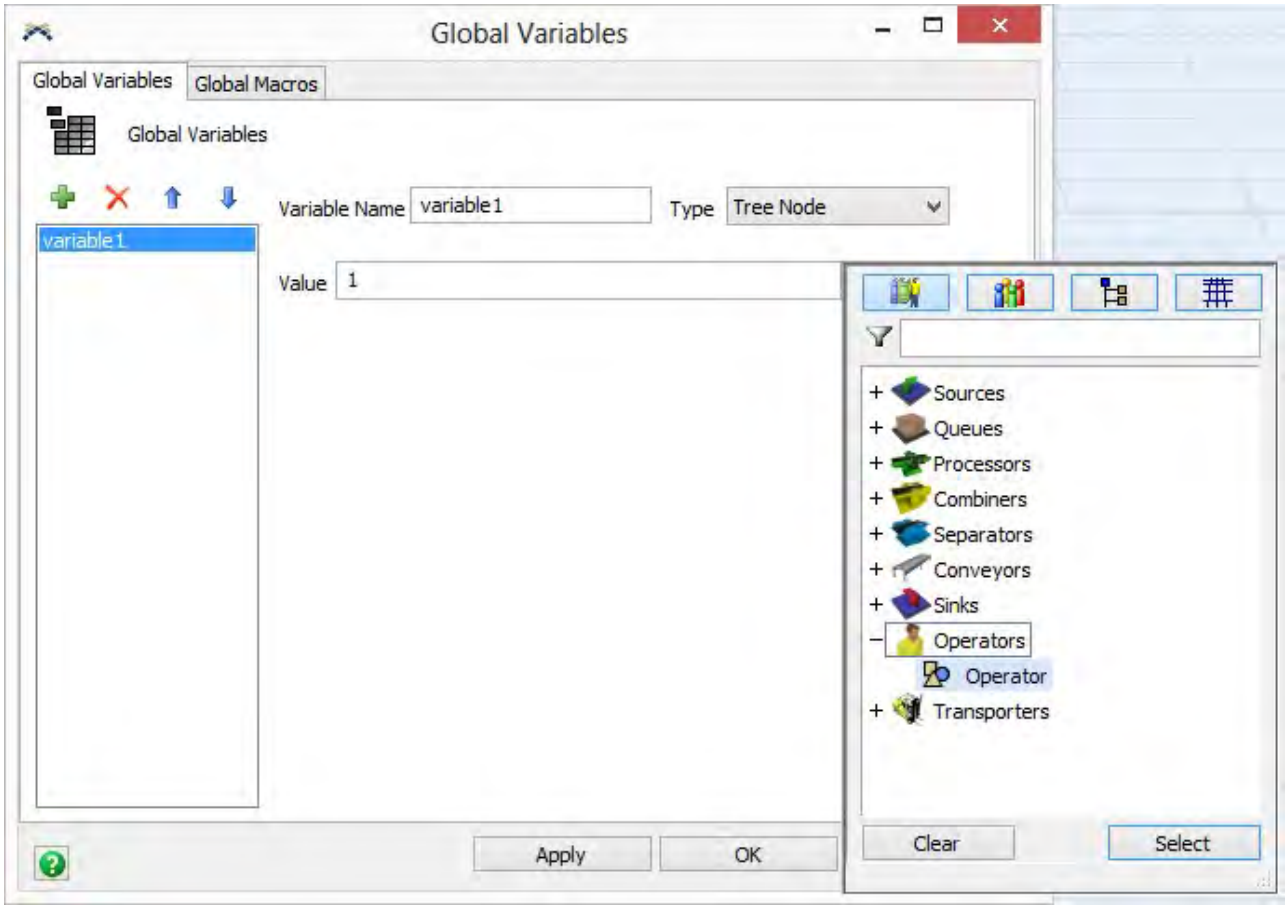
Step 5: Add the Operators

To show another way of connecting objects together, we are not going to connect our operators using center ports. Instead, we're going to create Global Variables that point to our operators. We will also set up a Global Macro to define the processing time of our two processors. See the Global Variables Window reference to learn more about Global Variables.

- Add a new Global Variable from the Toolbox (View > Toolbox or click on the Tools button in the main toolbar).

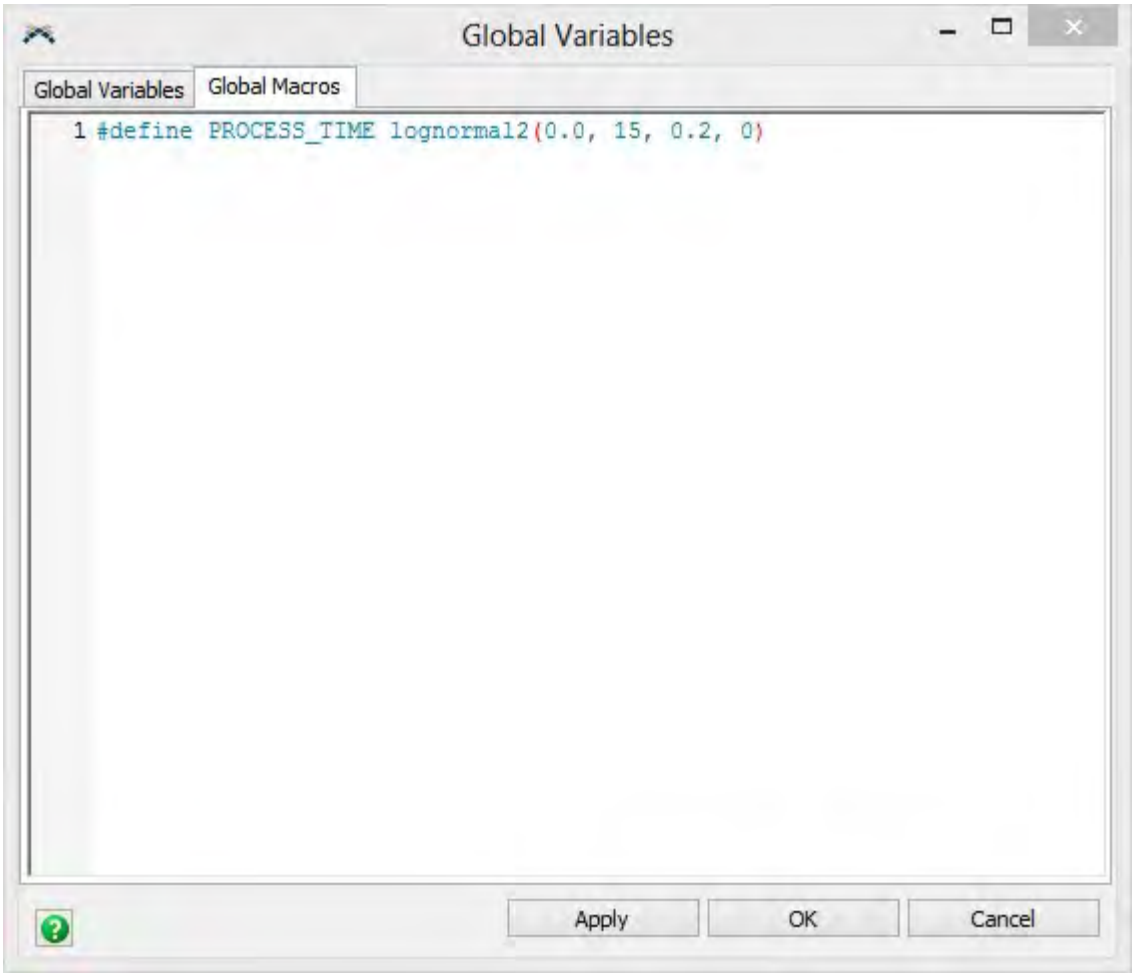


- Click and set the Variable Name to *Operator*.
- Select *Tree Node* for the Type.
- Click next to the Value field and select the *Operator* as the value then click Select.



- Repeat the previous steps to create another variable for *Forklift*.
- Next, click on the Global Macros tab and enter the follow code:

```
#define PROCESS_TIME lognormal2(0.0, 15, 0.2, 0)
```

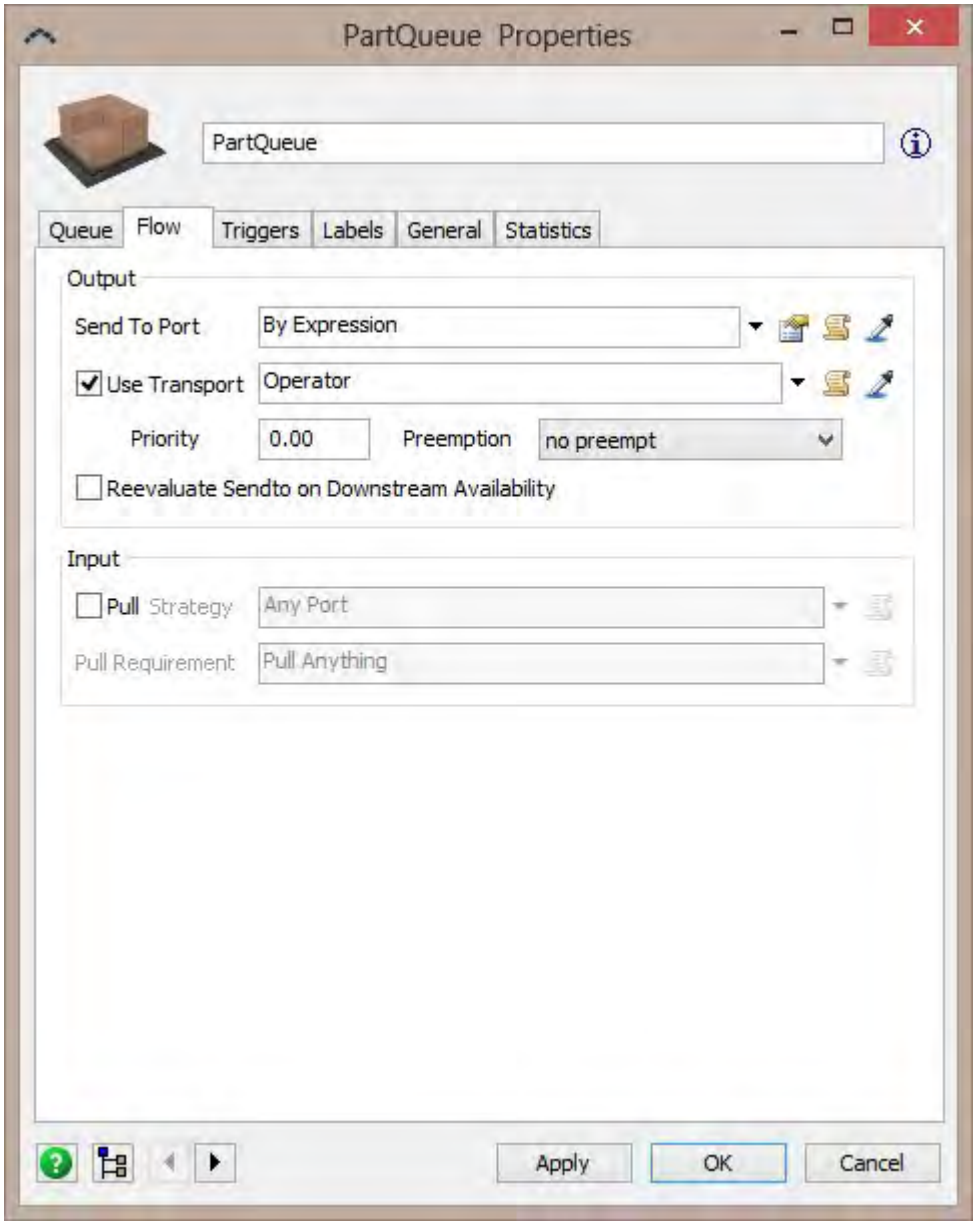


- Click OK to apply and close the Global Variables window.

Step 6: Setup the Queues

Processor1 is going to handle all items of type 1 and *Processor2* will handle all items of type 2.

- Set the Maximum Content of *PartQueue* to 25.
- Under the Flow tab, select By Expression in the pick list for Send To Port. This will default to *item.type*.
- Check the Use Transport button and replace *centerobject(current, 1)* with *Operator*.

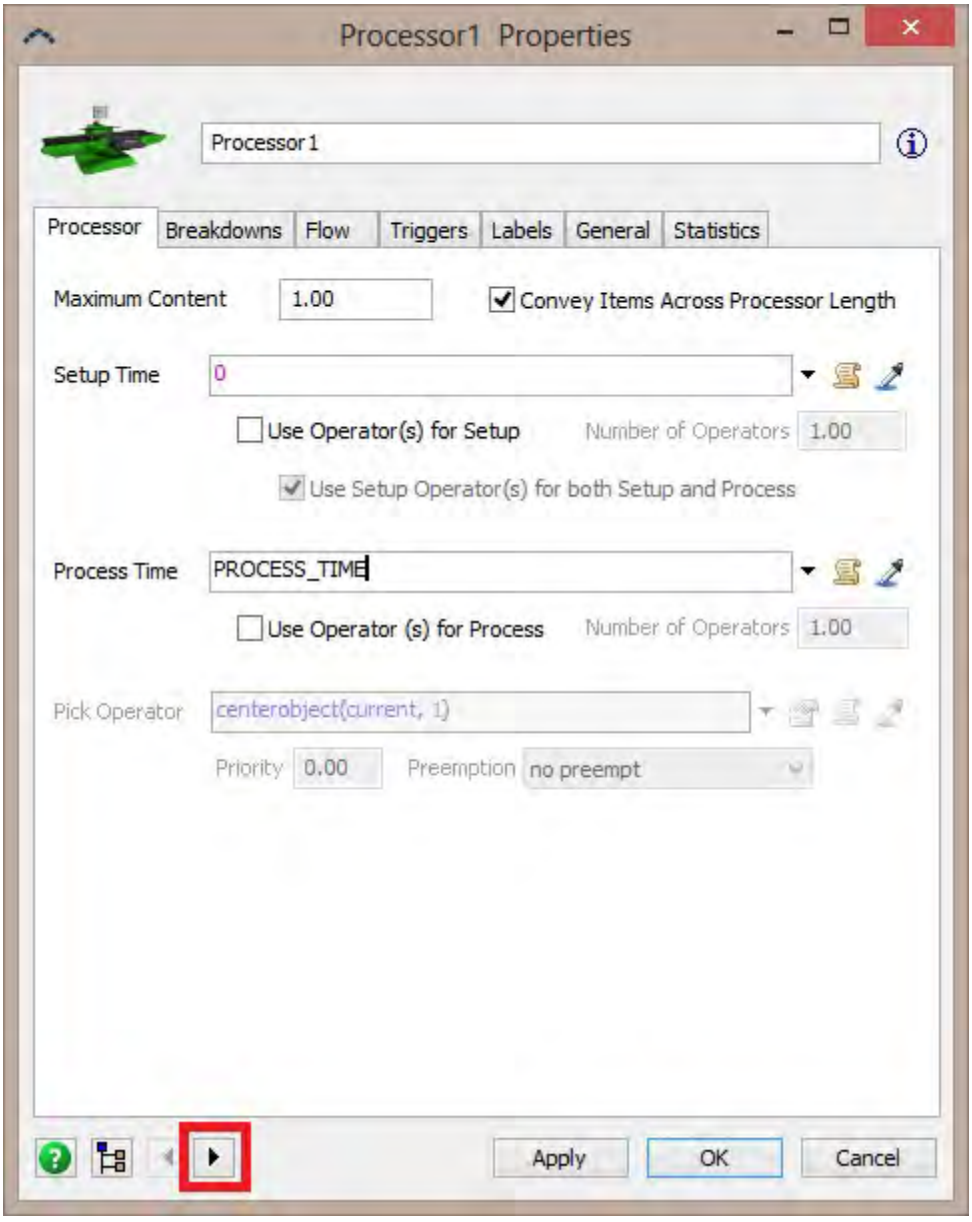


- Click OK to apply and close the Properties window.
- Set the Maximum Content of *Queue1* and *Queue2* to 100.

Step 7: Setup the Processors

We just defined a Global Macro that can be used for the process time of our processors.

- Open *Processor1* and type PROCESS_TIME into the Process Time box.



- Click the Right Arrow button at the bottom of the properties window to move to the next processor.
- Set the Process Time of *Processor2* to PROCESS_TIME.
- Click OK to apply and close the Properties window.

Step 8: Setup a Global Table

We will now set up a Global Table for use with our combiner.

The Combiner is used to group multiple flowitems together. The process by which this works is as follows:

- The Combiner will first accept a single flowitem through input port 1. This becomes the container flowitem into which all the other flowitems will be combined with.

- Once the first flowitem is accepted, the combiner collects a batch of flowitems from the remaining input parts based on its component list.

Components List

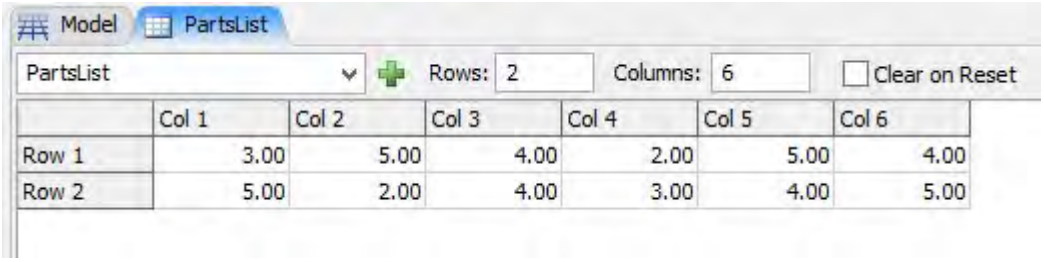
	Target Quantity
From Input Port 2	2.00
From Input Port 3	3.00

- Once the batch is complete, the Combiner goes through its setup and process time before sending the combined flowitems on to the next step of the model.

For this model we're going to have different component lists to represent different orders or assemblies that are being created. Since we connected the pallet to input port 1 it can be used to determine which component list is to be used.

We can create a Global Table that will store all of the different orders (component lists) for use with the Combiner.

- Add a new Global Table from the Toolbox (View > Toolbox or click on the Tools button in the main toolbar).
- Rename your table to *PartsList*.
- Set the number of rows to 2 and the number of columns to 6.
- Enter the following numbers into your table.



	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6
Row 1	3.00	5.00	4.00	2.00	5.00	4.00
Row 2	5.00	2.00	4.00	3.00	4.00	5.00

- Close the Global Table window.

Step 9: Setup the Combiner

- Open the *Combiner* properties window and click on the Combiner tab.
- Select Convey Items Across Combiner Length.
- Click on the Flow tab and select Use Transport.
- Replace *centerobject(current,1)* with *Forklift*.

We want the Combiner's Component List to update each time a new pallet comes into port 1.

- Click on the Triggers tab and add a new operation to the OnEntry trigger.
- Select the *Update Combiner Component List* from the pick list.
- Replace *"tablename"* with *"PartsList"*.
- Click OK to apply and close the Properties window.

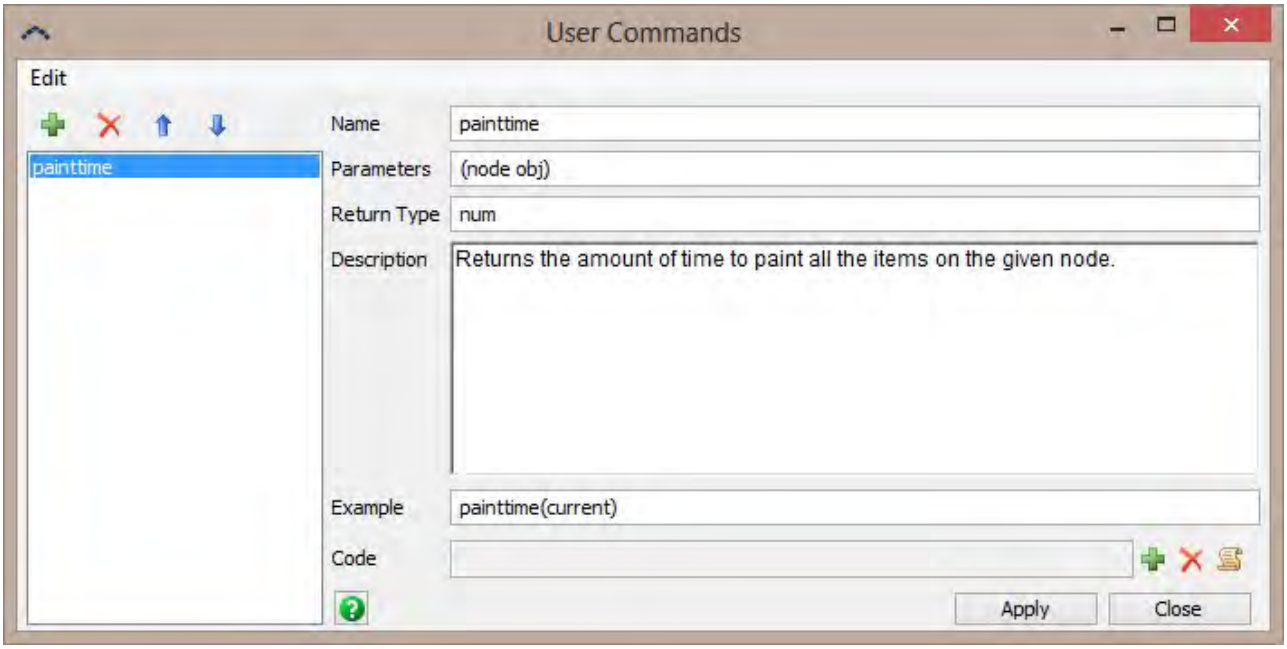
The *Update Combiner Component List* pulls data from a Global Table based on the type of the incoming flowitem on port 1. So, if the pallet is of type 4, it will pull the two row values from column 4 of our Global Table *PartsList*.

Step 10: Write Custom User Commands

FlexSim allows you to write your own custom User Commands, or functions, that can be used throughout your model.

The first command we will write will return the time it takes to paint all of the items on our pallet. This code could be written directly into the process time of the *Painter1* and *Painter2* processors, but as we have multiple painters all utilizing the same processing time, writing a custom command makes it easier to make changes to that processing time.

- Add a new User Command from the Toolbox (View > Toolbox or click on the Tools button in the main toolbar).
- Click Add to create a new command.
- Set the Name of the command to *painttime*.
- Set the Parameters to *(node obj)*.
- Set the Return Type to *num*.
- The command's Description is: *Returns the amount of time to paint all the items on the given node.*
- Set the Example to *painttime(current)*.



For the code, we want our command to pass the *Painter* processor, cycle through all of the items on the pallet, and return a total time for the painting where items with type 1 take 20 seconds to paint and items of type 2 take 14 seconds to paint.

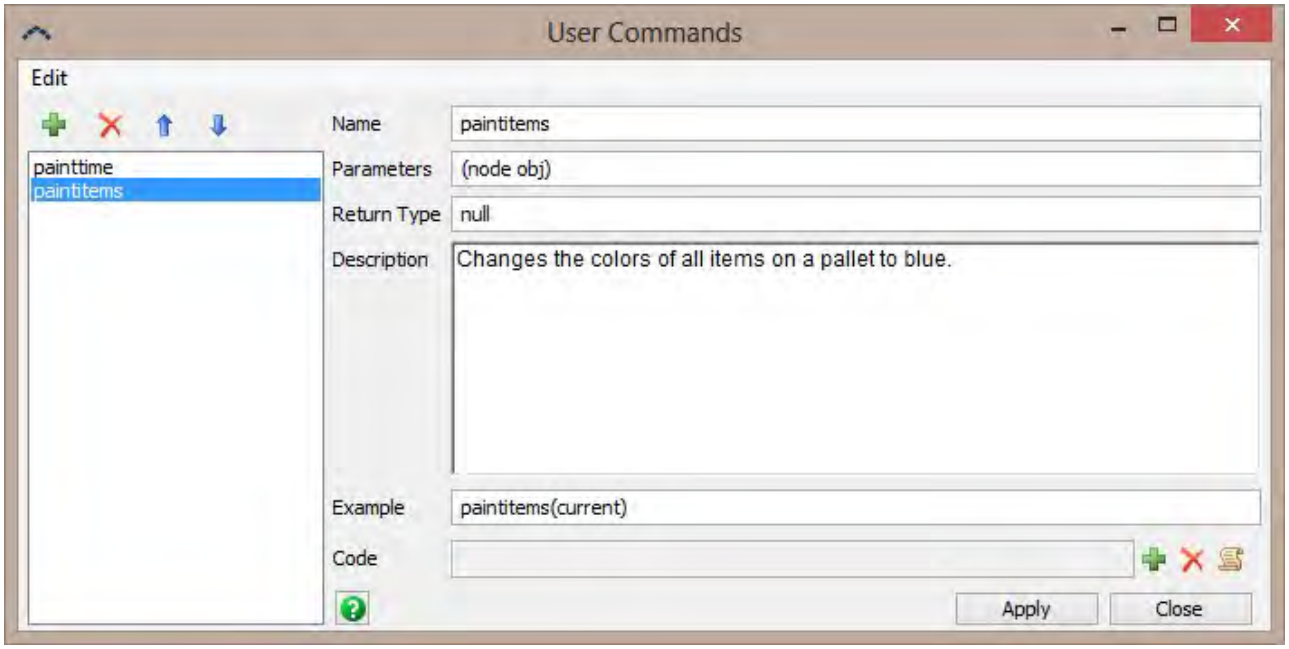
- Click on the button to open the code for our command.
- Enter the following code:

```
Object object = param(1); int painttime = 0; for(int index = 1; index <
object.first.subnodes.length; index++) {        if(object.first.subnodes[index].type == 1)  {
                painttime += exponential(0.0, 20.0, 0);
          } else {
                              painttime += exponential(0.0, 14.0, 0);
        }
}
return painttime;
```

- Click the OK button to apply and close the code window.

Next we will create a command that will change the color of all items on the pallet to be blue once they've gone through the painter.

- Add a new user command and set the Name to *paintitems*.
- Set the Parameters to *(node obj)*.
- Set the Return Type to *null*.
- The command's Description is: *Changes the colors of all items on a pallet to blue.*
- Set the Example to *paintitems(current)*.



For the code, we will cycle through all of the items on the pallet and change the color to blue.

- Click on the button to open the code for our command.
- Enter the following code:

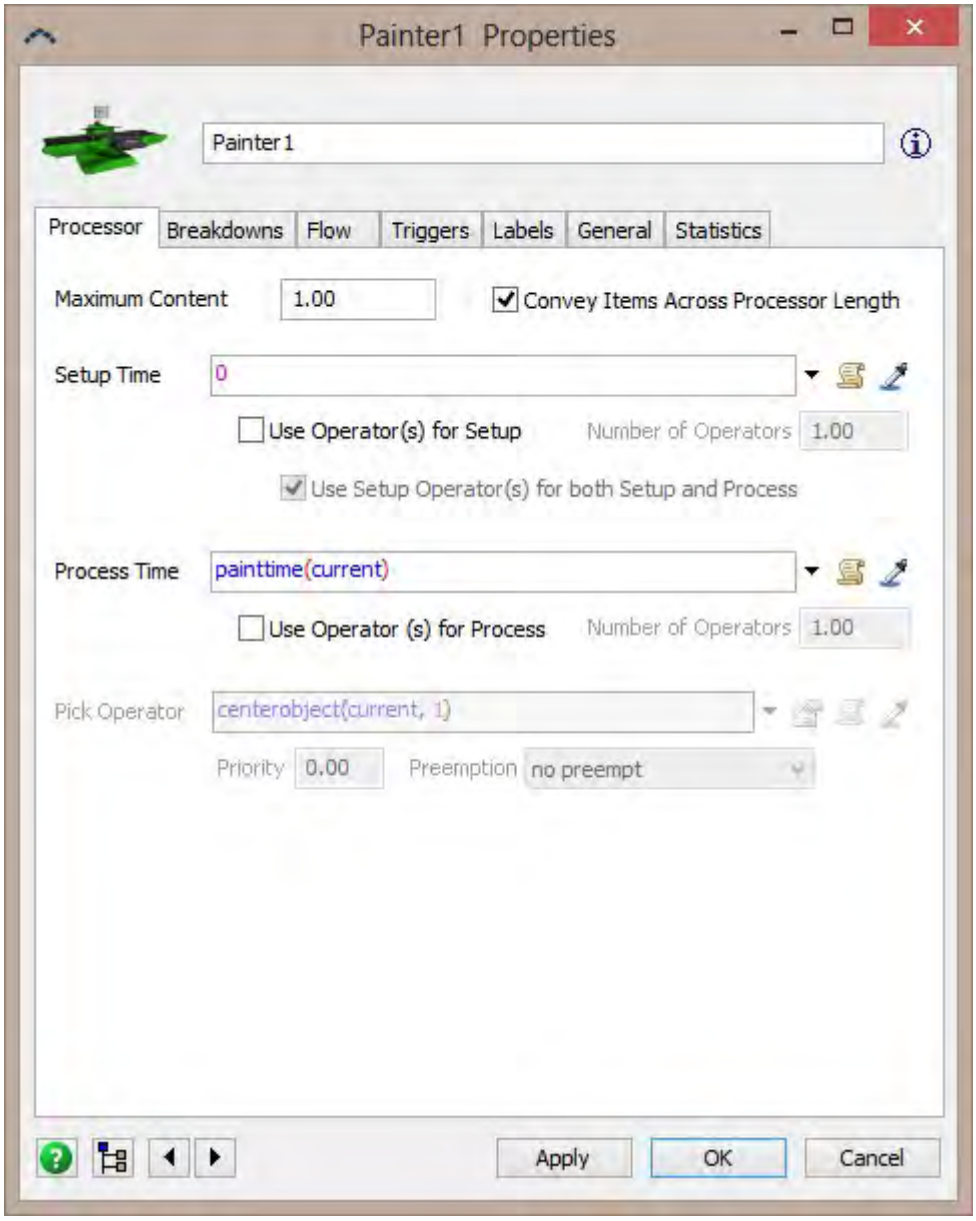
```
Object object = param(1);
for(int index = 1; index <= object.first.subnodes.length; index++) {
    colorblue(object.first.subnodes[index]); }
```

- Click the OK button to apply and close the code window.

- Click Apply and then Close on the User Commands window.

Step 11: Setup the Painters

- Open *Painter1's* properties window.
- Set the Process Time to *painttime(current)*.



- Click the Flow tab and check the Use Transport button.
- Set the transport to be *Forklift*.
- Click on the Triggers tab and click on the next to the OnProcessFinish trigger to edit the code of that trigger.
- Add the following line to the bottom of the code: paintitems(current);

- Click OK to apply and close to the code window.
- Click OK to apply and close the Properties window.

Set up the same properties for *Painter2*. For a fast way to copy properties from one object to another, see the Edit Selected Objects Utility.

Step 12: The Separator

As the *Separator* currently stands, we do not need to make any changes to its properties. By default, when the *Separator* receives a flowitem, it separates the container from the contents and then sends the container (pallet) through output port 1 and the contents through output port 2. If you wish to change these settings, go to the Flow tab of the *Separator* and change the Send To Port option. We are now ready to run the model.

Step 13: Reset and Run the Model

- Remember to hit the 

button to reset model parameters to their initial state.
- Click the 

button to start the simulation.

You should see flowitems entering the queue and then being moved by the Operator to one of the two processors. Red items will move to *Processor1* and green items will move to *Processor2*.

Pallets will be place on the *Combiner* and then wait until it receives its full batch of items from *Queue1* and *Queue2*. The *Forklift* will then move the pallet to *Painter1* or *Painter2*. Before exiting the painters, all of the items will turn blue. The pallet will then be moved to the *Separator* where the pallet will be separated from the items. The pallet will travel across *Conveyor1* and the blue items will travel across *Conveyor2* to the *Sink*.

This completes the Global Modeling Tools lesson. Congratulations!

User Events Tutorial Introduction

This tutorial will help you understand how to use User Events in your simulation. It assumes you know how to use Global Variables. If you don't, please see our Global Modeling Tools Tutorial.

What You Will Learn

- How to use the User Events Tool.

Approximate Time to Complete this Lesson

This lesson should take about 15-20 minutes to complete.

Model Overview:

In this model we will relocate operators every time we reset the model, as well as set the inter-arrival rate and number of processors to use. At a given time in the model we will change the rate of part arrivals and open up additional processors.

Click here for the Step-By-Step Tutorial.

User Events Tutorial Step-By-Step Model Construction

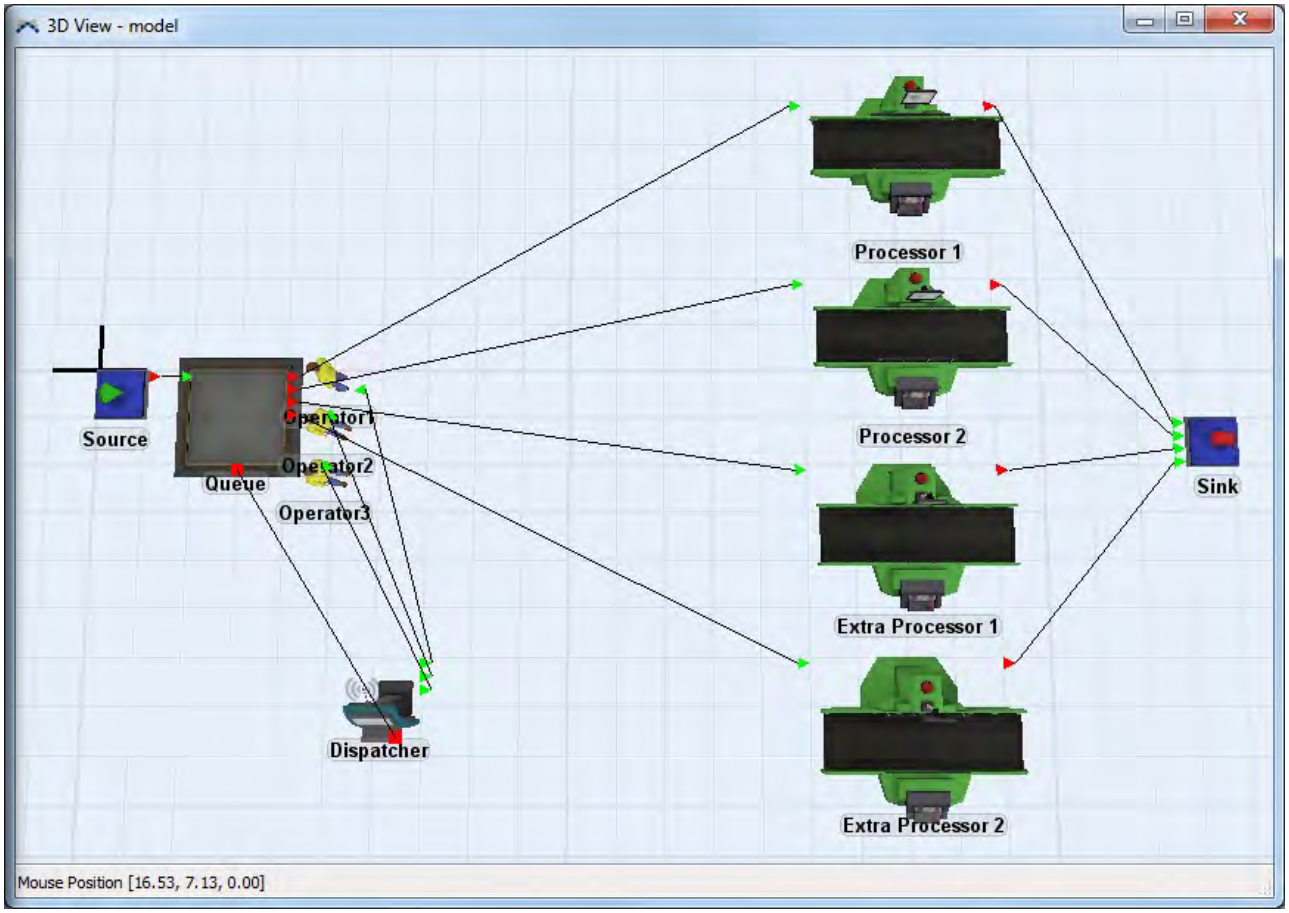
Building Labels Model

 button on the toolbar. Click OK on the Model Units window, we will Begin a new model by clicking the use the default units for our model.

If at any time you encounter difficulties while building this model, a fully functional tutorial model can be found at http://www.flexsim.com/tutorials

Step 1: Create the Objects

- Drag objects from your Library Icon Grid onto the 3D View to create the model shown below.
- Rename the objects as shown.



Connect all of the objects as shown:

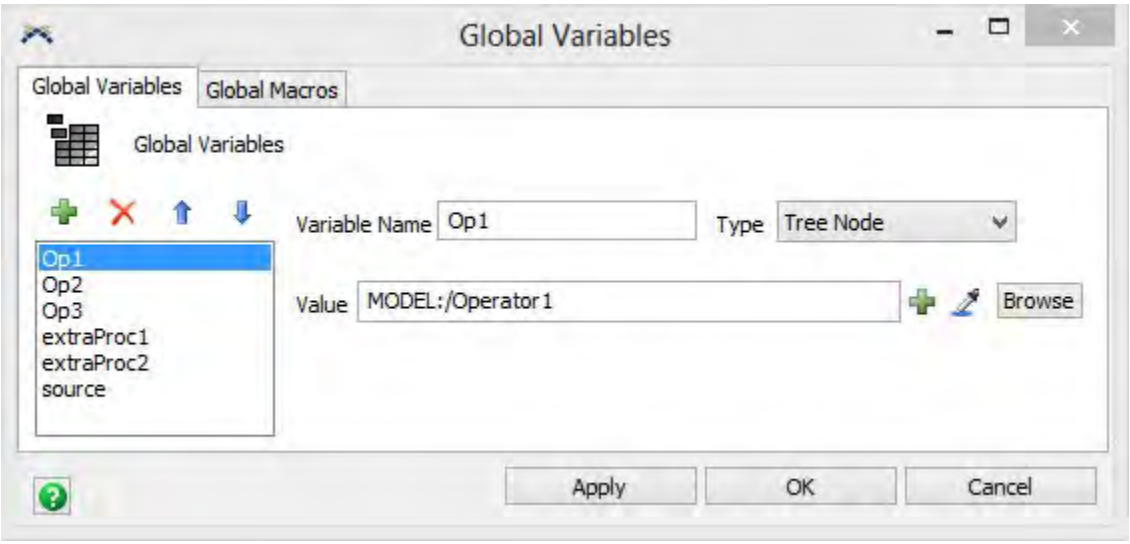
- Connect *Source* to *Queue*.
- Connect *Queue* to *Processor 1*, *Processor 2*, *Extra Processor 1* and *Extra Processor 2*.
- Connect *Processor 1*, *Processor 2*, *Extra Processor 1* and *Extra Processor 2* to *Sink*.
- Connect *Queue* to *Dispatcher* with a centerport connection (S key).
- Connect *Dispatcher* to *Operator1*, *Operator2* and *Operator3* with a standard connection (A key).

Step 2: Setup Global Variables

Here we will create global variables for objects that we will access in our User Event. Alternatively, objects can be accessed by the find command:

```
Object Op1 = model().find("Operator1");
```

- Add a new Global Variable from the Toolbox (View > Toolbox or click on the Tools button in the main toolbar).
- Create a global variable for each operator, the two extra processors, and the source. Each of them will be of type *Tree Node*, named as shown:



Step 3: Setup Source and Queue

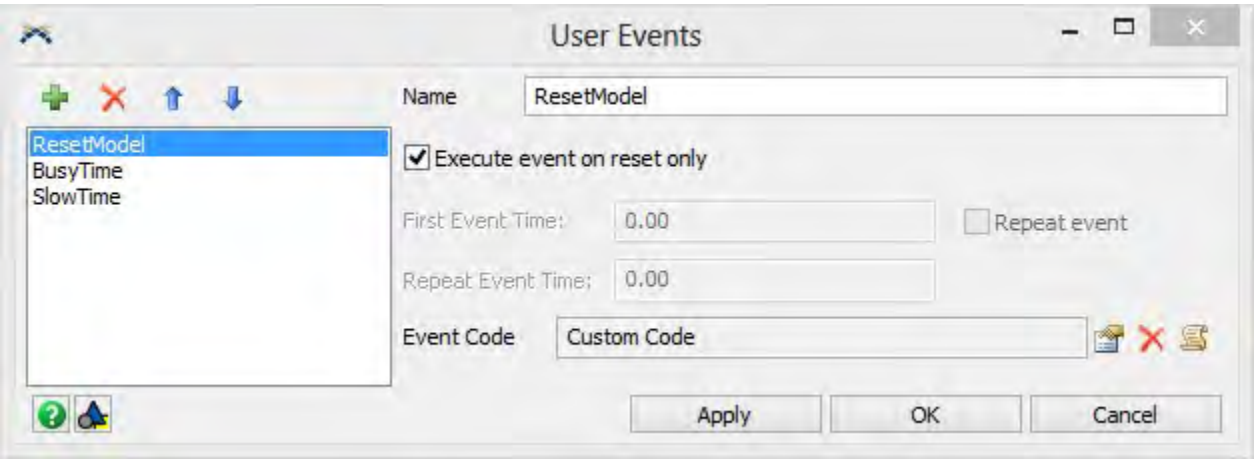
- Open the *Source* Properties Window.
- Go to the Labels tab.
- Create a number label called *"arrivalTime"*, leave it set to 0.
- Go to the *Source* tab.
- For the Inter-arrival time enter *current.arrivalTime*.
- Click Ok to apply and close the properties window.

- Open the *Queue* properties window.
- Go to the Flow tab.
- Check Use Transport.

Step 4: Create User Events

We will create three user events. One to fire when the model resets and two to change the "busyness" of the model.

- Add three new User Events from the Toolbox (View > Toolbox or click on the Tools button in the main toolbar).
- Name them *"ResetModel"*, *"BusyTime"* and *"SlowTime"*.



- With *ResetModel* selected, check Execute event on reset only. Every time you press the Reset button, this code will be executed.
- Click the code edit button and write in the following code:

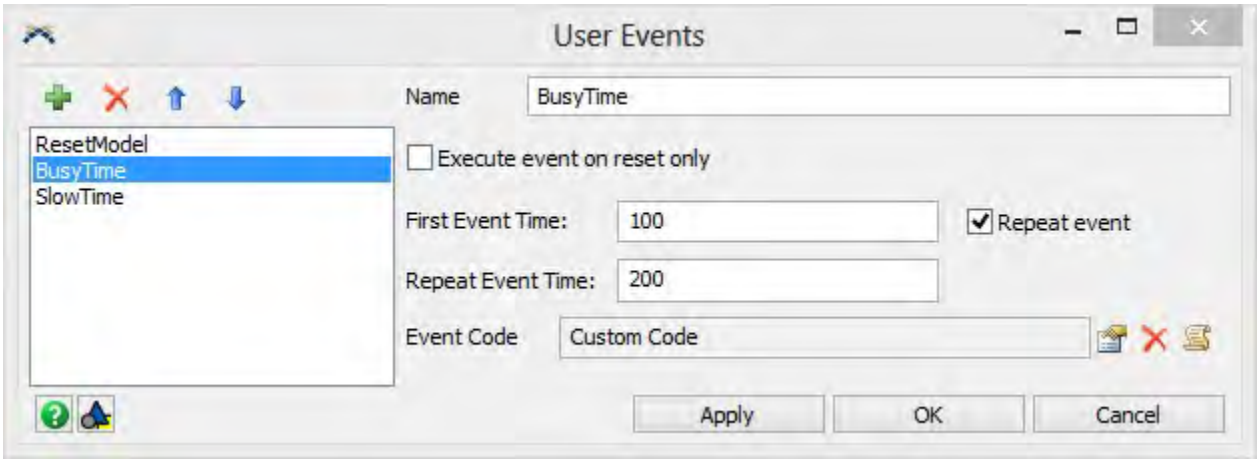
```
Op1.as(Object).setLocation(5, 0, 0);
Op2.as(Object).setLocation(5, -1, 0); Op3.as(Object).setLocation(5, -2, 0);
source.arrivalTime = 10;
```

```
1 /**Custom Code*/
2 Object current = ownerobject(c);
3
4 Op1.as(Object).setLocation(5, 0, 0);
5 Op2.as(Object).setLocation(5, -1, 0);
6 Op3.as(Object).setLocation(5, -2, 0);
7 source.arrivalTime = 10;
```

.as(Object)
The .as(Object) gives you access to object properties and methods. The setLocation() method is only available on object types.

This will set the location of each operator next to the queue (if your queue isn't in the same place as mine [x:2.00, y:0.00, z:0.00], check the location of your own queue and enter appropriate x, y and z values). This user event will also set the label on the source to 10, so the Inter-arrival time will change accordingly.

- With *BusyTime* selected, enter *100* for First Event Time and *200* for Repeat Event Time. Every 200 seconds the model will go into busy time, starting at time 100.

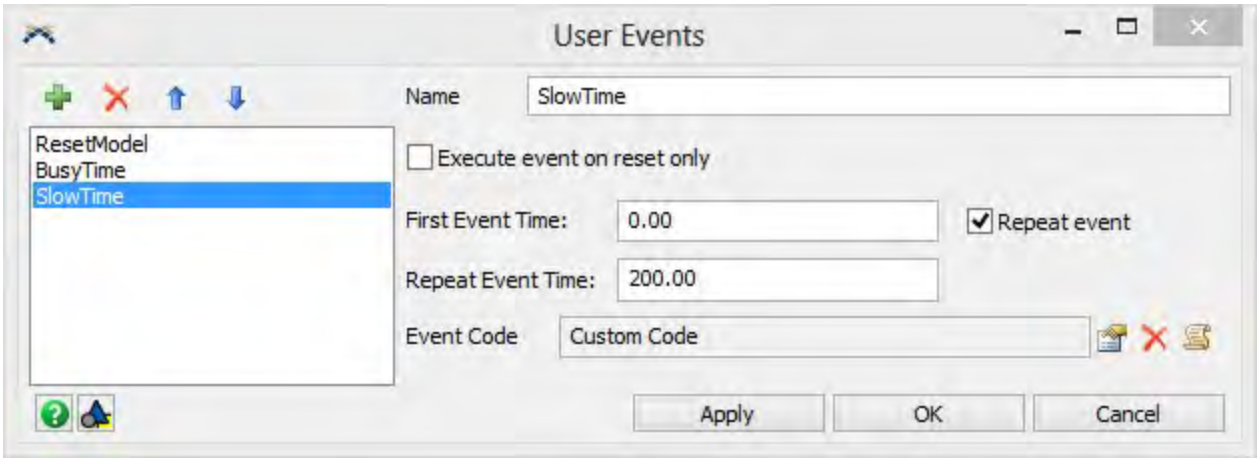


Click the code edit button and write in the following code:

```
source.arrivalTime = 5; openinput(extraProc1);
openinput(extraProc2);
msg("Busy Time", "It's Busy Time!");
```

This will set the inter-arrival time to 5, so parts are created twice as fast. The extra processors will open so they can be used during busy time. You will also get a message telling you it is busy time.

- With *SlowTime* selected, enter *0* for First Event Time and *200* for Repeat Event Time. Every 200 seconds the model will go into slow time, starting at time 0.

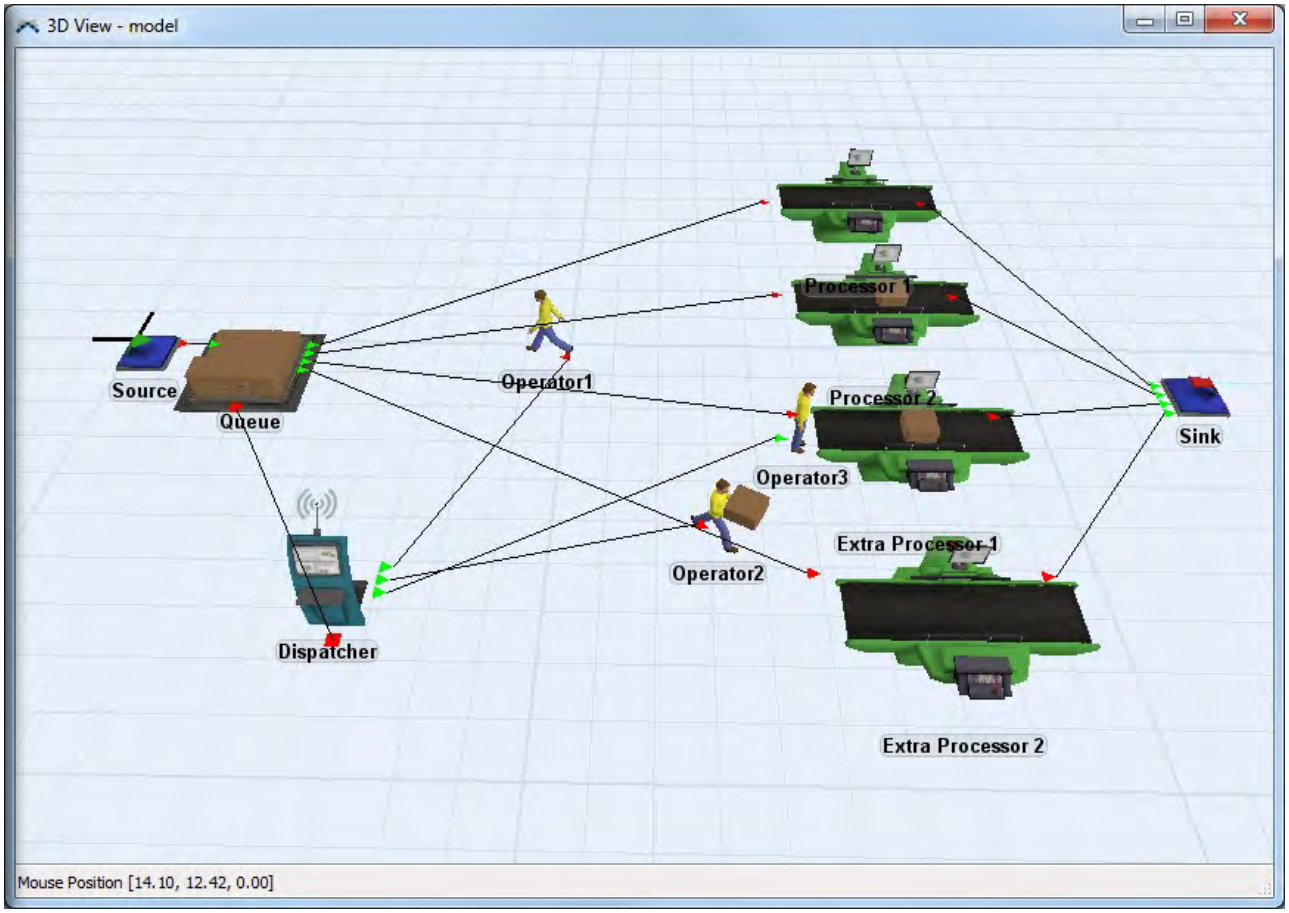


- Click the code edit button and write in the following code:

```
closeinput(extraProc1);
closeinput(extraProc2);
source.arrivalTime = 10;
```

This will set the inter-arrival time to 10, so parts come slow again. The extra processors will closed during slow time.

- Press OK on the User Events window to close and apply your changes.
- Reset and run your model. Notice that the operators are always reset to the same place and it switches between busy and slow times.



This completes the User Events tutorial. Congratulations!

Time Table Tutorial Introduction

This tutorial will introduce you to the TimeTable tool. The TimeTable can be used to specify specific times when a FixedResource or a TaskExecuter are scheduled to be down. This could be due to a break, maintenance, repairs, etc.

What You Will Learn

- How to create TimeTables and assign members.
- How to use a TimeTable to specify down times for your processors and operators.

Approximate Time to Complete this Lesson

This lesson should take about 15-20 minutes to complete.

Model Overview

In this model we will have multiple operators performing a task. A TimeTable will be created to specify when an operator is on break. Another TimeTable will schedule maintenance for the processors. Click here for the Step-By-Step Tutorial.

Time Table Tutorial Step-By-Step Model Construction

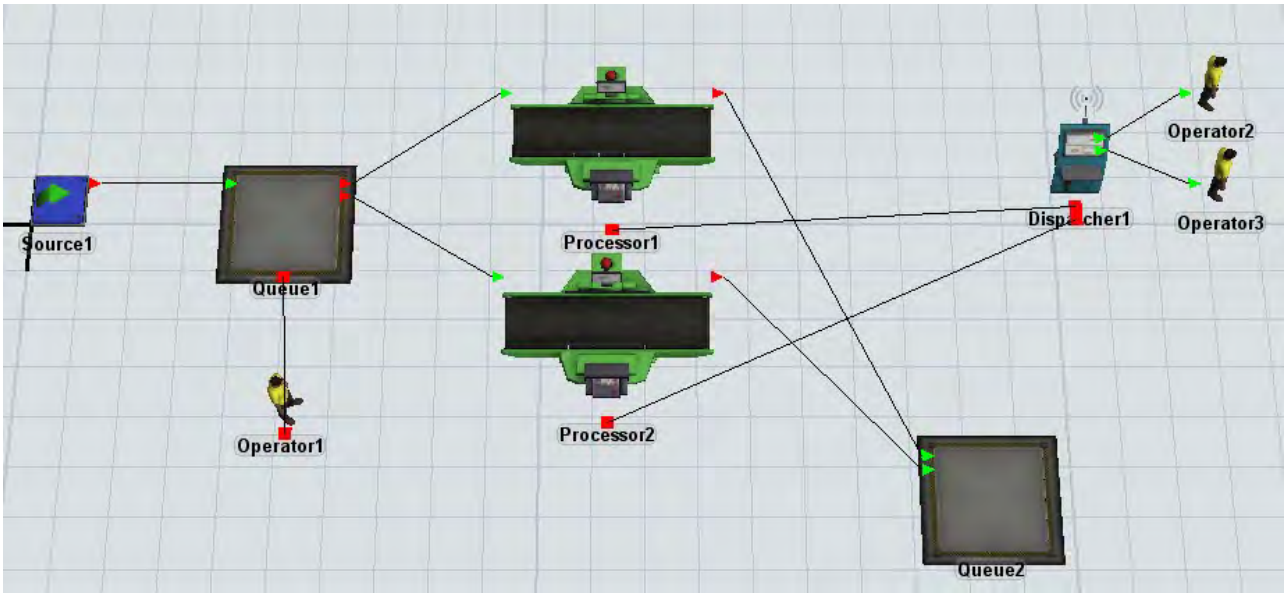
Building TimeTable Model


button on the toolbar. Click OK on the Model Units window, we will Begin a new model by clicking the use the default units for our model.

If at any time you encounter difficulties while building this model, a fully functional tutorial model can be found at http://www.flexsim.com/tutorials

Step 1: Create the Objects

- Drag objects from your Library Icon Grid onto the 3D View to create the model shown below.



Connect all of the objects as shown:

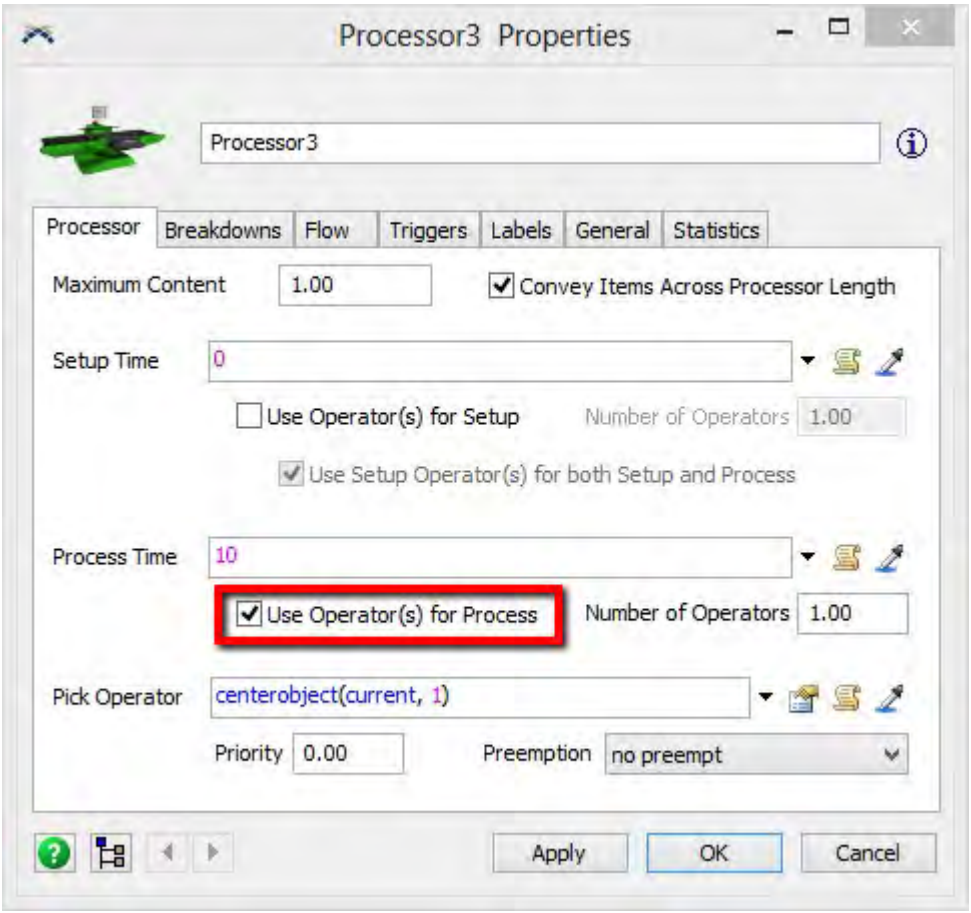
- Connect *Source1* to *Queue1*.
- Connect *Queue1* to *Processor1* and *Processor2*.
- Connect *Processor1* and *Processor2* to *Queue2*.
- Connect *Dispatcher1* to *Operator2* and *Operator3*.

- Connect *Queue1* to *Operator1* with a centerport connection (S key).
- Connect *Processor1* and *Processor2* to *Dispatcher1* with a centerport connection (S key).

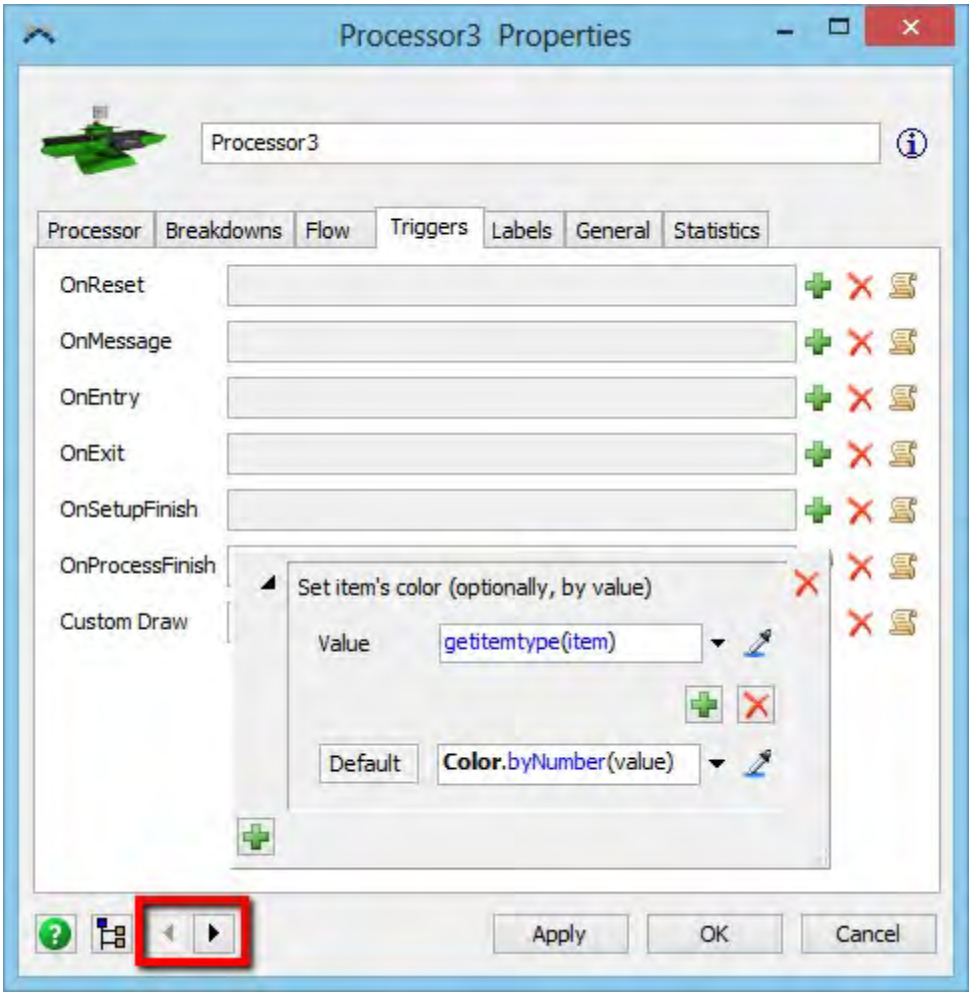
Step 2: Setup the Queue and Processors

One operator will transport flowitems from *Queue1* to the two processors. The other two operators will be used to process flowitems at the two processors and to transport flowitems from the processors to *Queue2*.

- Click on *Queue1* to open its properties in the Quick Properties window.
- Under the Flow section, check Use Transport and leave it using the default centerobject option.
- Open the properties window of *Processor1*.
- On the Processor tab, check Use Operator(s) for Process and leave it using the default centerobject option.



- Go to the Flow tab.
- Check Use Transport and leave it using the default centerobject option.
- Go to the Triggers tab.
- Click the add button for the OnProcessFinish trigger.
- Select the *Set Color* option and leave it on the default options.



Note: You can switch quickly between processors (or other similar objects) by clicking on the left and right arrow buttons at the bottom of any properties window.

- Repeat the last set of steps for *Processor2*.
- Click OK to apply and close the properties window.

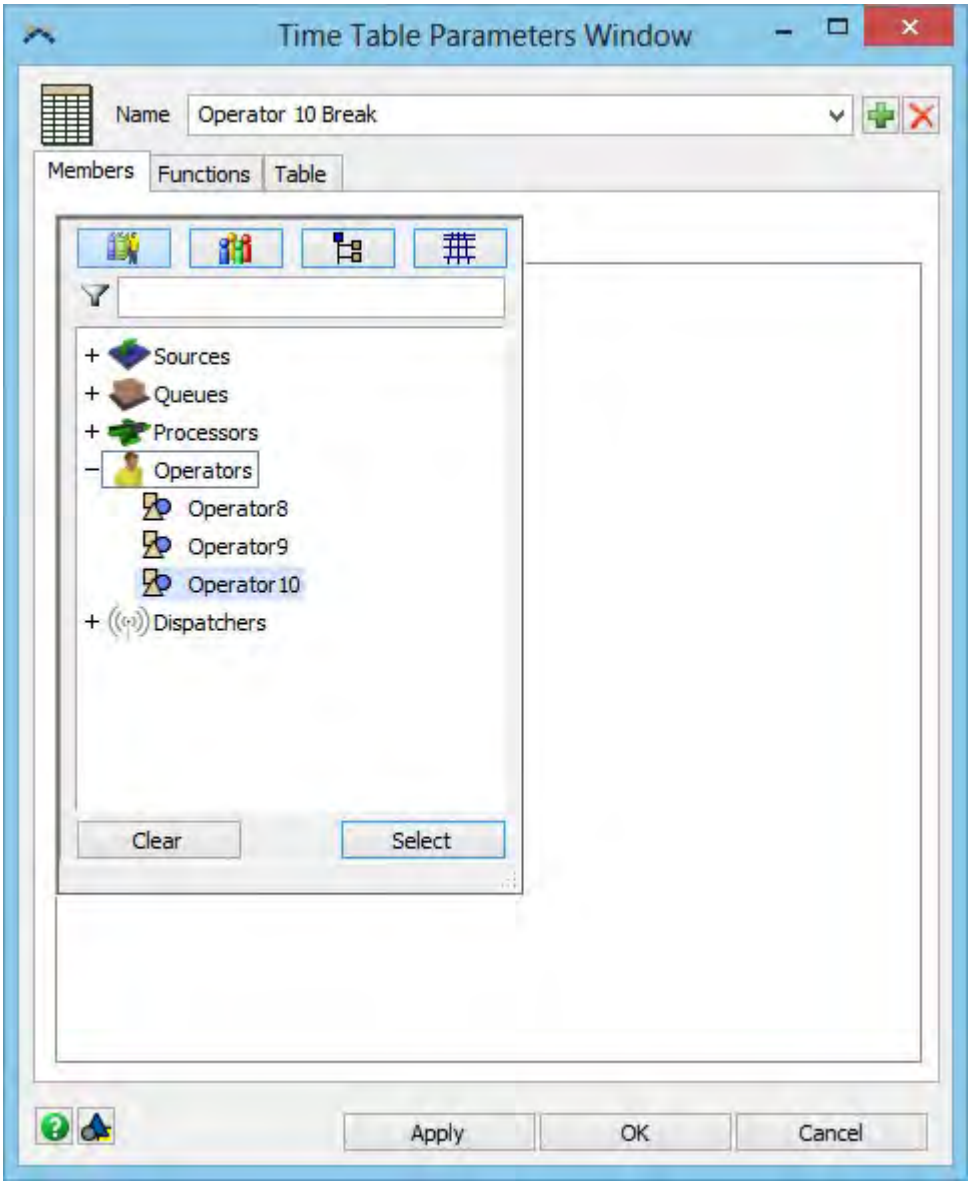
Reset and run the model to make sure the operators are taking the boxes from *Queue1* to the processors, processing the boxes and then taking them to *Queue2*. The boxes should also be changing color after processing.

Step 3: Create a TimeTable

We will now create a TimeTable for *Operator1*.

- Add a new Time Table from the Toolbox (View > Toolbox or click on the Tools button in the main toolbar).
- Rename the TimeTable to *Operator 10 Break*.
- Go the Members tab and click the button to add a member. Highlight *Operator1* and click Select.
- Go to the Table tab and in row 1 of the table, set Time to *200*, set State to *12*, and set Duration to *30*.

- Set the Repeat to Custom and change the value to *200*. This will cause the operator to go on break every 200 seconds.
- Go to the Functions tab and in the pick list for the Down Function, select *Travel to Location, Delay Until Down Time Complete*. Change only the Location to *2, -8, 0*. These are the x, y and z coordinates, respectively, that the operator will go to while on break.



- In the pick list for the Resume Function, select *Do Nothing*.
- Click OK to apply and close the TimeTable window.

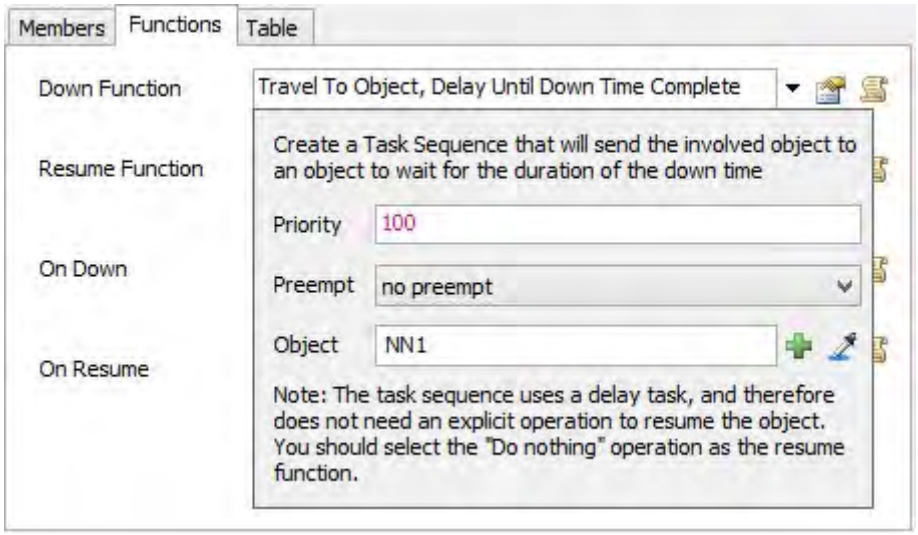
Reset and Run your model. When you run you model, at 200 seconds you will see that the operator walks away from the work area for 30 seconds and then returns to work.

Note: If the operator is in the middle of a task, he will complete the task before starting his down time.

Step 4: Update the TimeTable

We can also have the operator break to a specific object instead of a set of coordinates.

- Create a *Network Node* object and place it away from your work area.
- Go back to your *Operator 10 Break* TimeTable.
- In the pick list for the Down Function, select *Travel to Object, Delay Until Down Time Complete*. Change the Destination Name to *NN1*.



- Leave the pick list for the Resume Function on *Do Nothing*.
- Click OK to apply and close the TimeTable window.
- Reset and Run your model and notice that the operator will go to *NN1*.

WARNING: The duration of your operator's break will be the time he spends at that location. If he takes 10 seconds to walk there, he will still break for 30 seconds once he gets there. Adding in his 10 second journey back, he will total 50 seconds taken between finishing his last task and starting his new one.

Step 5: Schedule Maintenance for the Processor

Next, we will give Processor1 a scheduled downtime for maintenance.

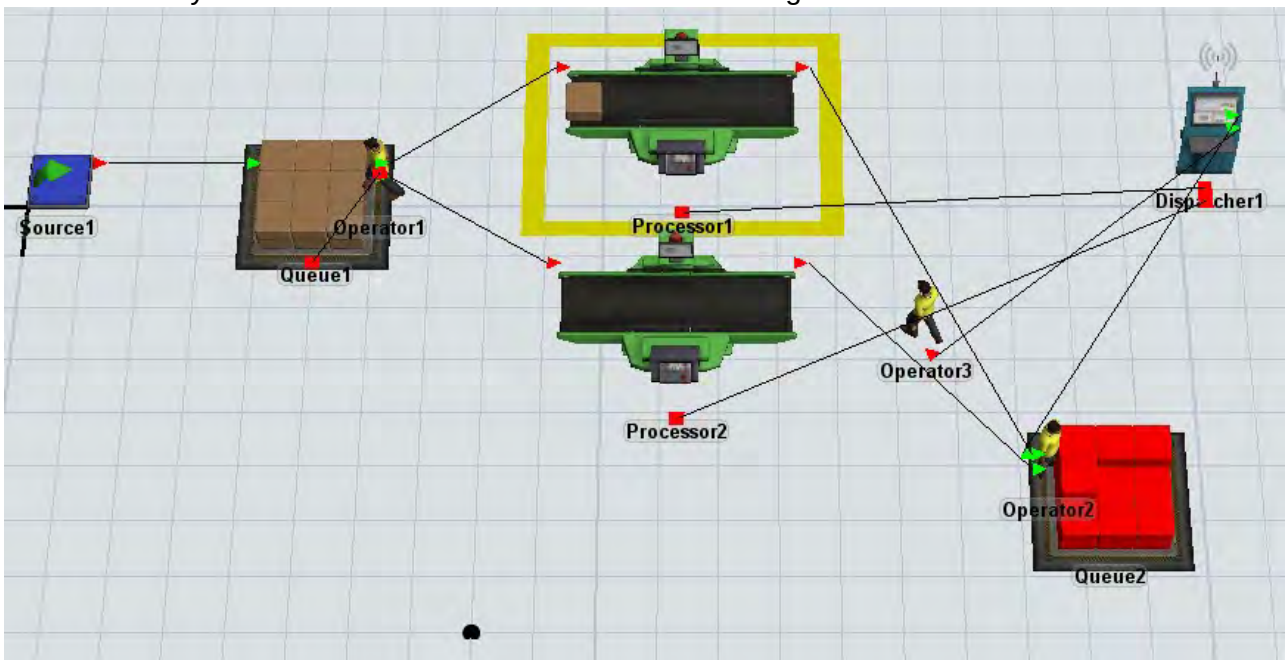
- Add another TimeTable.
- Rename the TimeTable to *Processor Down Time*.
- Add the member *Processor1*.
- Under the Table tab, in Row 1 of the table, set Time to *200*, set State to *12* and set Duration to *100*.
- Set the Repeat Time to Custom, *300*. This will cause the processor to go down every 300 seconds after the initial maintenance.
- In the pick list for the Down Function, select *Stop Input*.
- In the pick list for the Resume Function, select *Resume Input*.

Stopping and resuming the object's input will mean that it will continue to process any item that is currently in it, but will not receive anymore items until the down time is done. If you leave the TimeTable at the default

Stop/Resume Object, the processor will stop with or without an item in it and not resume until the down time is complete.

- Click OK to apply and close the TimeTable window.

Reset and run your model. Your model should look something like this:



This completes the TimeTables tutorial. Congratulations!

Kinematics Tutorial Introduction

This tutorial will introduce you to using Kinematics in your model. Kinematics allow you to perform simultaneous movements with a single object. This is best seen by looking at the way the Crane object moves.

What You Will Learn

- How to create and add Kinematics to a fixed resource object.
- How to move an object using Kinematics.

Approximate Time to Complete this Lesson

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

Model Overview

In this model we will treat a processor as if it is a centrifuge and have it spin as it processes flowitems. Click here for the Step-By-Step Tutorial.

Kinematics Tutorial Step-By-Step Model Construction

Building Kinematics Model

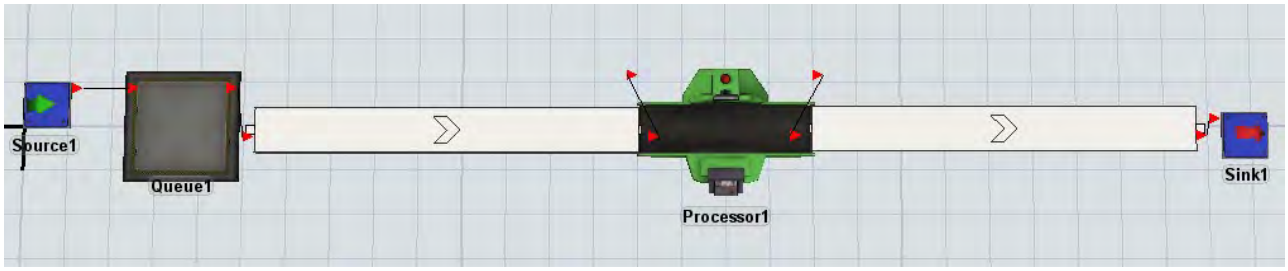
 button on the toolbar. Click OK on the Model Units window, we will Begin a new model by clicking the use the default units for our model.

If at any time you encounter difficulties while building this model, a fully functional tutorial model can be found at http://www.flexsim.com/tutorials

Part 1: Basic Kinematics

Step 1: Create the Objects

- Drag objects from your Library Icon Grid onto the 3D View to create the model shown below.



Connect all of the objects as shown:

- Connect *Source1* to *Queue1* to *Conveyor1* to *Processor1* to *Conveyor2* to *Sink1*

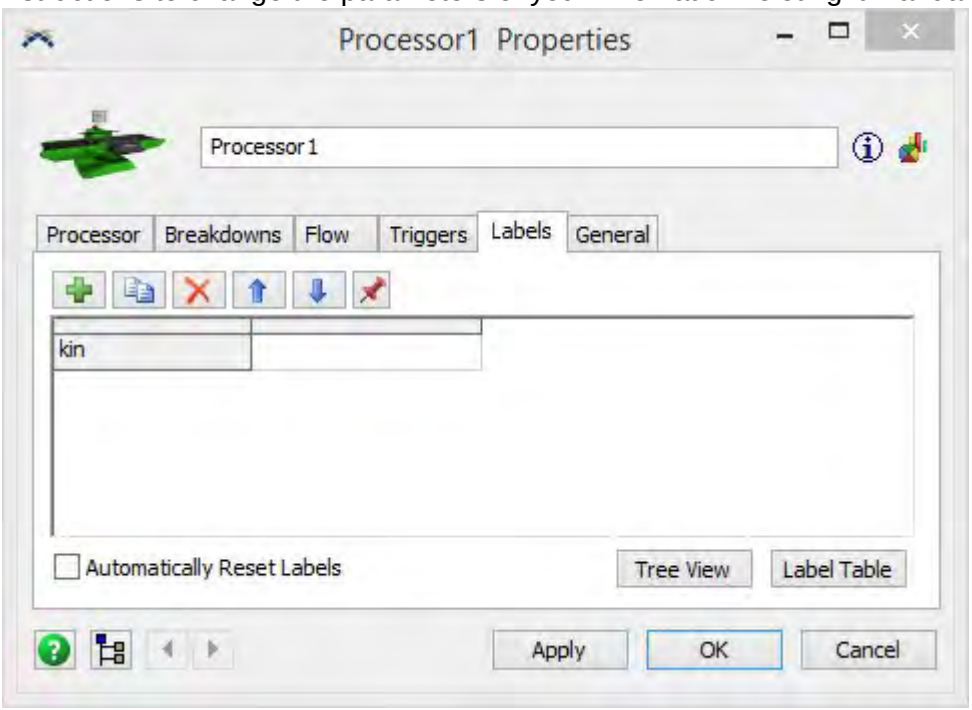
Step 2: Add Kinematics Label

All kinematics need a unique node for the information to be stored. The simplest way to do this is by creating a text label dedicated to the kinematics.

Alternatively, a node could be added to the object's variables, in which case of all the *label* commands would be replaced with *getvarnode*.

- Open the properties window of the *Processor*.
- Go to the Labels tab.
- Add a text label by clicking the Add Text Label and rename the label *"kin"*.

WARNING: Do not try to delete the kinematics node from within the properties window. Use the following instructions to change the parameters of your kinematic. Deleting it manually may crash the program.

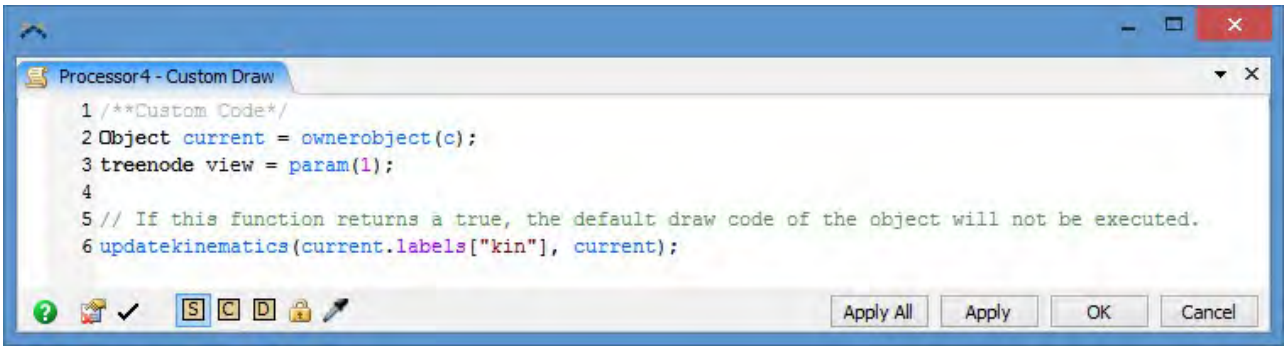


Step 3: Add Custom Draw Code

The kinematics will need to be updated continually as the model runs.

- Go to the Triggers tab.
- Click the code edit button next to the Custom Draw trigger.
- Enter the following line of code:

```
updatekinematics(current.labels["kin"], current);
```



- Click OK to apply and close the code edit window.

Step 4: Update Kinematics when Process Finishes

The kinematics will need a final update at the time they should be complete. This will ensure that, regardless of the framerate of your 3D view (which defines how often the Custom Draw trigger fires), or whether you have any 3D views open at all, the kinematic will still be properly updated to its final resting position/rotation.

- In the Triggers Tab, click the code edit button next to the OnProcessFinish trigger.
- Copy the same *updatekinematics* command used in the Custom Draw trigger:

```
updatekinematics(current.labels["kin"], current);
```

- Click OK to apply and close the code edit window.

Step 5: Add OnReset Code

You want your object to return to its original position when you reset the model.

- In the Triggers tab, click the code edit button next to the OnReset trigger.

- Use the *initkinematics* command with the node being the text label you just created. Switch over to the General tab of the properties window and check the x, y, and z position and rotation of the object. Set the corresponding values in the *initkinematics* command. (The rotation values of your processor will be 0 by default.)
- Set the last two parameters to *0*.

```
initkinematics(current.labels["kin"], x, y, z, 0, 0, 0, 0, 0);
```

Note: The last two parameters will indicate rotation management and local coordinates. When rotation management is set to 1, your object will rotate so the the "front" (the positive x-direction) of it is facing the direction of travel. If local coordinates is set to 1, it will use the coordinates of the object's container instead of the model itself.

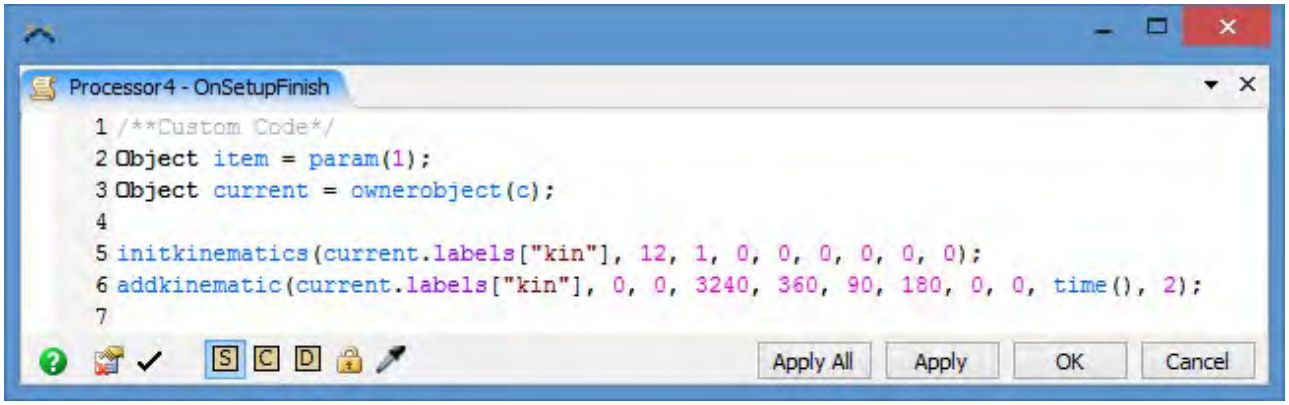
- Click OK to apply and close the code edit window.

Step 6: Update Kinematics when Setup Finishes

You will also want your object to return to its original state at the start of the kinematic. We will now add the kinematic information that will move the object.

- In the Triggers tab, click the code edit button next to the OnSetupFinish trigger.
- Copy the *initkinematics* command with the same parameters as in the OnReset trigger.
- Enter the following line of code:

```
addkinematic(current.labels["kin"], 0, 0, 3240, 360, 90, 180, 0, 0, time(), 2);
```



The *addkinematic* command sets the x, y, and z parameters, to 0, 0, and 3240, respectively. This will be a rotational motion so it will rotate around the z axis 3240 degrees (9 turns). The target speed (or maximum speed) is set to 360 degrees/sec, with an acceleration of 90 degrees/sec/sec and deceleration of 180 degrees/sec/sec. The start speed and end speed are 0. The start time will be the time that the command is called, so we use the time() command. Last of all, we want this to be a rotational motion so we set the parameter to 2 or KINEMATIC_ROTATE (for translational motion we would set this to 1 or KINEMATIC_TRAVEL).

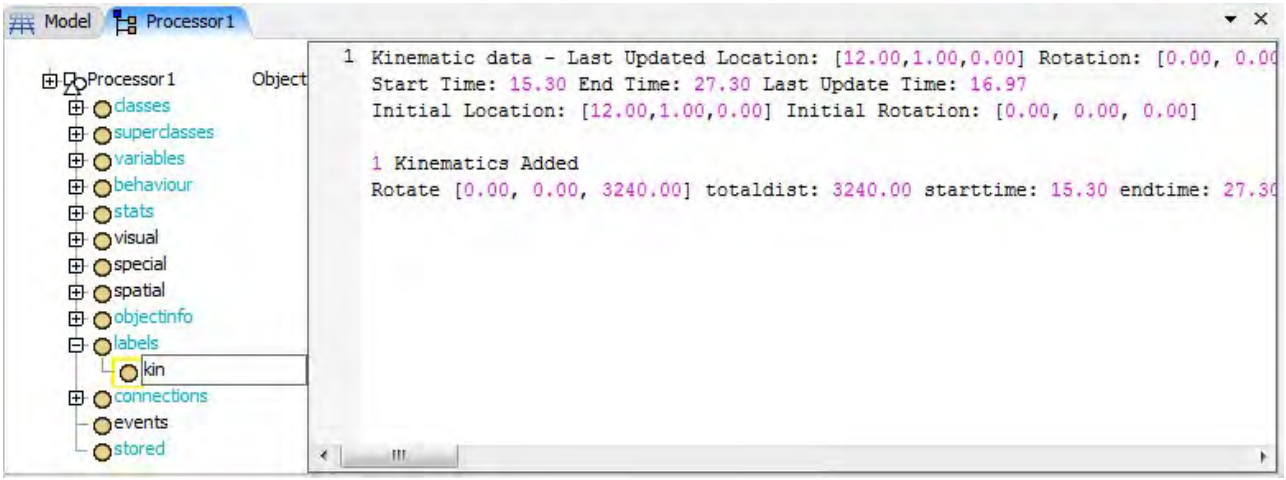
- Click OK to apply and close the properties window.

Reset and run the model and you should see the processor accelerate, spin and decelerate to a stop. You may notice that if the next flowitem starts being processed before your kinematic is done that it will instantaneously reset its position to match the *initkinematics* parameters. Our next step will be to match the process time with the time it takes for you kinematic to end.

Step 7: View the Kinematic Information

Important information about your kinematic will be stored in the *"kin"* label.

- Run the model until a flowitem enters the processor and stop the model. DO NOT reset.
- Right click on the *Processor* and click the Explore Tree button.
- Expand the processor tree, then expand the *"labels"* node within that tree. Click on the *"kin"* label and you will see the information for the kinematic as in the following figure:



- Scroll to the right until you can see the starttime and endtime. The difference between the two is the time it takes for the kinematic to complete. You should see a difference of 12 seconds.
- Open the *Processor's* properties window.
- Under the Processor tab, change the Process Time to *12*.
- Optional: Adding a Setup Time of a second or two will allow our item to move out of the processor before the processor begins spinning again.

When you run the model now you should see that the processor finishes processing when it is done spinning. Kinematics lets you set up your simulation in terms of speeds and accelerations of your equipment as well as giving you the visual.

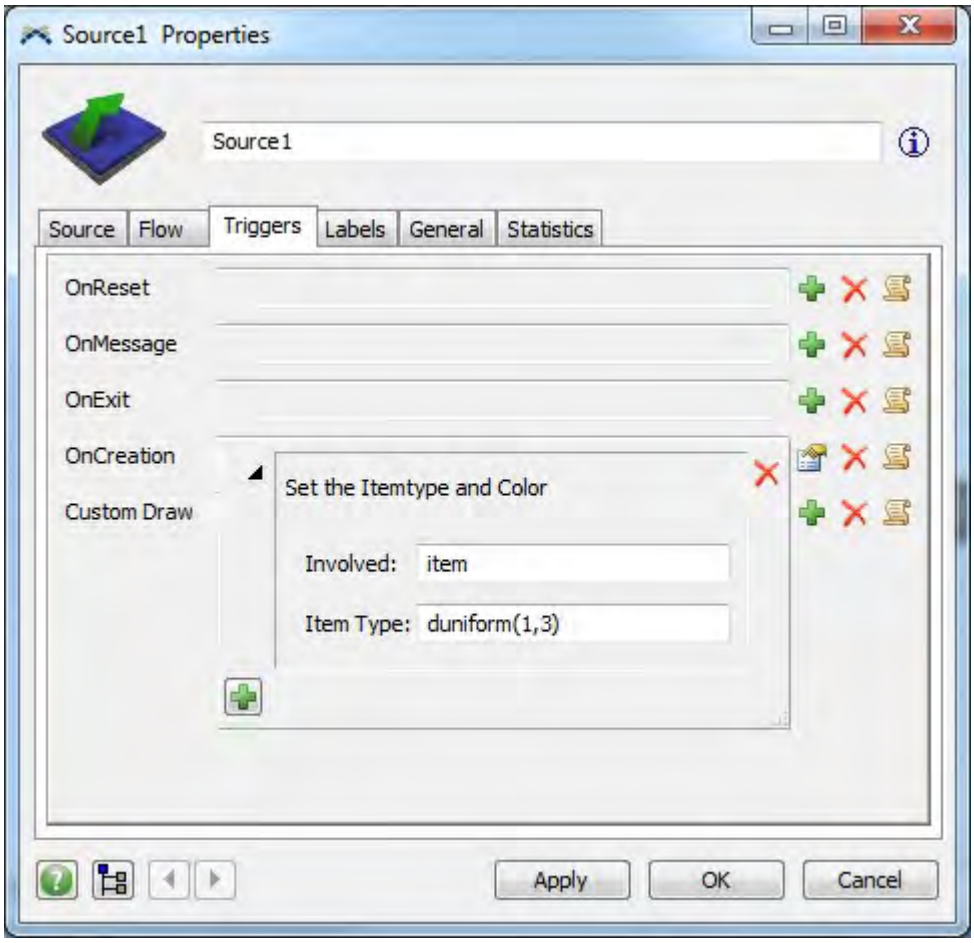
Kinematics can be used to do mutliple simultaneous movements to a single object. Try adding more *addkinematics* commands to the OnSetupFinish trigger.

Part 2: Update Kinematics Dynamically

In this part of the tutorial, we will have the processor spin at different speeds according to the item type.

Step 8: Create Multiple Item Types

- Open the properties window of the *Source*.
- Go to the Triggers tab.
- Click on the add button the next to the OnCreation trigger.
- Select the *Set Item Type and Color* pick option.
- Leave the values as default.

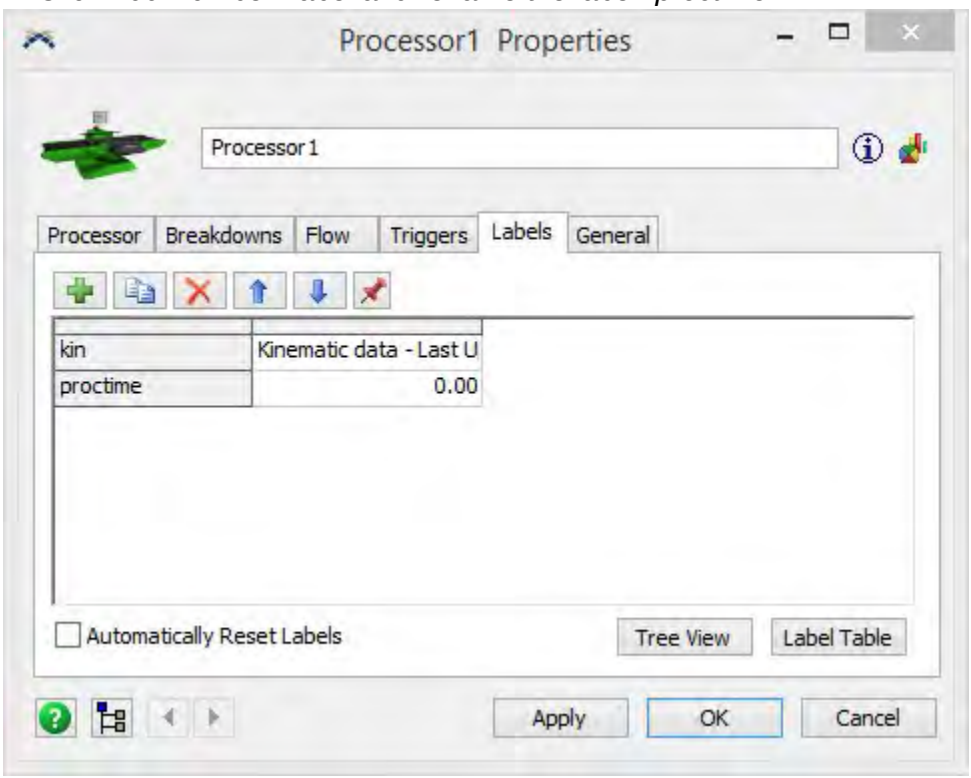


- Click OK to apply and close the properties window.

Step 9: Process Each Item Type Differently

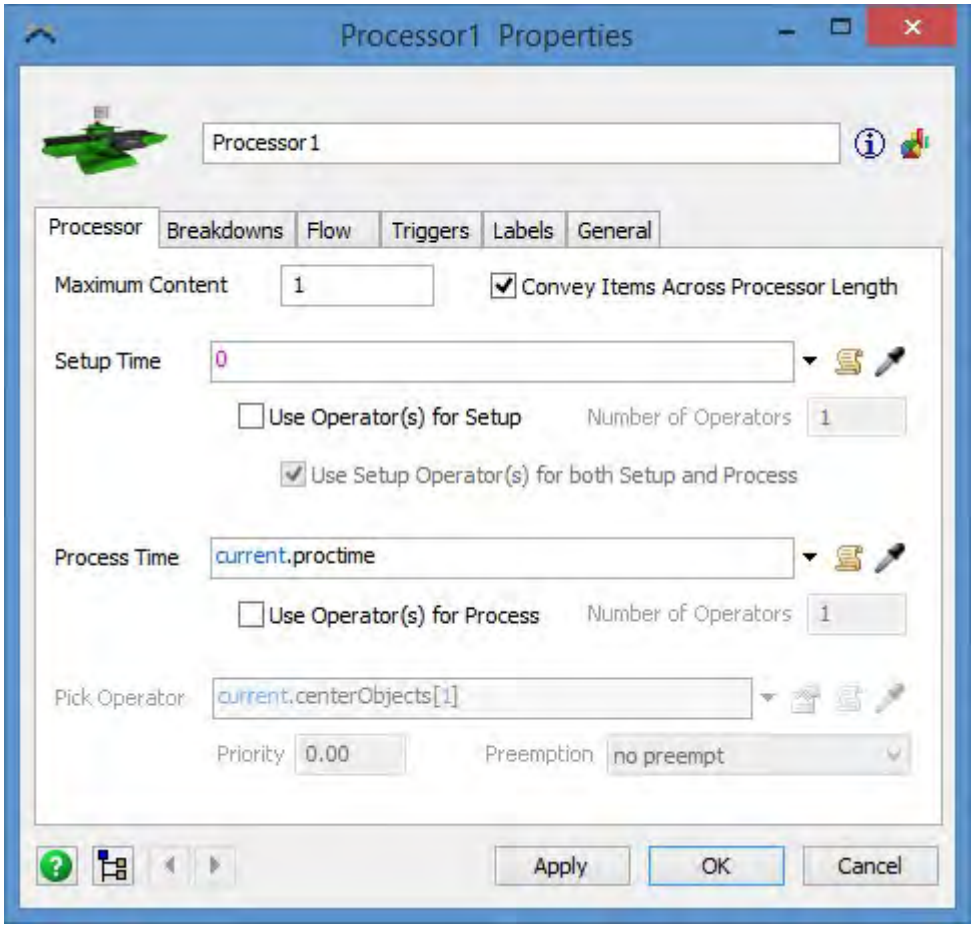
Each item type will be processed at a different speed and therefore require a different process time. Since this will change dynamically as the model runs, we will make a number label to keep track of the information.

- Open the properties window of the *Processor*.
- Go to the Labels tab.
- Click Add Number Label and rename the label *"proctime"*.



- Go to the Processor tab.
- Enter the following command into the Process Time field:

current.proctime



Step 10: Create Dynamic Kinematics

We need to customize the kinematics so that it changes for each item type.

- Go to the Triggers tab.
- Click the code edit button for the OnSetupFinish trigger.
- Enter the following code:

```
 initkinematics(current.labels["kin"], 13, 1, 0, 0, 0, 0, 0, 0); int z; int speed; int type =
item.type; switch (type)
{
    case 1:          z
= 1080;     speed =
360;          break;
    case 2:          z
= 3240;     speed =
360;          break;
    case 3:          z
```

= 3240; speed =

180; break;

}

addkinematic(current.labels["kin"], 0, 0, z, speed, 90, 180, 0, 0, time(), 2);

The *initkinematics* command should be the same as in Part 1. The *addkinematics* command is placed within an if statement that checks the item type for each item as the setup finishes. Try playing with the zrotation value as well as the maxspeed, acceleration, and deceleration values of the processor.

Step 11: Update the Process Time

We will now get the kinematic information from our object and use it to change the processing time of the processor.

- While still in the custom code for the OnSetupFinish trigger, add the following code:

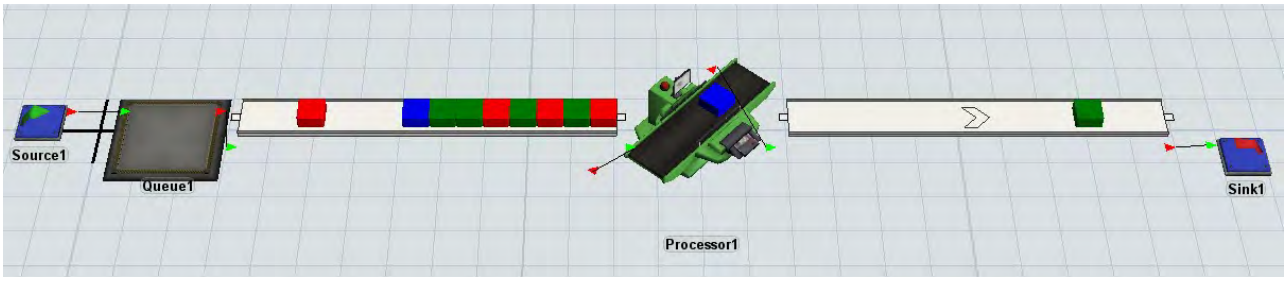
double endtime = getkinematics(current.labels["kin"], KINEMATIC_ENDTIME); double starttime = getkinematics(current.labels["kin"], KINEMATIC_STARTTIME); double proctime = endtime - starttime; current.proctime = proctime;

The *getkinematics* command gets information from the last update made to the kinematic. The available information that can be grabbed is described in the Kinematics section of the help manual.

If you have more than one kinematic at a time, you can set which kinematic it pulls information from, and even get information at specific distances or travel times in the middle of the kinematic action. See the manual for details.

- Click OK to apply and close the code edit window.
- Click OK to apply and close the properties window.

Reset and run your model. Your model should look similar to this:



This completes the Kinematics tutorial. Congratulations!

Task Sequence 1 Tutorial Introduction

In this tutorial, you will learn how to build a basic task sequence from scratch. The operator will pick up the flowitem from a queue, take it to a table to inspect the item, then take the item to a processor. Writing your own task sequence will allow you to allocate and use one Operator for the entire task. This tutorial assumes a solid knowledge of basic interaction with the software, and will ask you to write several lines of code.

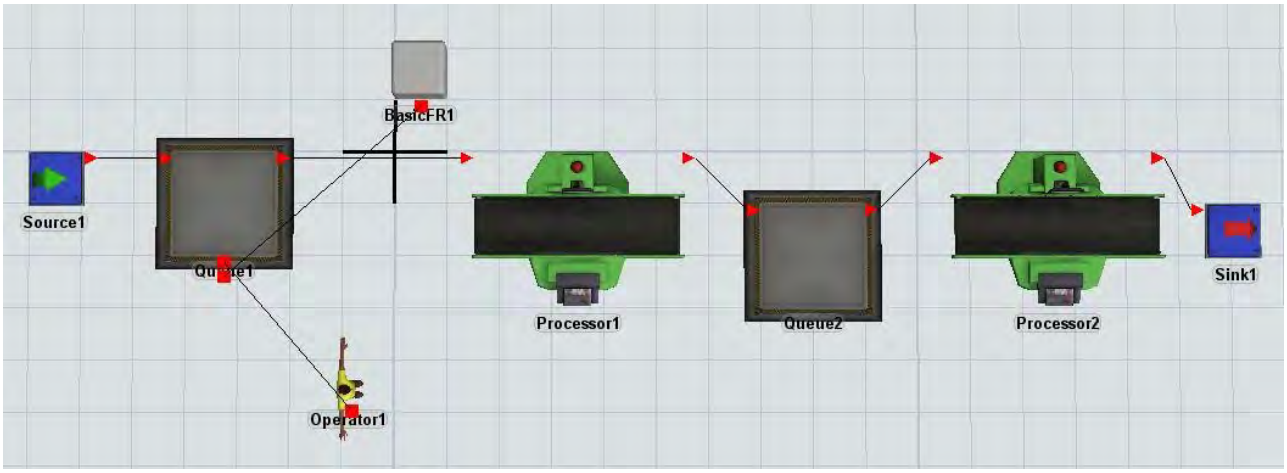
If at any time you encounter difficulties while building this model, a fully functional tutorial model can be found at http://www.flexsim.com/tutorials

For more on Tasktypes and their parameters, see the Task Types page. Click here for the Step-By-Step Tutorial.

Task Sequence 1 Tutorial Step-By-Step Model Construction

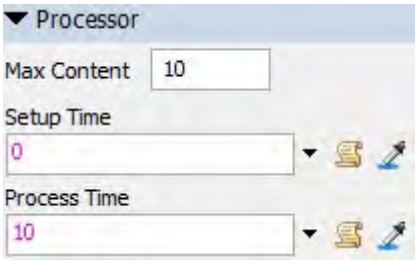
Step 1: Setup the model

- Create a Source, a Queue, a BasicFR, a Processor, a Queue, a Processor, an Operator and a Sink, and lay them out as shown in the picture below. The BasicFR will act as our table, but will not have any logic applied to it, and will not function in any way. It is just there to give the Operator somewhere to travel to. A Visual Tool could have also been used, or any of the Fixed Resources.
- Connect the objects as shown in the picture below, making sure to connect the Operator and the BasicFR to the center port of the first Queue in that order.

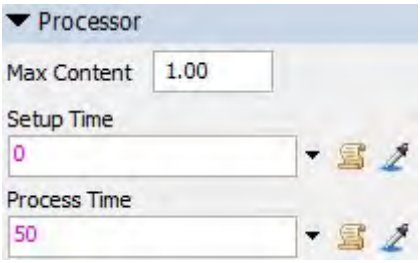


Step 2: Edit the Objects

- Click on the first Processor to open its properties in the Quick Properties.
- Under the the Processor section, change Max Content to 10.



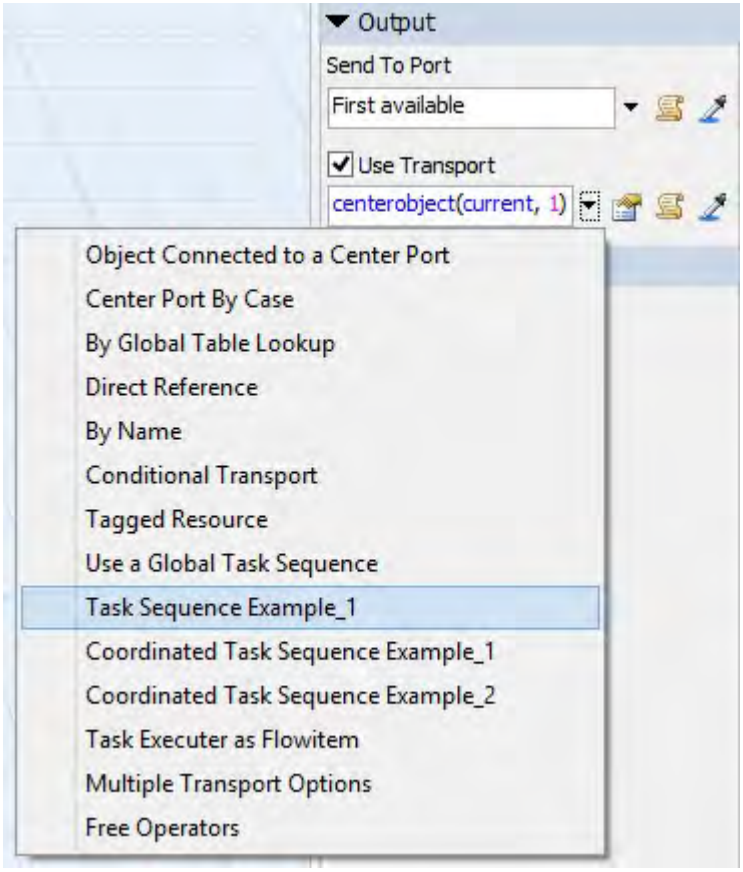
- Click on the first Processor to open its properties in the Quick Properties.
- Under the the Processor section, change Process Time to 50.



Step 3: Write the Task Sequence

To make things easy, you will use the Basic Task Sequence Example to start the task sequence. From there, you will alter and add to it to fit your needs.

- Click the first Queue to open its properties in the Quick Properties.
- Under the the Flow section, check the Use Transport box, and select Task Sequence Example_1 from the list. This task sequence example by default provides the same functionality as referencing an operator. The Operator travels to the current object, loads the item, travels to the downstream object, and unloads the item. You will alter this just a little bit.



- Click the Code Edit button to the right of the drop-down list to open the code editor. In this model, the Operator should do this task, and nothing else. So, you will remove the Break task by deleting all of the code on line 21. See the image below.

```
Object current = ownerobject(c);
Object item = param(1);
int port = param(2);
Object destination = param(3);
double priority = param(4);
int preempt = param(5);

/**Task Sequence Example 1*/
/**Creates a standard task sequence manually.*/
/**If this "Request Transport From" field returns a 0 rather than a valid pointer
to either a dispatcher or taskexecuter, then no call is made, and it is assumed
that the user will dispatch their own tasksequence.

This example shows the code that is required to create the exact same tasksequence
that is normally created automatically and dispatched to the object referenced by this field.*/

treenode dispatcher = current.centerObjects[1]; // the dispatcher or task executer

treenode ts = createemptytasksequence(dispatcher,priority,preempt);

inserttask(ts,TASKTYPE_TRAVEL,current,NULL);
inserttask(ts,TASKTYPE_LOAD,item,current,port);
inserttask(ts,TASKTYPE_BREAK,NULL,NULL);
inserttask(ts,TASKTYPE_TRAVEL, destination,NULL);
inserttask(ts,TASKTYPE_UNLOAD,item, destination,opipno(current,port));

dispatchtasksequence(ts);
// return a 0 so this object will know that you made your own tasksequence and it doesn't need
//to make the standard tasksequence automatically
return 0;
```

After the Operator loads the item, we want him to Travel to the BasicFR, and Delay for 10 seconds before traveling to the downstream Processor.

- Add the following line of code after the TASKTYPE_BREAK:
  inserttask(ts, TASKTYPE_TRAVEL, current.centerObjects[2], NULL);

```
Object current = ownerobject(c);
Object item = param(1);
int port = param(2);
Object destination = param(3);
double priority = param(4);
int preempt = param(5);

/**Task Sequence Example 1*/
/**Creates a standard task sequence manually.*/
/**If this "Request Transport From" field returns a 0 rather than a valid pointer
to either a dispatcher or taskexecuter, then no call is made, and it is assumed
that the user will dispatch their own tasksequence.

This example shows the code that is required to create the exact same tasksequence
that is normally created automatically and dispatched to the object referenced by this field.*/

treenode dispatcher = current.centerObjects[1]; // the dispatcher or task executer

treenode ts = createemptytasksequence(dispatcher,priority,preempt);

inserttask(ts,TASKTYPE_TRAVEL,current,NULL);
inserttask(ts,TASKTYPE_LOAD,item,current,port);
inserttask(ts,TASKTYPE_BREAK,NULL,NULL);
inserttask(ts, TASKTYPE_TRAVEL, current.centerObjects[2], NULL);
inserttask(ts,TASKTYPE_TRAVEL, destination,NULL);
inserttask(ts,TASKTYPE_UNLOAD,item, destination,opipno(current,port));

dispatchtasksequence(ts);
// return a 0 so this object will know that you made your own tasksequence and it doesn't need
//to make the standard tasksequence automatically
return 0;
```

- Go to the next line and add:
  inserttask(ts, TASKTYPE_DELAY, NULL, NULL, 10, STATE_BUSY);
  Click the OK button to close the code window.

```
Object current = ownerobject(c);
Object item = param(1);
int port = param(2);
Object destination = param(3);
double priority = param(4);
int preempt = param(5);

/**Task Sequence Example 1*/
/**Creates a standard task sequence manually.*/
/**If this "Request Transport From" field returns a 0 rather than a valid pointer
to either a dispatcher or taskexecuter, then no call is made, and it is assumed
that the user will dispatch their own tasksequence.

This example shows the code that is required to create the exact same tasksequence
that is normally created automatically and dispatched to the object referenced by this field.*/

treenode dispatcher = current.centerObjects[1]; // the dispatcher or task executer

treenode ts = createemptytasksequence(dispatcher,priority,preempt);

inserttask(ts,TASKTYPE_TRAVEL,current,NULL);
inserttask(ts,TASKTYPE_LOAD,item,current,port);
inserttask(ts,TASKTYPE_BREAK,NULL,NULL);
inserttask(ts, TASKTYPE_TRAVEL, current.centerObjects[2], NULL);
inserttask(ts, TASKTYPE_DELAY, NULL, NULL, 10, STATE_BUSY);
inserttask(ts,TASKTYPE_TRAVEL, destination,NULL);
inserttask(ts,TASKTYPE_UNLOAD,item, destination,opipno(current,port));

dispatchtasksequence(ts);
// return a 0 so this object will know that you made your own tasksequence and it doesn't need
//to make the standard tasksequence automatically
return 0;
```

- Click the OK button on the Properties window to close it.

Step 4: Reset and Run the Model

- Reset and Run the model. The Operator should Travel to the Queue, Load the item, Travel to the BasicFR, Delay for 10 seconds, Travel to the Processor, and Unload the item.
- Save the model. The next tutorial will build off of what you have done here.

Task Sequence 2 Tutorial Introduction

In this tutorial, you will build off the model you completed in Task Sequence Tutorial 1. The Operator will now stay at the Processor to process the item before he starts the next task sequence. This tutorial assumes a solid knowledge of basic interaction with the software, and will ask you to write several lines of code.

If at any time you encounter difficulties while building this model, a fully functional tutorial model can be found at http://www.flexsim.com/tutorials

For more on Tasktypes and their parameters, see the Task Types page. Click here for the Step-By-Step Tutorial.

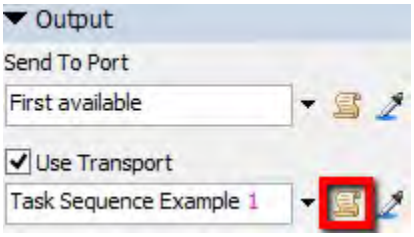
Task Sequence 2 Tutorial Step-By-Step Model Construction

Step 1: Load the Model

If you have not already done so, load the model from Task Sequence Tutorial 1.

Step 2: Add the Utilize Task

- Click on the first Queue to open its properties in the Quick Properties.
- Under the the Flow section, click the Code Edit button to the right of the Use Transport picklist to open the code editor.



- Add a new line after the last inserttask command and type:
  inserttask(ts, TASKTYPE_UTILIZE, item, current.outObjects[1], STATE_UTILIZE);

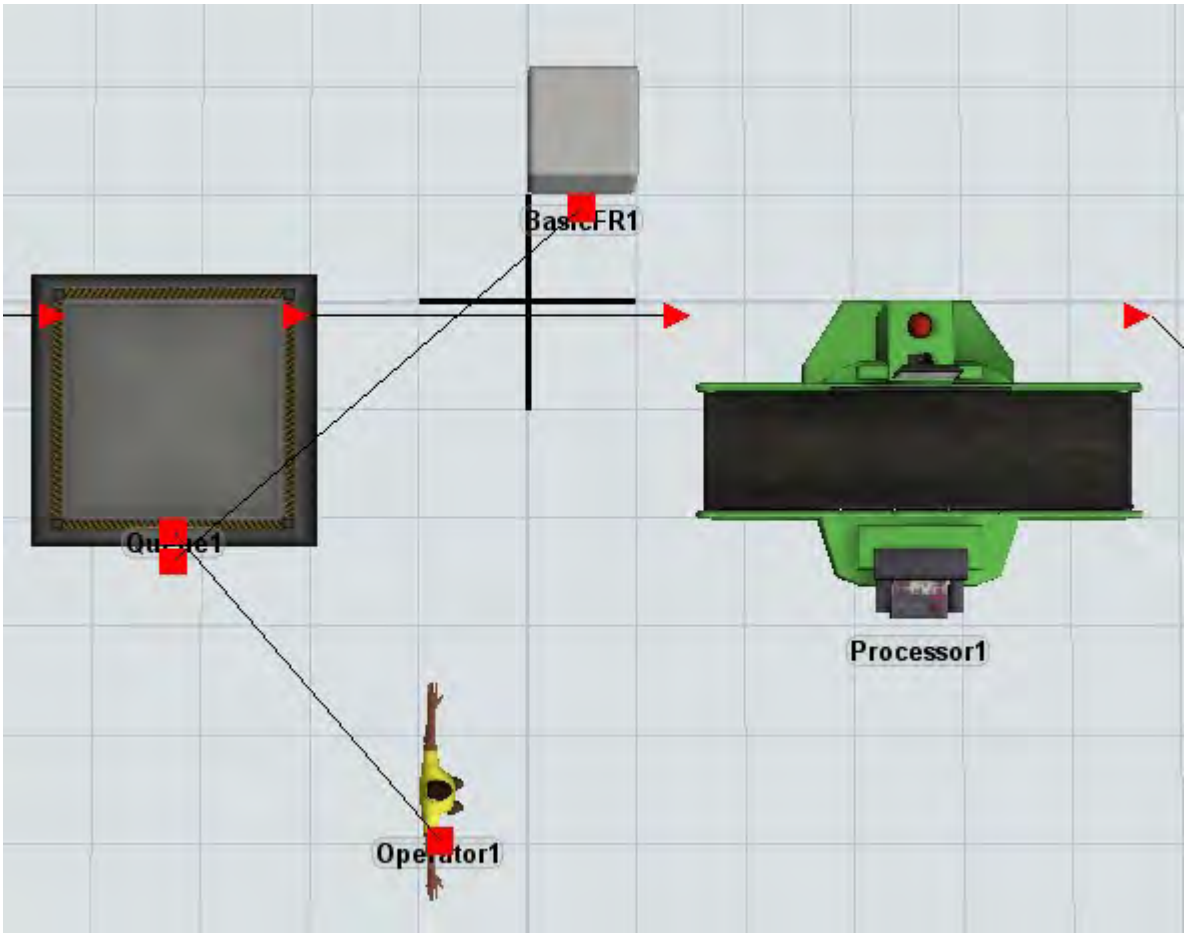
```
Object current = ownerobject(c);
Object item = param(1);
int port = param(2);
Object destination = param(3);
double priority = param(4);
int preempt = param(5);

/**Task Sequence Example 1*/
/**Creates a standard task sequence manually.*/
/**If this "Request Transport From" field returns a 0 rather than a valid pointer
to either a dispatcher or taskexecuter, then no call is made, and it is assumed
that the user will dispatch their own tasksequence.

This example shows the code that is required to create the exact same tasksequence
that is normally created automatically and dispatched to the object referenced by this field.*/

treenode dispatcher = current.centerObjects[1]; // the dispatcher or task executer

treenode ts = createemptytasksequence(dispatcher,priority,preempt);

inserttask(ts,TASKTYPE_TRAVEL,current,NULL);
inserttask(ts,TASKTYPE_LOAD,item,current,port);
inserttask(ts,TASKTYPE_BREAK,NULL,NULL);
inserttask(ts, TASKTYPE_TRAVEL, current.centerObjects[2], NULL);
inserttask(ts, TASKTYPE_DELAY, NULL, NULL, 10, STATE_BUSY);
inserttask(ts,TASKTYPE_TRAVEL, destination,NULL);
inserttask(ts,TASKTYPE_UNLOAD,item, destination,opipno(current,port));
inserttask(ts, TASKTYPE_UTILIZE, item, current.outObjects[1], STATE_UTILIZE);

dispatchtasksequence(ts);
// return a 0 so this object will know that you made your own tasksequence and it doesn't need
//to make the standard tasksequence automatically
return 0;
```

- Click OK to close the Code Edit window.

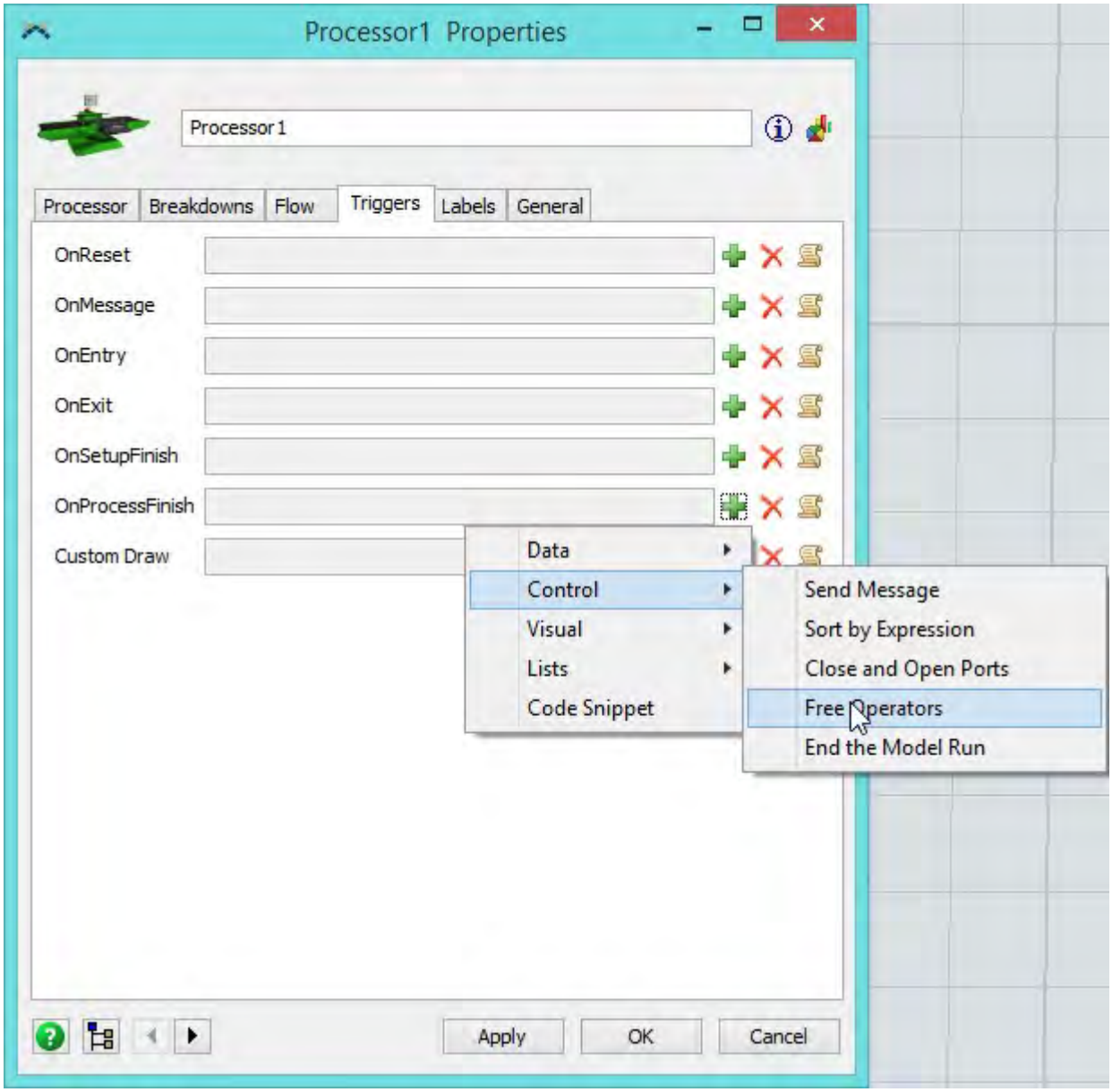
If you Run the model now, you will notice that the operator stays at the Processor forever after the item has been unloaded and processed. This is because nothing is freeing the Operator, and the Operator will stay utilized until it is freed by an object. The best place to do this is in the OnProcessFinish of the Processor that is utilizing the Operator.

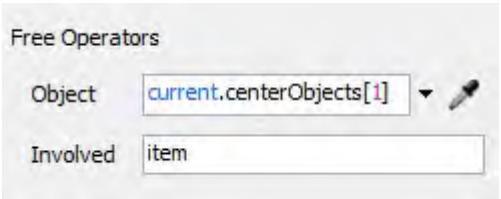
Step 3: Edit the Processor to Free the Operator

- Connect the Operator to the first Processor with a center port connection.



- Double-click the first Processor to open its Properties window, then click the Triggers tab. In the OnProcessFinish trigger list, select Free Operators. The default trigger parameters will work for this model. The Involved is the first parameter in the TASKTYPE_UTILIZE command we used in the previous step. In order for the Operator to be freed, the involved object must match, in this case, item.





- Click the OK button on the Properties window to close it.

Step 4: Reset and Run the Model

- Reset and Run the model. The Operator should Travel to the Queue, Load the item, Travel to the BasicFR, Delay for 10 seconds, Travel to the Processor, Unload the item, and stay at the Processor for the Process Time.
- Save the model. The next tutorial will build off of what you have done here.

Task Sequence 3 Tutorial Introduction

In this tutorial, you will build off the model you completed in Task Sequence Tutorial 2. The Operator will now pick the item up from the Processor, and carry it to the second Queue. This tutorial assumes a solid knowledge of basic interaction with the software, and will ask you to write several lines of code. If at any time you encounter difficulties while building this model, a fully functional tutorial model can be found at http://www.flexsim.com/tutorials

For more on Tasktypes and their parameters, see the Task Types page. Click here for the Step-By-Step Tutorial.

Task Sequence 3 Tutorial Step-By-Step Model Construction

Step 1: Load the Model

If you have not already done so, load the model from Task Sequence Tutorial 2.

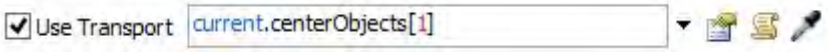
Step 2: Delete the OnProcessFinish Trigger

Since you will be adding to the task sequence, a change needs to be made in how the Operator is released from the Utilize task. The freeoperator() command needs to be moved to the Request Transport From field in order for the model to work correctly. If you need more information, an in-depth explanation will be provided at the end of the tutorial.

- Double-click on the first Processor to open its Properties window, and click the Triggers tab.
- In the OnProcessFinish trigger, click the button, and then click the button to remove the function on this trigger.
- Don't close the Properties window yet.

Step 3: Write the Flow Logic

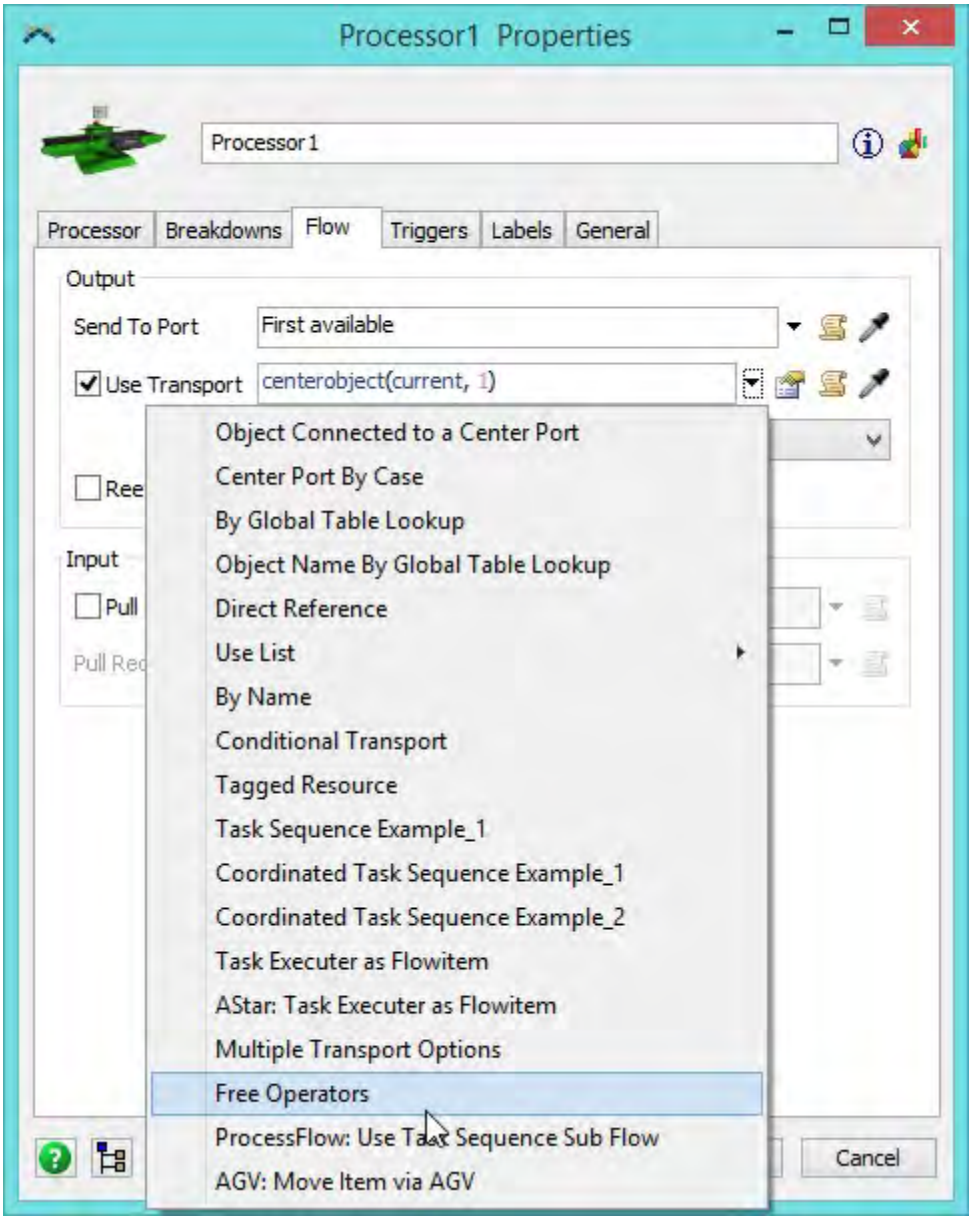
Now you will need to free the operator in the Request Transport From (

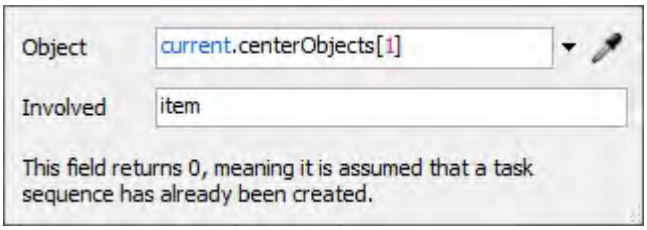


) field. Also, since the next set of tasks you will add will override the Processor's flow logic, specifically the Request Transport From logic, you need to tell the processor that it doesn't need to create a task sequence.

Note: Whenever you write a task sequence that controls the output of an object somewhere other than the Request Transport From on the object the task sequence is affecting, you MUST return 0 in that object's Request Transport From logic, otherwise there will be serious problems. In this example, you are writing the task sequence on the Queue, but the next tasks you will add affect the Processors natural transport logic, so even though the task sequence is on the Queue, you will need to return 0 on the Processor's Request Transport From logic. You would also need to do this if you were writing a task sequence in a trigger field instead of in the Request Transport From.

- Double-click on the first Processor to open its Properties window, and click the Flow tab.
- Check the Use Transport box, then choose Free Operators from the drop down list. The default parameters should work fine for this model. Notice that the description points out that this option returns 0 for you, so you don't have to worry about writing this anywhere.

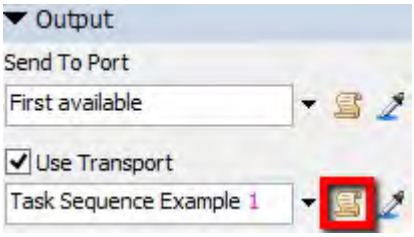




- Click the OK button to close the Properties window.

Step 3: Write the rest of the Task Sequence

- Click on the first Queue to open its properties in the Quick Properties.
- Under the the Flow section, click the Code Edit button to the right of the Use Transport picklist to open the code editor.



- You will need to create a local variable so that you can more easily reference the second Queue in you task sequence. On line 17, type the following:
  Object downQueue = current.outObjects[1].outObjects[1];

```
1 Object current = ownerobject(c);
2 Object item = param(1);
3 int port = param(2);
4 Object destination = param(3);
5 double priority = param(4);
6 int preempt = param(5);
7
8 /**Task Sequence Example 1*/
9 /**Creates a standard task sequence manually.*/
10 /**If this "Request Transport From" field returns a 0 rather than a valid pointer
11 to either a dispatcher or taskexecuter, then no call is made, and it is assumed
12 that the user will dispatch their own tasksequence.
13
14 This example shows the code that is required to create the exact same tasksequence
15 that is normally created automatically and dispatched to the object referenced by this field.*/
16
17 treenode dispatcher = current.centerObjects[1]; // the dispatcher or task executer
18
19 Object downQueue = current.outObjects[1].outObjects[1];
20 treenode ts = createemptytasksequence(dispatcher,priority,preempt);
21
22 inserttask(ts,TASKTYPE_TRAVEL,current,NULL);
23 inserttask(ts,TASKTYPE_LOAD,item,current,port);
24 inserttask(ts,TASKTYPE_BREAK,NULL,NULL);
25 inserttask(ts, TASKTYPE_TRAVEL, current.centerObjects[2], NULL);
26 inserttask(ts, TASKTYPE_DELAY, NULL, NULL, 10, STATE_BUSY);
27 inserttask(ts,TASKTYPE_TRAVEL, destination,NULL);
28 inserttask(ts,TASKTYPE_UNLOAD,item, destination,opipno(current,port));
29 inserttask(ts, TASKTYPE_UTILIZE, item, current.outObjects[1], STATE_UTILIZE);
30
31 dispatchtasksequence(ts);
32 // return a 0 so this object will know that you made your own tasksequence and it doesn't need
33 //to make the standard tasksequence automatically
34 return 0;
```

- Add the following to the end of the task sequence:
  inserttask(ts, TASKTYPE_FRLOAD, item, current.outObjects[1]); inserttask(ts, TASKTYPE_TRAVEL, downQueue, NULL); inserttask(ts, TASKTYPE_FRUNLOAD, item, downQueue, 1);

```
Object current = ownerobject(c);
Object item = param(1);
int port = param(2);
Object destination = param(3);
double priority = param(4);
int preempt = param(5);

/**Task Sequence Example 1*/
/**Creates a standard task sequence manually.*/
/**If this "Request Transport From" field returns a 0 rather than a valid pointer
to either a dispatcher or taskexecuter, then no call is made, and it is assumed
that the user will dispatch their own tasksequence.

This example shows the code that is required to create the exact same tasksequence
that is normally created automatically and dispatched to the object referenced by this field.*/

treenode dispatcher = current.centerObjects[1]; // the dispatcher or task executer

Object downQueue = current.outObjects[1].outObjects[1];
treenode ts = createemptytasksequence(dispatcher,priority,preempt);

inserttask(ts,TASKTYPE_TRAVEL,current,NULL);
inserttask(ts,TASKTYPE_LOAD,item,current,port);
inserttask(ts,TASKTYPE_BREAK,NULL,NULL);
inserttask(ts, TASKTYPE_TRAVEL, current.centerObjects[2], NULL);
inserttask(ts, TASKTYPE_DELAY, NULL, NULL, 10, STATE_BUSY);
inserttask(ts,TASKTYPE_TRAVEL, destination,NULL);
inserttask(ts,TASKTYPE_UNLOAD,item, destination,opipno(current,port));
inserttask(ts, TASKTYPE_UTILIZE, item, current.outObjects[1], STATE_UTILIZE);
inserttask(ts, TASKTYPE_FRLOAD, item, current.outObjects[1]);
inserttask(ts, TASKTYPE_TRAVEL, downQueue, NULL);
inserttask(ts, TASKTYPE_FRUNLOAD, item, downQueue, 1);

dispatchtasksequence(ts);
// return a 0 so this object will know that you made your own tasksequence and it doesn't need
//to make the standard tasksequence automatically
return 0;
```

- Click the OK button on the Code Window and theProperties window to close them.

Note: The reason that the freeoperators() command needed to be changed from the OnProcessFinish is due to the fact that a written task sequence has the ability to override the internal logic of objects. If the downstream queue isn't available when the Processor finishes its process time, then the Processor will release the Operator to load/unload the part into the Queue before it's actually available, which can cause the Queue to be over-filled, and can cause some other problems. So by moving the freeoperators() to the Request Transport From field, the operator will only be freed to continue with the load/unload only when the downstream queue is ready to receive that part. The perfect solution to this problem would be to have two utilize tasks, freeing the Operator on both the OnProcessFinish, and the Request Transport From. This way, the operator would be free after the Processor finishes, in case you need him to break, but he won't move the item until the downstream object is available. See the Fixed Resource page for more information.

Step 4: Reset and Run the Model

- Reset and Run the model. The Operator should Travel to the Queue, Load the item, Travel to the BasicFR, Delay for 10 seconds, Travel to the Processor, Unload the item, stay at the Processor for the Process Time, Load the item, Travel to the next Queue, and Unload the item.
- Save the model.

SQL Tutorial Introduction

This tutorial will help you understand how to connect FlexSim to SQL databases.

Note: This is not a beginner tutorial, it is assumed you know basic FlexSim and are familiar with SQL databases. I will be using phpMyAdmin for SQL during the tutorials.

What You Will Learn

- Send information to a SQL database from FlexSim
- Read data from a SQL database for dynamic use within FlexSim

Approximate Time to Complete this Lesson

This lesson should take about 15-20 minutes to complete.

Model Overview

In this model we will get data from a table in a SQL database to determine process times. We will also send information to another table in the same database about flowitem staytimes. Click here for the Step-By-Step Tutorial.

SQL Tutorial Step-By-Step Model Construction

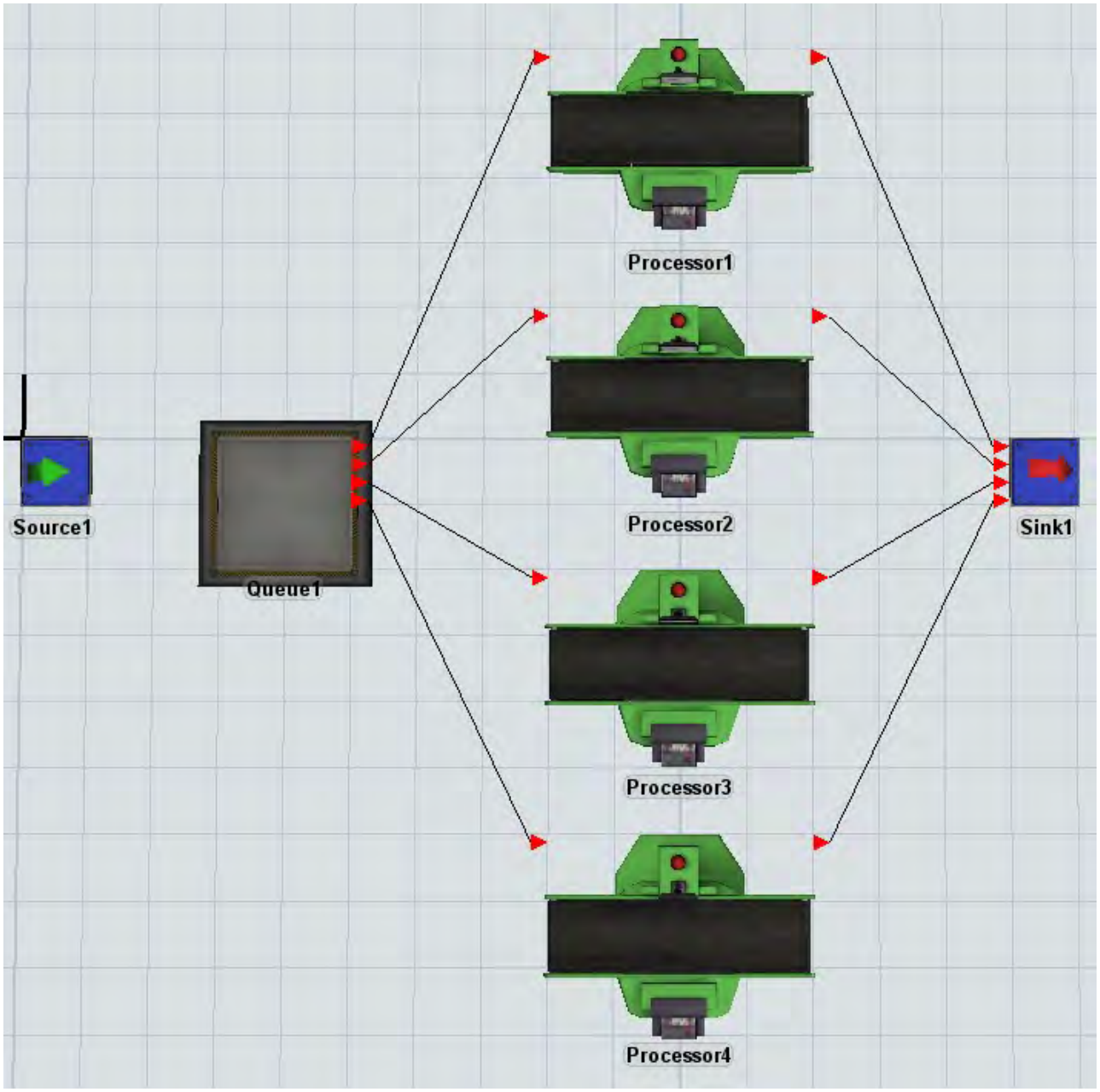
Building SQL Model

 button on the toolbar. Click OK on the Model Units window, we will Begin a new model by clicking the use the default units for our model.

If at any time you encounter difficulties while building this model, a fully functional tutorial model can be found at http://www.flexsim.com/tutorials

Step 1: Create the Objects

- Drag objects from your Library Icon Grid onto the 3D View to create the model shown below.



Connect all of the objects as shown:

- Connect *Source1* to *Queue1*.
- Connect *Queue1* to *Processor1*, *Processor2*, *Processor3* and *Processor4*.
- Connect *Processor1*, *Processor2*, *Processor3* and *Processor4* to *Sink1*.

Step 2: Setup the SQL Database

We will create an SQL database that we can both read from and write to from our FlexSim model. First, we'll create a table that stores the process times for each processor based on item type.

- Create an SQL database called *"flexsimdata"*
- Create a table within *"flexsimdata"* called *"processtimes"* with 4 columns named *"Proc1"*, *"Proc2"*, *"Proc3"*, and *"Proc4"*. Each of these columns should contain FLOAT number data.
- Fill the columns as shown in the following figure:

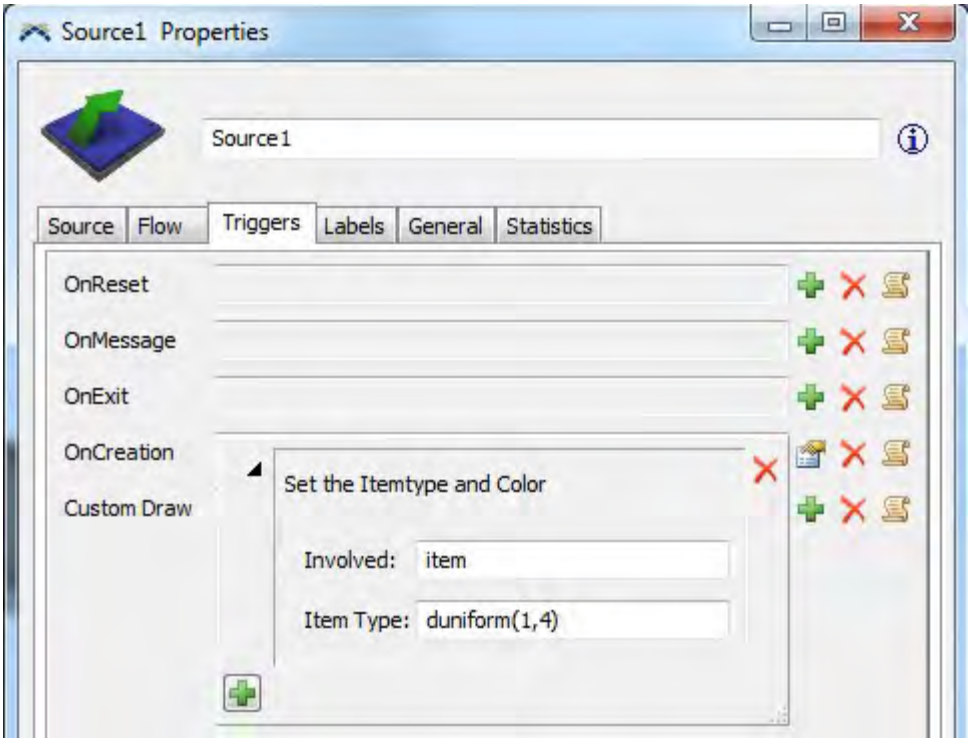
Proc1	Proc2	Proc3	Proc4
3.2	3.4	3.6	3.8
4.2	4.4	4.6	4.8
5.2	5.4	5.6	5.8
6.2	6.4	6.6	6.8

Step 3: Setup the Source

We will have the *Source* create 4 different item types.

- Change the *Source's* Inter-Arrivaltime to *exponential(0, 3, 0)*

- Go to the Triggers tab.
- Click the add button for the OnCreation trigger.
- Select the Set Item Type and Color picklist option.
- Change Type to *duniform(1, 4)*



- Click OK to apply and close the Properties window.

Step 4: Setup the Processors

Each of the 4 processor's will access the SQL database we just created in order to set their processing time.

- Open one of the *Processor's* properties window.
- Click the code edit button next to *Process Time*.
- Enter the following code:

```
1 /*Custom Code*/
2 treenode current = ownerobject(c);
3 treenode item = parnode(1);
4
5 dbopen("flexsimdata", "processtimes", 1);
6
7 double proctime = dbgettablenum(getitemtype(item), 1);
8
9 dbclose();
10
11 return proctime;
```

- The dbopen() command will access your database and open it up for reading or writing. The first parameter in the name of the database as named in your Windows ODBC Data Source Administrator (not necessarily the name shown in phpMyAdmin). The second parameter in this case is the table you want to read from or write to. The third parameter toggles between table mode (1) and SQL mode (0). We use 1 in this case because we aren't using direct SQL queries, we are declaring a table to work with in the database.
- We use a special command called dbgettablenum() to get information from a table in a database. In our case, we want the row to match the item type, and we want each column to represent each processor. You will change the 1 to 2, 3, and 4 for the other processors.
- You need to use dbclose() to close the database so other databases can be accessed later on.

- Click OK to apply and close the Properties window.

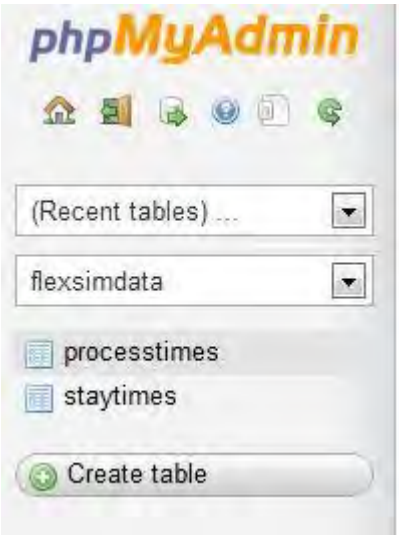
Write the same code on the other three processors, replacing each consecutive processor's second parameter in dbgettablenum() to 2, 3, and 4, respectively.

Reset and Run the model. If you look at the staytime statistic, you should notice that they correspond to the values in the *flexsimdata* database's *processtimes* table.

Step 5: Add Staytimes Table to Database

We will now add another table to our SQL database to store the stay times of all the items going through our model.

- Create a table within *"flexsimdata"* called *"staytimes"* with 2 columns named *"ID"* and *"Staytime"*. The first column should contain integer data and should autoincrement, and the second column should contain FLOAT number data.



- Create one initial row as shown in the following figure with *ID = 1* and *Staytime = 0*.

Step 6: Write to the SQL Database

Each time an item enters the Sink, we will record that item's time in the system (staytime) and write that value to the SQL database.

- Open the *Sink's* properties window.
- Go to the Triggers tab.
- Click the code edit button button for the OnEntry trigger.
- Enter the following code:

```
 string staytime = numtostring(time() - get(stats_creationtime(item)),2, 3); string query = concat("INSERT INTO `flexsimdata`.`staytimes` (`Staytime`)VALUES ('", staytime, "');"); string altquery = concat("UPDATE staytimes SET Staytime =", staytime, " WHERE ID =", numtostring(getinput(current))); dbopen("flexsimdata", "SELECT * FROM staytimes", 0); int rows = dbgetnumrows(); if (getinput(current) < rows)
{     dbsqlquery(altquery);
} else {
     dbsqlquery(query); }
dbclose();
```

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

string staytime = numtostring(time() - get(stats_creationtime(item)),2, 3);

string query = concat("INSERT INTO  `flexsimdata`.`staytimes` (`Staytime`)VALUES ('", staytime, "');");
string altquery = concat("UPDATE staytimes SET Staytime =", staytime, " WHERE ID =", numtostring(getinput(current)));

dbopen("flexsimdata", "SELECT * FROM staytimes", 0);
int rows = dbgetnumrows();
if (getinput(current) < rows)
{
    dbsqlquery(altquery);
}
else
{
    dbsqlquery(query);
}
dbclose();

```

Once we get the stay time into a variable called *staytime*, we make a query for adding data into the Staytime column of the *staytimes* table. The alternate query is made as well in case the table already contains data and needs to be over-written. Therefore, we have the if statement requiring us to use the alternate query for as long as there are previously existing rows.

- Click OK to apply and close the Properties window.

Reset and Run the model. You should notice the *staytimes* table of your database filling up with the information gathered at the *Sink*, similar to the following figure.

+ Options

	ID	Staytime
Edit Copy Delete	1	4.8
Edit Copy Delete	2	6.6
Edit Copy Delete	3	6.8
Edit Copy Delete	4	6.4
Edit Copy Delete	5	3.6
Edit Copy Delete	6	4.8
Edit Copy Delete	7	3.2
Edit Copy Delete	8	4.6
Edit Copy Delete	9	6.4
Edit Copy Delete	10	6.8

This completes the SQL tutorial. Congratulations!

Fluid Objects Tutorial 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.

Fluid Model Overview

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.

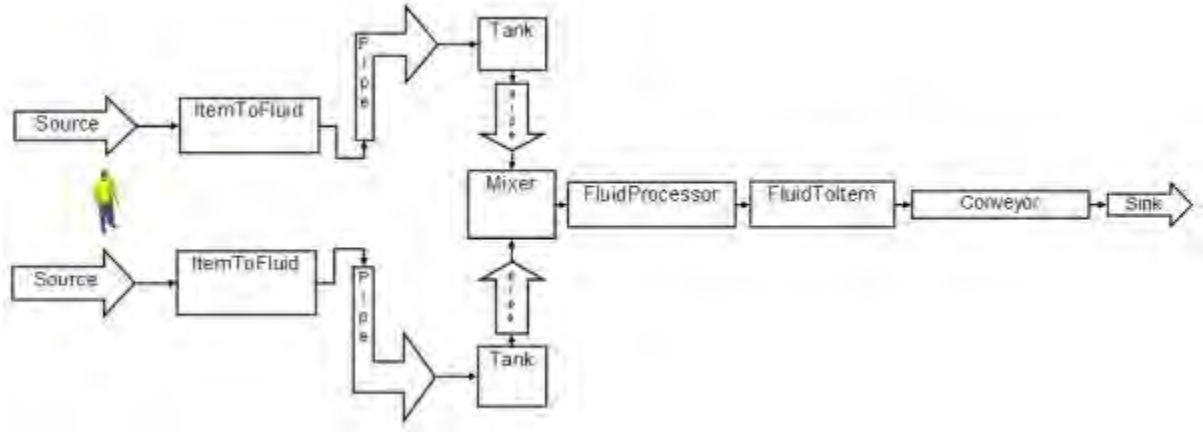


Figure 4-1 Fluid Model diagram

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

New Concepts

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. Click here for the Step-By-Step Tutorial.

Fluid Objects Tutorial Step-By-Step Model Construction

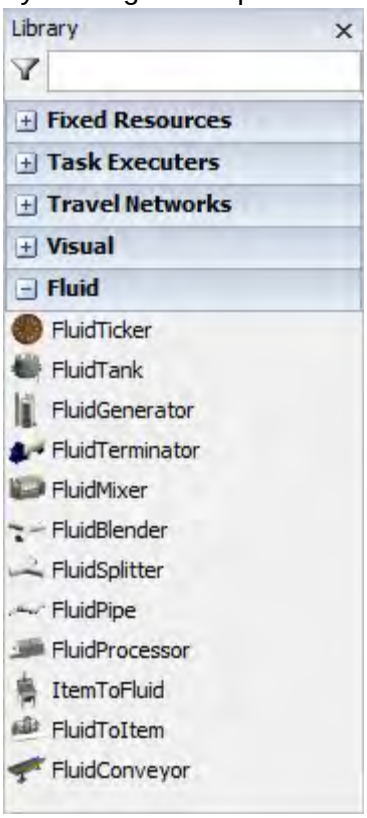
Building the Fluid Model

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

If at any time you encounter difficulties while building this model, a fully functional tutorial model can be found at http://www.flexsim.com/tutorials

Step 1: Model Layout and Connections

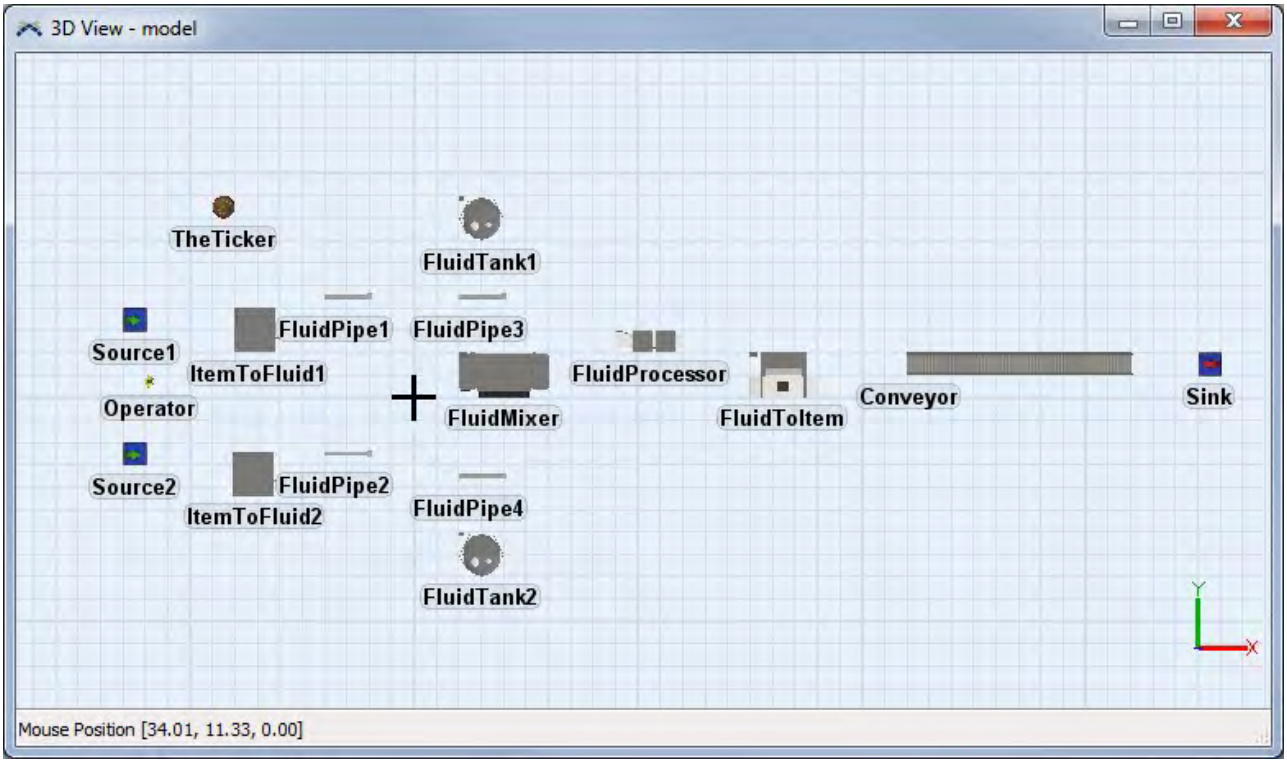
To begin this lesson, drag out the following objects from the library. These objects can be found in the library by clicking on the plus button next to "Fluid" found near the bottom of the library window:



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.

- 2 Sources named *Source1* and *Source2* (Discrete Objects).
- 1 Operator named *Operator* (Discrete Objects).

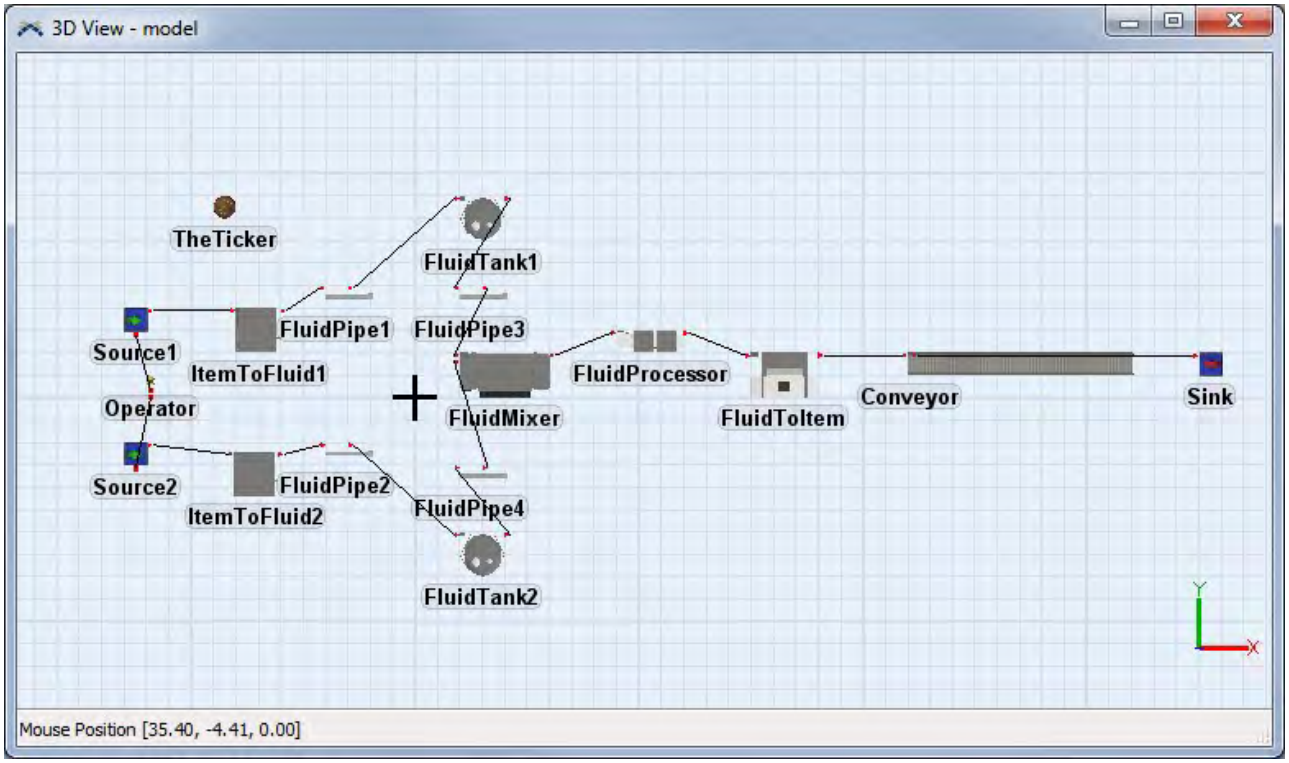
- 2 ItemToFluids named *ItemToFluid1* and *ItemToFluid2* (Fluid Objects).
- 2 FluidPipes named *FluidPipe1* and *FluidPipe2* (Fluid Objects).
- 2 FluidTanks named *FluidTank1* and *FluidTank1* (Fluid Objects).
- 2 more FluidPipes named *FluidPipe3* and *FluidPipe4* (Fluid Objects).
- 1 FluidMixer named *FluidMixer* (Fluid Objects).
- 1 FluidProcessor named *FluidProcessor* (Fluid Objects).
- 1 FluidToItem named *FluidToItem* (Fluid Objects).
- 1 Conveyor named *Conveyor* (Discrete Objects).
- 1 Sink named *Sink* (Discrete Objects).
- Arrange the objects as shown below.



Once the objects have all been 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 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 o from *Source1* to *ItemToFluid1* o from *ItemToFluid1* to *FluidPipe1* o from *FluidPipe1* to *FluidTank1* o from *FluidTank1* to *FluidPipe3* o from *FluidPipe3* to *FluidMixer* o from *FluidMixer* to *FluidProcessor* o from *FluidProcessor* to *FluidToItem* o from *FluidToItem* to *Conveyor* o from *Conveyor* to *Sink* in that order, as shown below.

- Make a parallel line of connections from *Source2* to *FluidMixer*
- Both sources should call for the *Operator* to transport the flowitem to *ItemToFluid*, so a center port connection needs to be made from each *Source1* and *Source2* to the *Operator*.



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.

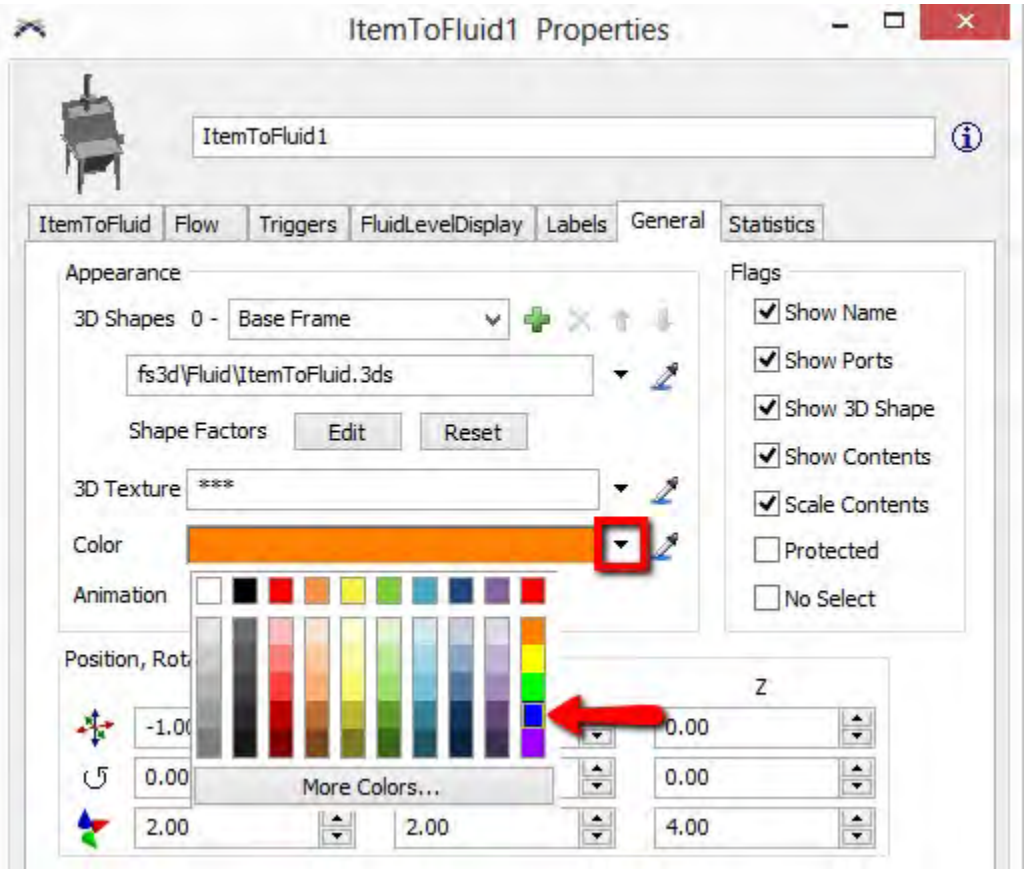
- Click on *Source1* to open its properties in the Quick Properties. Under the Flow section check Use Transport.
- Repeat this step for *Source2*.

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, we will change the color of those four objects on one line to blue and the objects on the other line to red. This will cause pieces of the objects to change to that color as the model is running.

- Double-click on *ItemToFluid1* to open its Properties window, and click the General tab.
- Click the ▾ button located next to the Color field. The color popup will appear. Select a blue color from the the options, then click OK.

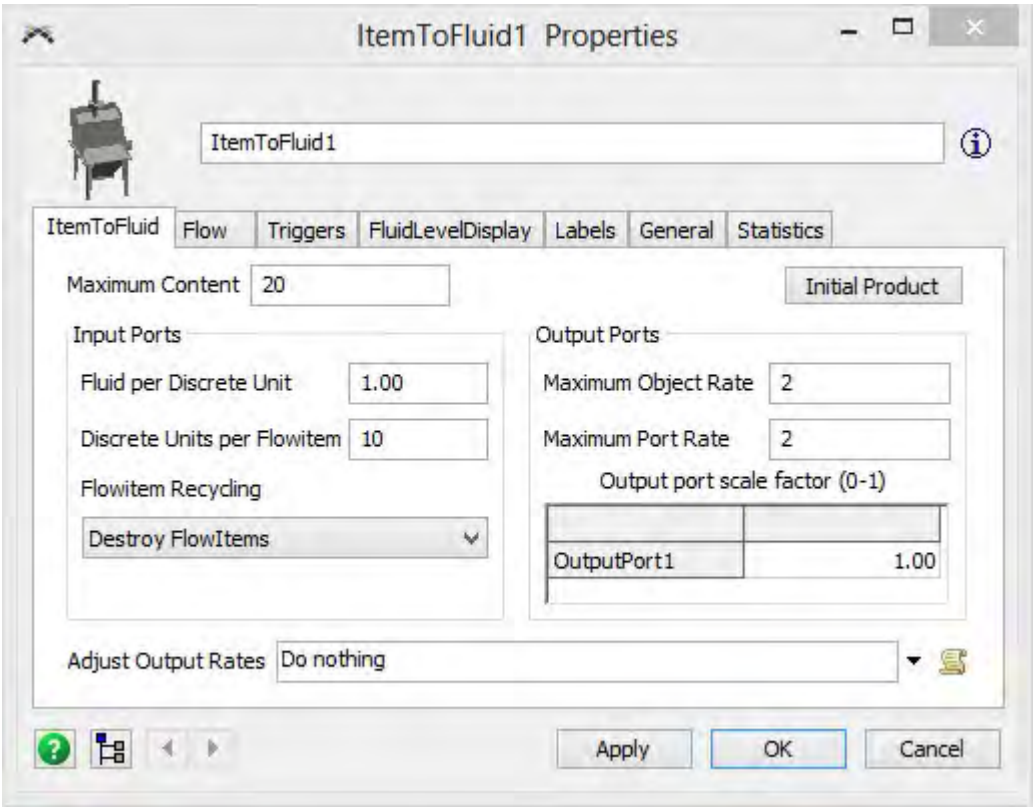
- Click OK to apply the changes and close the Properties window.
- Repeat this step for the rest of the objects mentioned above, coloring the other processing line red instead of blue.



Step 4: Configure the ItemToFluid

Next the ItemToFluid objects have to be configured to create the correct amount of material for each flowitem that comes in.

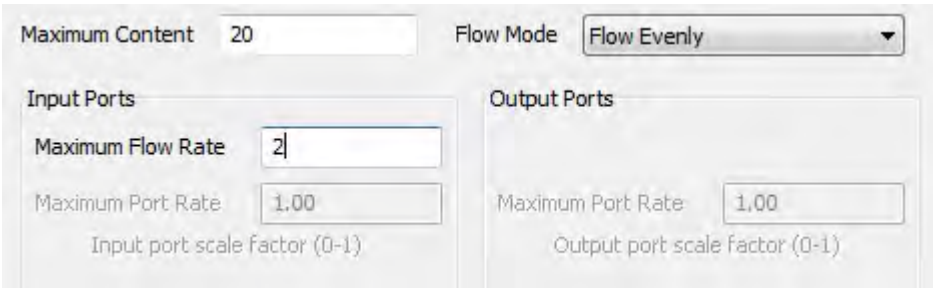
- Double-click *ItemToFluid1* to open its Properties window. On the ItemToFluid tab, change Discrete Units per Flowitem to 10. This tells *ItemToFluid* to create 10 gallons of fluid for each flowitem that enters.
- Change Maximum Object Rate and Maximum Port Rate to 2.
- Change the Maximum Content to 20.
- Click OK to apply the changes and close the Properties window.
- Repeat this step for *ItemToFluid2*.



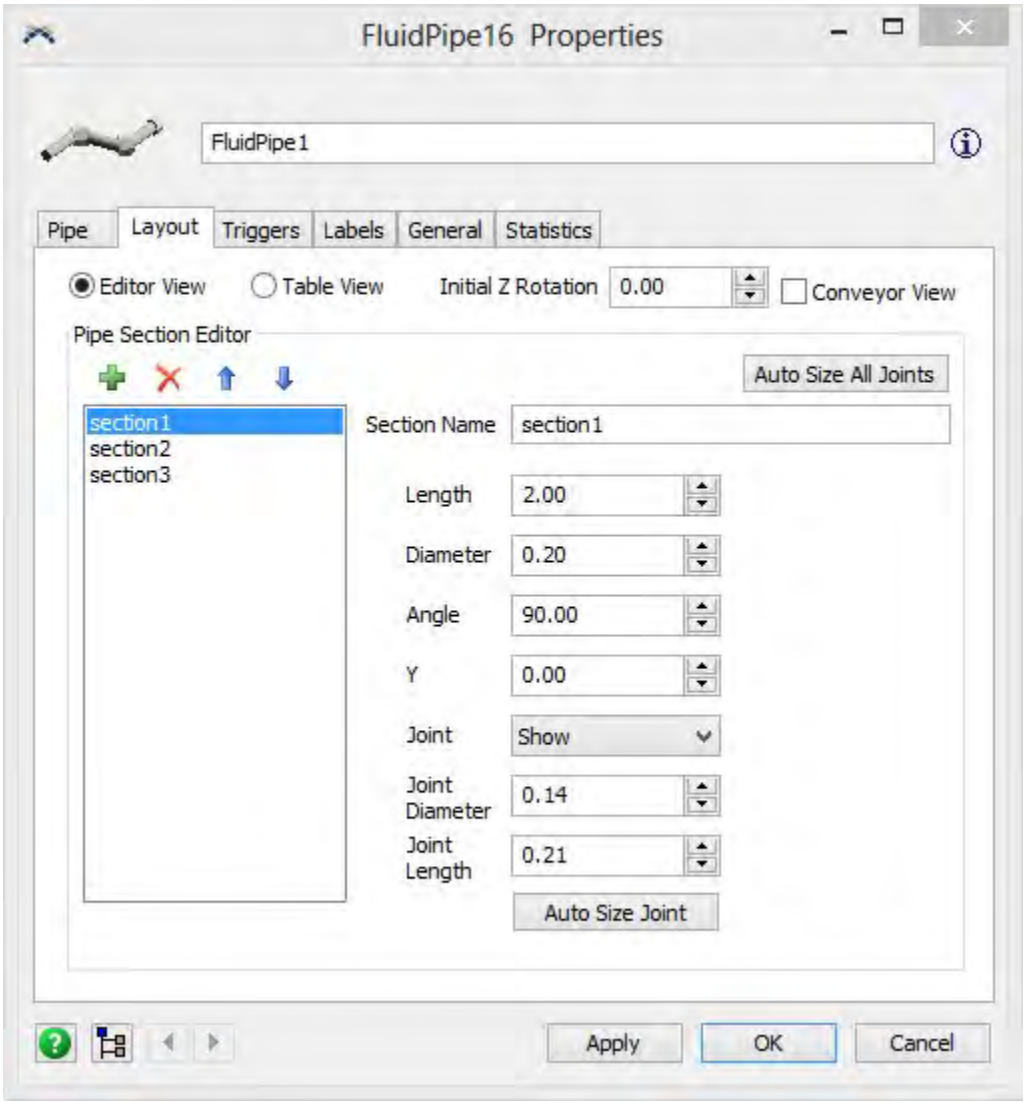
Step 5: Configure the FluidPipes

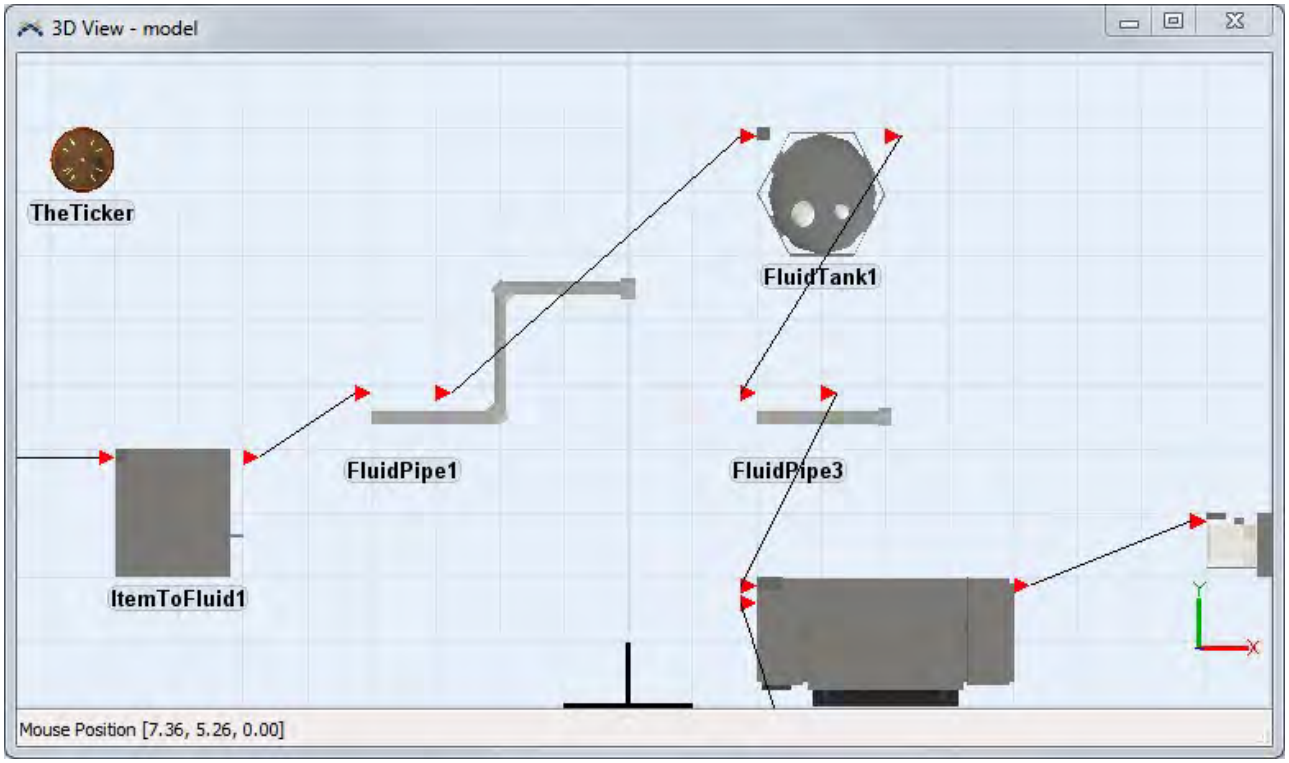
The Pipes that lead away from the ItemToFluids are the next objects that need to be configured. Pipes do not allow the modeler to specify the input and output rates. The output rate is based on the actual rate that material was received.

- Double-click *FluidPipe1* to open its Properties window.
- On the Pipe tab, set Maximum Flow Rate to 2, and set the Maximum Content to 20. This will ensure that material spends time in the Pipe, but not too much time.



- Click the Layout tab. Adjust the FluidPipe, changing and/or adding sections of pipe, so that it appears to start at *ItemToFluid* and end near *FluidTank* (the actual values of your FluidPipe may be different than the ones shown below). Changing the Layout does not affect the behavior of the pipes. More information about this tab can be found here.


FluidPipe16 Properties
FluidPipe1
Pipe
Layout
Triggers
Labels
General
Statistics
Editor View
Table View
Initial Z Rotation
0.00
Conveyor View
Pipe Section Editor
Auto Size All Joints
section1
section2
section3
Section Name
section1
Length
2.00
Diameter
0.20
Angle
90.00
Y
0.00
Joint
Show
Joint Diameter
0.14
Joint Length
0.21
Auto Size Joint
Apply
OK
Cancel

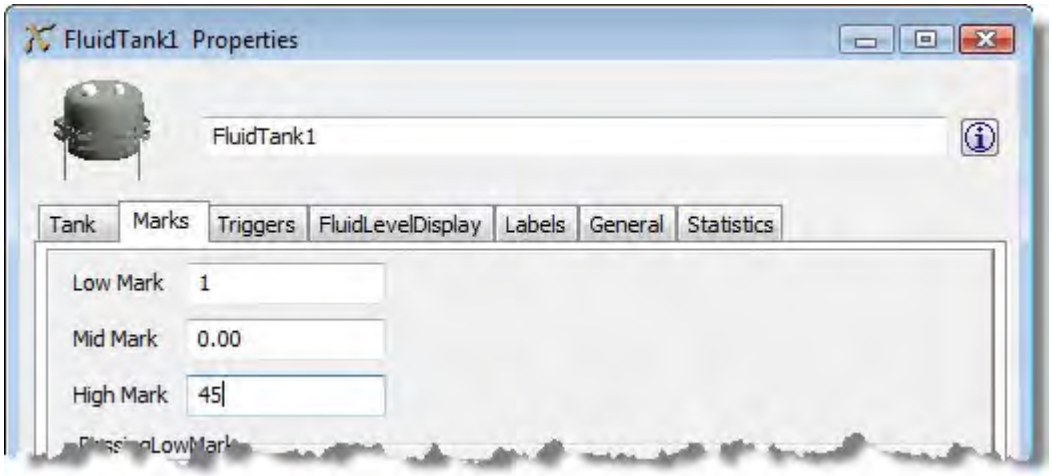


- Click the OK button to apply the changes and close the Properties window. Repeat this step for *FluidPipe2*.

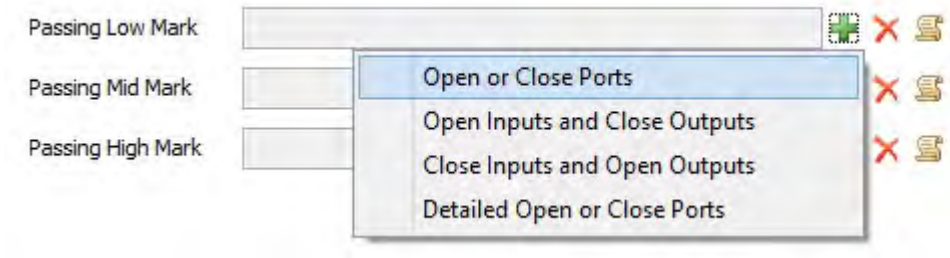
Step 6: Configure the FluidTanks

The ItemToFluids now have a maximum output rate of 2, but the FluidTanks have a maximum input rate 1. If these values are left as they are, the FluidTank's rate will be used during the model run (because it is the lower of two values) and the FluidPipe will not be able to send material downstream as fast as you have told it to. So the rate on the FluidTanks needs to be changed. FluidTanks allow the modeler to set three levels that will cause triggers to fire when the content of the FluidTank reaches them. These values are called marks. They are edited on the Marks tab of the Properties Window. 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.

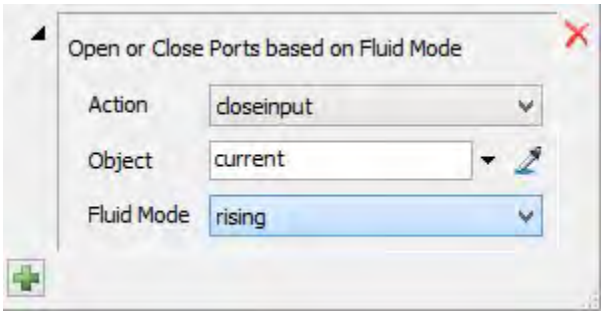
- Double-click *FluidTank1*. On the Tank tab, under Input Ports, change the Maximum Object Rate and the Maximum Port Rate to 2.
- Click the Marks tab. Change the Low Mark to 1, and the High Mark to 45. Leave the Mid Mark at 0. If a mark has the value 0, the trigger for that mark will never fire.



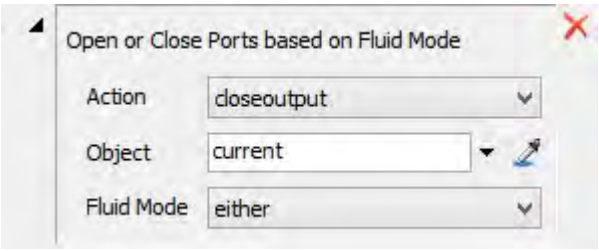
- On the Marks tab, click the + to add a function to the Passing Low Mark trigger. Select the Open or Close Ports option.



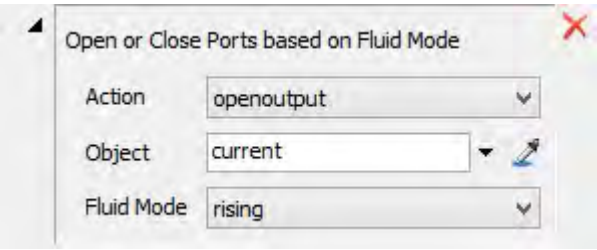
- Click where it says rising. The available options will appear.



- Give the function the following parameters:



- Add the Open or Close Ports function to the Passing High Mark trigger in the same way. Give it the following parameters:



- Click OK to apply the changes and close the Properties window.
- Repeat the same procedure for FluidTank2.

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 adjusted so that they point toward the Mixer. Again, this is completely visual, the size and layout of the Pipes will not affect their behavior. Use the Layout tab to configure the Pipes in your model so that they look good to you. The default maximum content of these Pipes is currently too large for this model.

- Double-click *FluidPipe3* to open its Properties window. On the Pipe tab, change the Maximum Content to 10. This will make sure that the material leaving the Tanks takes just a little time to get to the Mixer.
- Repeat this step for *FluidPipe4*.

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

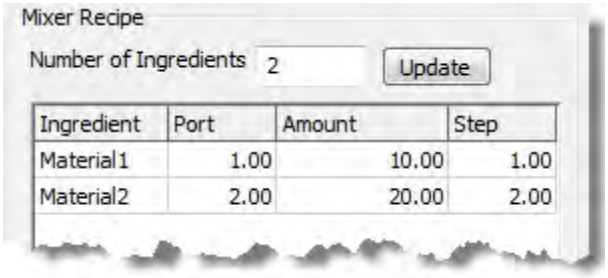
The FluidMixer now needs to be configured to receive the two different materials and combine them into a new material. This is done by changing the two tables found on the Steps tab in the FluidMixer's Properties window. The Mixer Steps table is used to define a series of steps that the Mixer must go through for each batch that it processes. In this model, the Mixer Steps table needs 2 steps. 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.

- Double-click the *FluidMixer* to open its Properties window, and click the Steps tab.
- Under Mixer Steps, change the Number of Steps to 2.
- Click Update. The steps will appear in the table.

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.

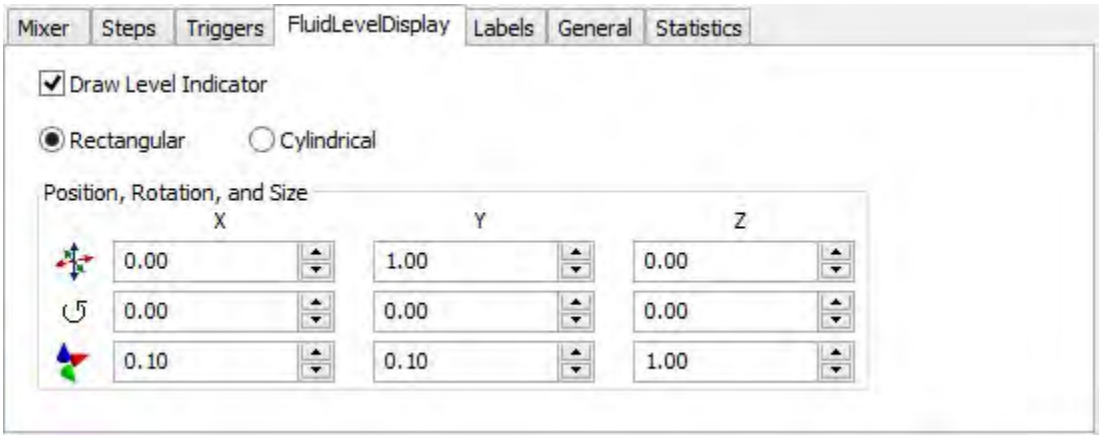
- Set the Delay for Step 1 to 0, and the Delay for Step 2 to 10.

- On the Steps tab, under Mixer Recipe, change the Number of Ingredients to 2.
- Click Update. The ingredients will now appear in the table. Once more, the description can be anything you want because the object does not refer to these names. Call them *Material1* and *Material2*.
- For *Material1*, change the Port to 1, the Amount to 10, and the Step to 1.
- For *Material2*, change the Port to 2, the Amount to 20, and the Step to 2.



The Mixer's level display 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. By default it is hidden behind the Mixer's 3D shape. More information about the fluid level display is found here. You can also manipulate the size and shape of the fluid level display so that it looks like part of the object to get a more realistic look.

- Click the FluidLevelDisplay tab. Change Y to 1.



Step 9: Examine the FluidProcessor

The FluidProcessor's default values will work well for this model. It will receive material from input port 1, process it for a certain amount of time and send it to output port 1. The amount of time it spends processing is based on its Maximum Content and the Maximum Output Rate that the modeler defines in the

FluidProcessor Properties window on the FluidProcessor tab. You can play with these values if you want to see how they affect the model. To make the bar on this object appear in front like on that of the *Mixer*, you will once again need to change the Y location of the level indicator bar to 1, as done in the previous step.

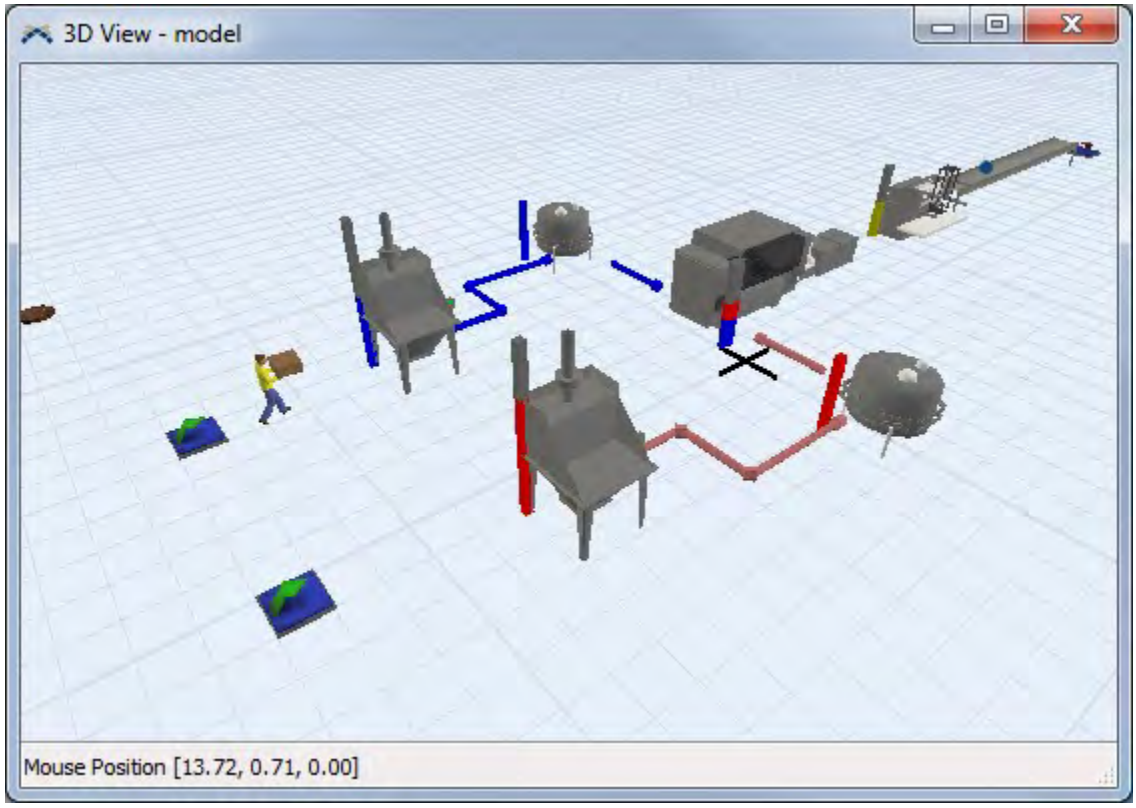
Step 10: Configure the FluidToItem

The FluidToItem near the end of the line will convert the fluid material coming from the FluidProcessor and change it into flowitems. The Fluid per Discrete Unit and Discrete Units per Flowitem 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.

- Double-click the *FluidToItem* to open its Properties window.
- On the FluidToItem tab, under Flowitem Output, change Fluid per Discrete Unit to 10.
- On the FluidToItem tab, change the Maximum Content 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.

- Reset and Save the model.
- Run the model. You should see the level indicator bars on the objects going up and down. You should also see the Pipes flashing. If a Pipe is drawn in gray, it is empty. If it is a pulsing color, material is flowing through the pipe. If it is a solid color, the material is blocked. There will also be flowitems being changed into fluid and fluid being changed into flowitems.



After completing this lesson your should have an idea of how these Fluid Objects work and some of their capabilities. There is far more that they can do than was 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.

Tutorial - Process Flow Basics

This tutorial is designed to give you a hands-on introduction to the Process Flow module. In this tutorial you will create a very simple process flow that runs as its own independent simulation model. Be aware that in this tutorial, you will not learn how to link a process flow to a 3D simulation model yet. Connecting a process flow to a 3D simulation model will be covered in the Linking Process Flows to 3D Models tutorial. In this tutorial, you'll create a simulation model of a post office with one service window. Customers will be represented by tokens, which will arrive approximately every 60 seconds. The service time at the window is about 45 seconds. If a customer waits for longer than 200 seconds, they will leave the post office without going to the service window and will be considered an unhappy customer.

Tasks Covered

This tutorial will cover the following tasks:

1. Adding and connecting process flow activities
2. Renaming and resizing activities
3. Linking Shared Asset activities
4. Editing activity properties
5. Creating a Process Flow chart
6. Running the simulation and gather information
7. Using labels and Decide activities to add complexity

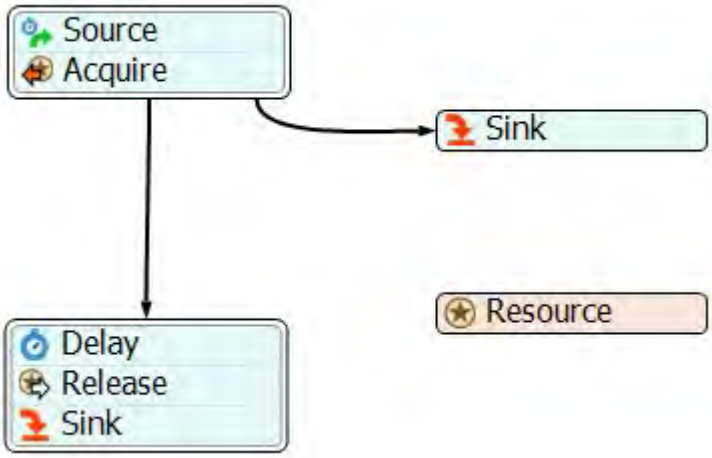
For More Information

For more in-depth information about the concepts covered in this tutorial, refer to the following topics:

- Overview of Process Flow
- Navigating in Process Flow
- Overview of Process Flow Objects
- Adding and Connecting Activities
- About Shared Assets
- Editing Activity Properties
- Running a Process Flow Simulation
- Creating and Using Charts

Step 1 - Adding and Connecting Process Flow Activities

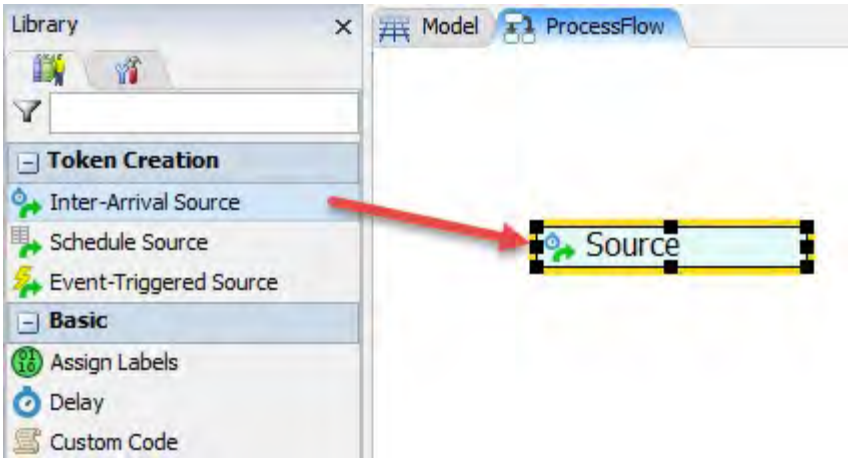
In this first step, you'll create a process flow. When you're finished with this step, your process flow should look approximately like the following image:



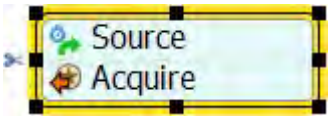
When you create the process flow, you'll make a General Process Flow because this type of process flow is ideal for creating global process flows. Then, you'll begin adding and connecting some activities to your process flow. (An activity is a logical operation or step in a process flow.) There are a number of different ways you can add and connect activities. In this step, you'll try out all of these methods. You can use whichever method you prefer when you begin building your own process flow.

1. Create a new model, then click the ProcessFlow button on the main toolbar to open a menu. Select Add a General Process Flow to create a general process flow and open it as a separate tab in the center pane. Also, notice that when the Process Flow view is open and active, the Library changes to display the process flow activities.

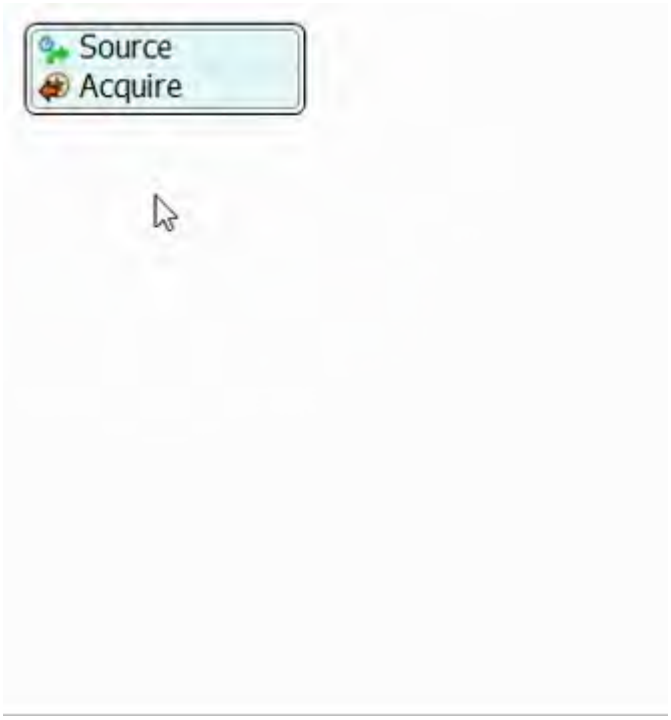
2. First, you'll need to add an activity that will create the tokens that will move through the other activities. In this model, the tokens will represent customers coming to the post office. Since customers will arrive at random, you'll use an Inter-Arrival Source to represent them. From the Library, under the Token Creation group, drag an Inter-Arrival Source activity into the process flow.



3. Now you'll add an activity that will represent a customer standing in line waiting to receive service from a postal worker at the service window. If the postal worker is busy, the customer will wait in line until the postal worker is free. Because the postal worker is a limited resource, you'll use an Acquire Resource activity for this step in the process flow. From the Library under the Resources group, drag out an Acquire Resource activity and drop it on top of the Source activity. Notice that the two activities have been snapped together, as shown in the following image. They are now automatically connected in a *block* (also sometimes referred to as a *stacked block*.)



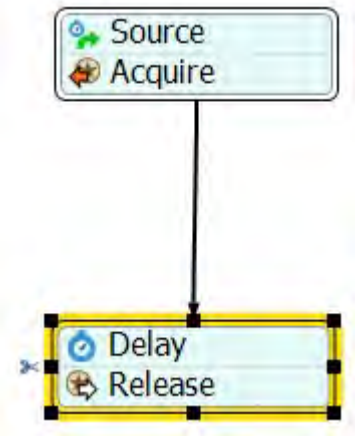
4. Now you'll try the second method for adding and connecting an activity using a connection coming out from the stacked block. Point your mouse on the bottom edge of the block. The mouse icon will change to a chain link. Click the edge of the block and, while holding down the mouse, drag it a little bit toward the bottom of the screen. Notice that there is a connector coming from the edge of the block to your mouse pointer. When you release the mouse, the Quick Library will appear. Under the Basic group, click the Delay activity , which will represent the time it takes a postal worker to help a customer. The following animated gif shows this step:



5. If needed, you can delete a connection by clicking it and pressing the Delete key. You can also change the connector's settings by clicking on it and editing the settings in Quick Properties. You can change the curve of the connection arrow by clicking on it and using the control handles to change its curves.

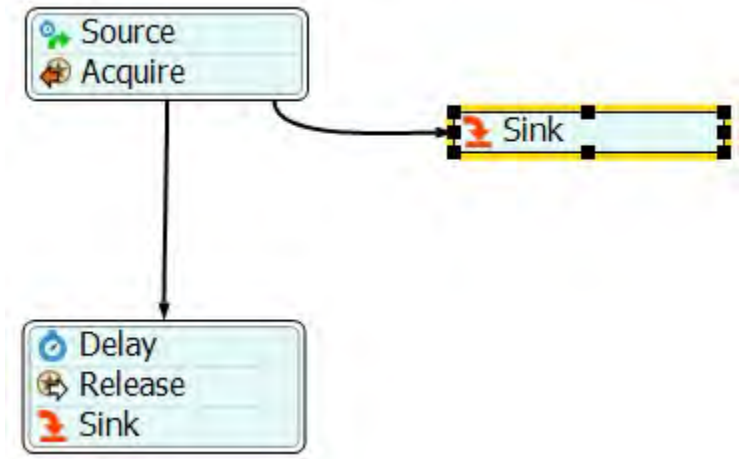
6. Now you'll try the last method for adding and connecting a new activity by adding an activity directly to the end of a stacked block. Point your mouse on the bottom edge of the Delay activity. The mouse icon will change to a chain link. Double-click this edge to open the Quick Library again.

7. Under the Resources group, click on the Release Resource activity . This activity will release the postal worker to help another customer.



8. Using your preferred method, add and connect a Sink activity  below the Release Resource activity. The Sink activity will remove the token (the customer) from the model and track the number of customers that exited the process flow through this particular Sink.

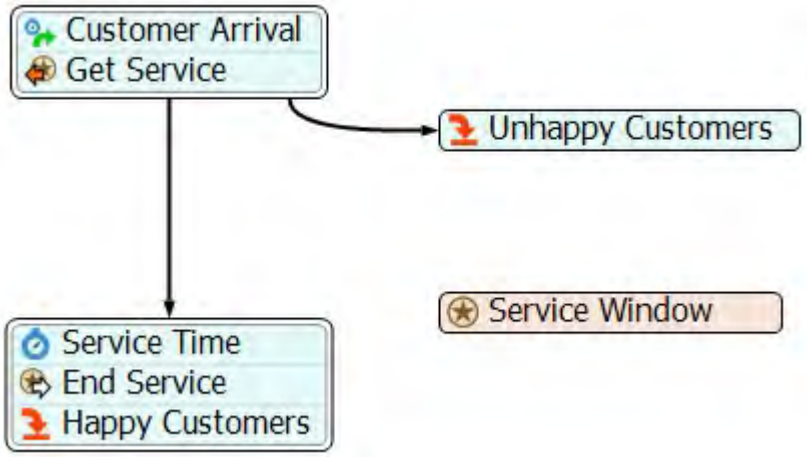
9. Now you will need to create an exit point for customers who are unable to get service in a reasonable amount of time. Some activities can have more than one outgoing connector, such as the Acquire Resource activity. Create a second connector coming out from the Acquire activity and connect it to a Sink activity, as shown in the following image. This Sink activity will be for customers (tokens) that will leave if the wait is too long.



10. From the Library, under Resources, drag a Resource activity into the process flow. This activity will represent the postal worker who will help the customers. For now, don't worry about connecting this activity to any other activities.

Step 2 - Rename and Resize Activities

In this step, you'll rename all of the activities to be more descriptive. When you finish this step, your process flow should look approximately like the following image:



This step in the tutorial will demonstrate two different methods for renaming activities. Be aware that changing an activity's name will have no effect on the way that activity behaves, but renaming activities will make your process flow a little more intuitive and understandable. This step will also explain how to resize activities, since some of the activities will be too small to display the new names.

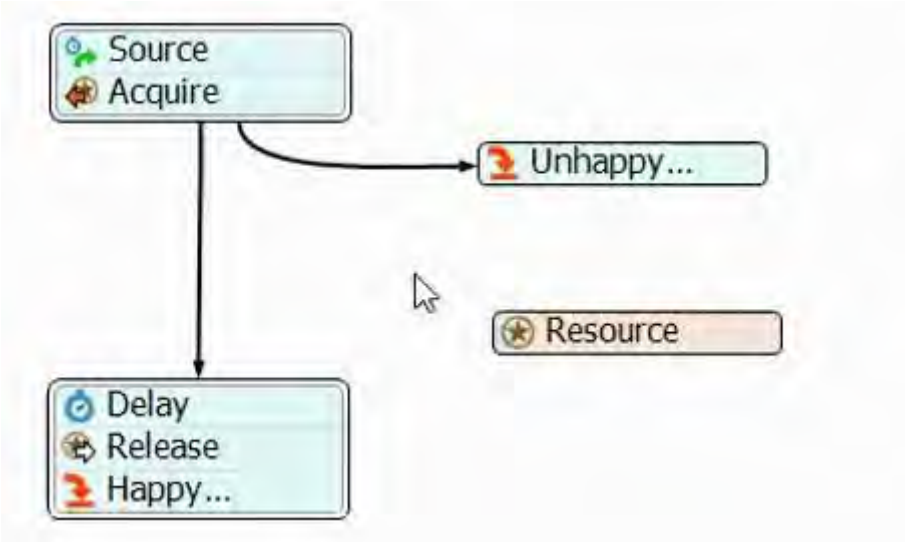
1. Click the first Sink activity (the one connected to the Acquire activity) to select it.

2. In Quick Properties under Activity Properties, click inside the Name box. Delete the current text and type *Unhappy Customers* to change the Sink activity's name, as shown in the following image:

3. Now you'll try the second method of changing an activity name. Double-click the second Sink activity (the one connected to the Release activity) to highlight its name. Type *Happy Customers* as the new name and hit the Enter key.

4. Notice that the names of the Unhappy Customers and Happy Customers activities are truncated because they're too long. Now you'll resize the activity block to make the full name visible. Click the Unhappy Customers activity to select it.

5. Notice that when the activity is selected, the entire border of the activity block turns yellow and the black sizer boxes appear. Click the right middle sizer box on one of the activities and drag it a little to the right until the full name becomes visible. Repeat this step for the activity block.



6. Using your preferred method, rename the following activities as listed in the following table:

Activity	New Name
Source	Customer Arrival
Acquire	Get Service
Delay	Service Time
Release	End Service
Resource	Service Window

7. Resize the activities if needed.

Step 3 - Link Shared Asset Activities

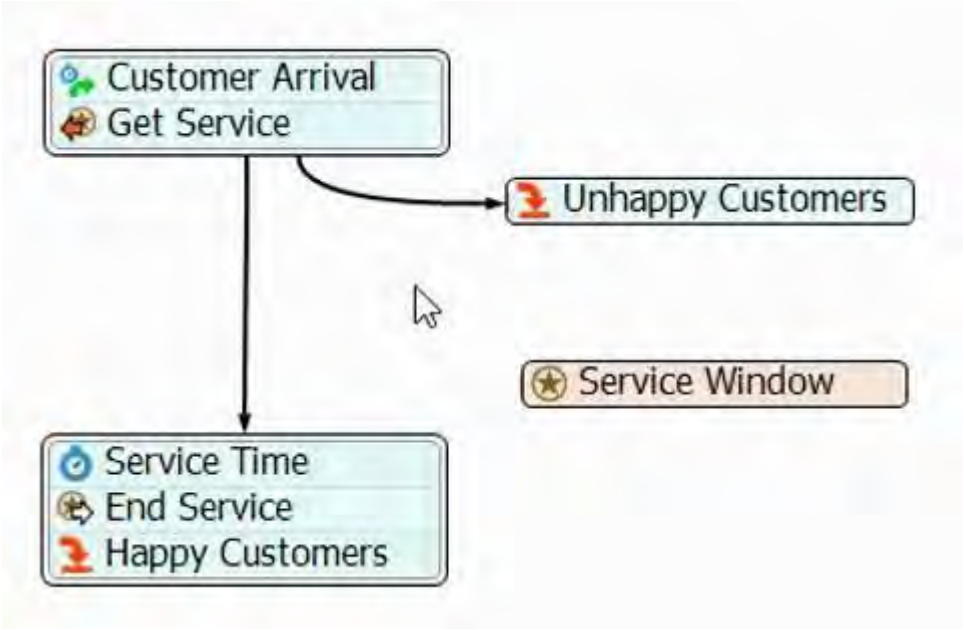
You might have noticed that the Resource activity is a different color than the other activities. That's because it is a *shared asset*. Shared assets don't interact with tokens the way other activities do. They can be used to build complex logic into process flows by controlling access to resources. (See About Shared Assets for more information.) The post office model will begin with only one employee, which is represented by the Resource shared asset in this model. In one sense, the post office employee is a finite resource. When that employee is busy helping a customer, he or she can't help other customers until the current customer's request has been completed.

In this step, you'll connect the Acquire Resource activities to the Resource shared asset so that when customers arrive at the post office, they will request access to the service window.

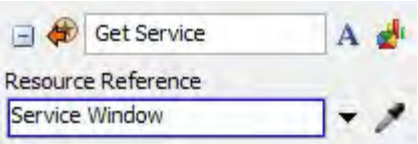
1. First, you'll connect the resource activities so that when customers arrive at the post office, they will request access to the service window. Click the Get Service activity. Notice there is a red Exclamation icon to the right of the activity. This means that this Acquire Resource activity is not connected to a Resource activity, which is required.

2. Click the Exclamation icon. The mouse will turn into a sampler to indicate you are in sampling mode.

3. Click on the Service Window resource to link the two activities. When they are linked, a blue line will appear when the activities are selected.



When you finish this step, you should be able to click the Get Service activity to select it. Then, you should see that in Quick Properties, the Resource Reference box is now set to *Service Window* as shown in the following image:



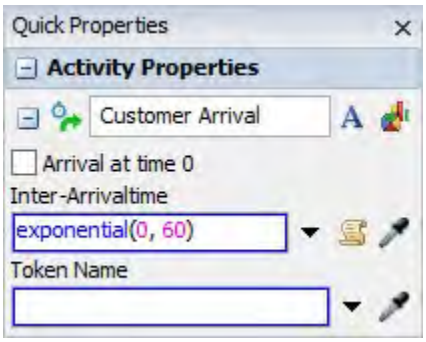
Fortunately, you don't need to link the Service Window (Resource) shared asset to the the End Service (Release Resource) activity because FlexSim will automatically release any resources that are associated with the tokens when they enter a Release Resource activity.

Step 4 - Edit Activities Properties

In this step, you'll create the basic logic behind your process flow by editing the activity settings. Using two different methods for editing activity properties, you'll change the customer arrival time schedule, set a maximum wait time for customers, and set the time it takes to provide service to a customer.

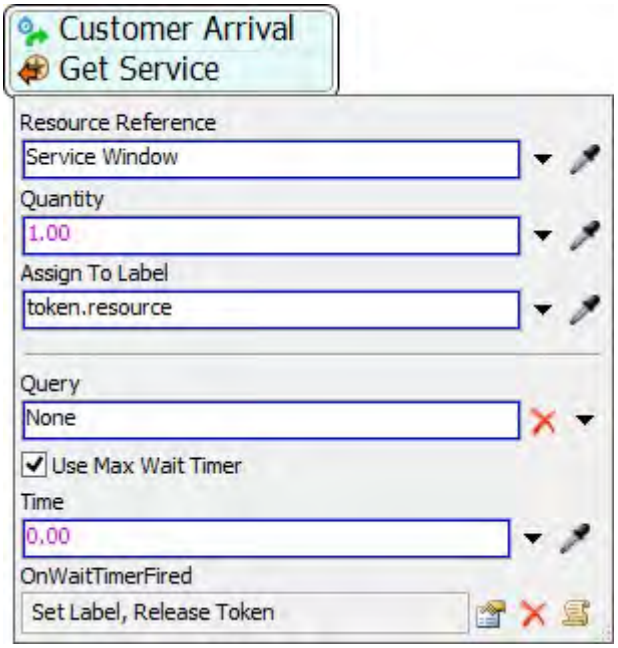
1. First you'll edit an activity's properties using the Quick Properties pane. You'll start by changing the arrival rate for customers so that a customer arrives approximately every minute. Click the Customer Arrival activity to select it.

2. Notice that Quick Properties displays all the properties for all the activities in this block. In Quick Properties, under the Customer Arrival, change the Inter-Arrivaltime to `exponential(0, 60)`. Now customers will arrive at random intervals, approximately every 60 seconds. As an alternative to typing this code in directly, you can click the arrow next to this property and select Statistical Distribution from the menu to use a different statistical distribution.



3. Now you'll try a different method for editing an activity's properties. You'll change the maximum wait time so that when a customer waits for more than a few minutes, they get frustrated and leave. Click the Acquire Resource icon (as labeled in the following image) on the Get Service activity to open the activity properties:

4. In the activity properties, check the Use Max Wait Timer checkbox to turn on the maximum wait time. A few new settings will appear underneath the checkbox.

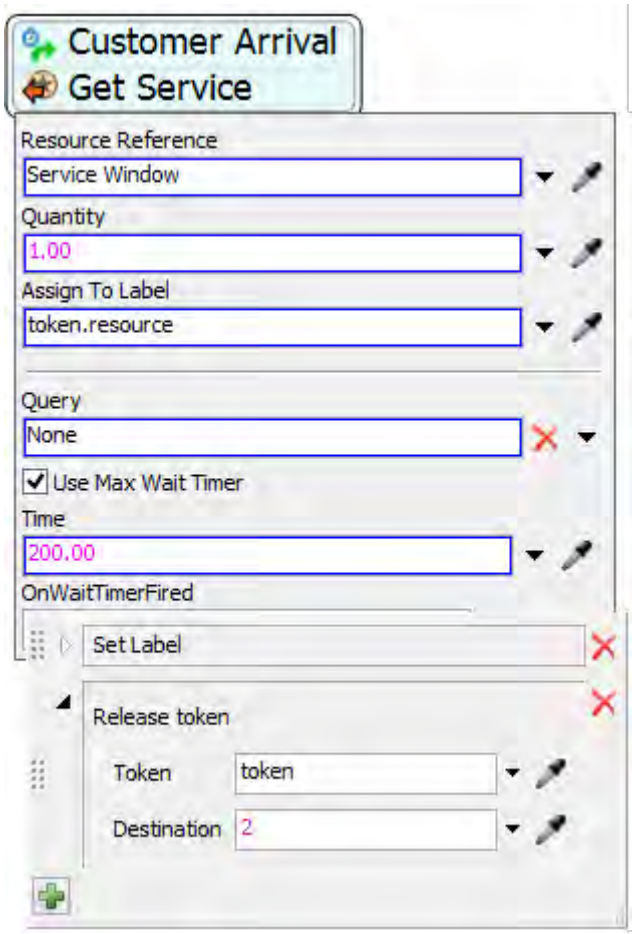


5. In the Time box, type *200*. 200 seconds is the length of time the customers will wait for the service window before getting frustrated and leaving.

6. Now you need to determine what happens when the customer reaches the maximum wait time. Click the Edit Properties button next to the OnWaitTimerFired trigger.

7. Click the arrow next to the Release Token picklist to open its picklist options.

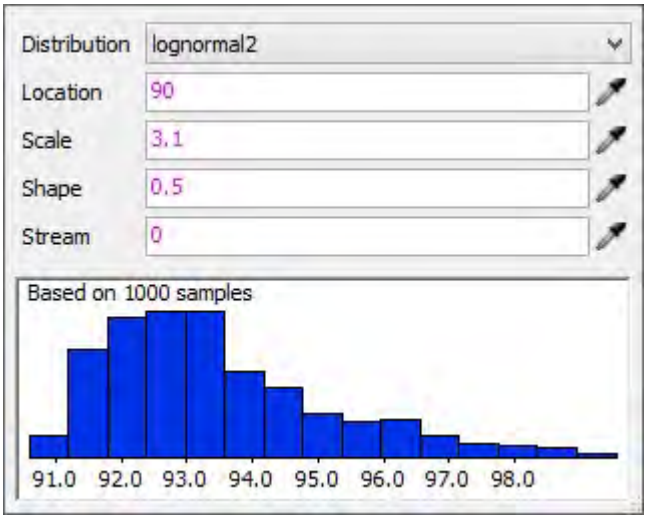
8. In the Connector box, type *2*. This setting will send customers (tokens) that reach the maximum wait time to the second connector: the Unhappy Customers activity.



9. Click outside of the activity properties to save the changes.

10. Now you'll change the statistical distribution that calculates the amount of time it takes for a customer to complete their order at the service window. Using your preferred method for changing an activity's properties, open the properties for the Service Time activity. Click the Edit Properties button next to the Delay Time box to open the Distribution Chooser.

11. From the Distribution menu, select the lognormal2 distribution. Edit the parameters as shown in the following image:



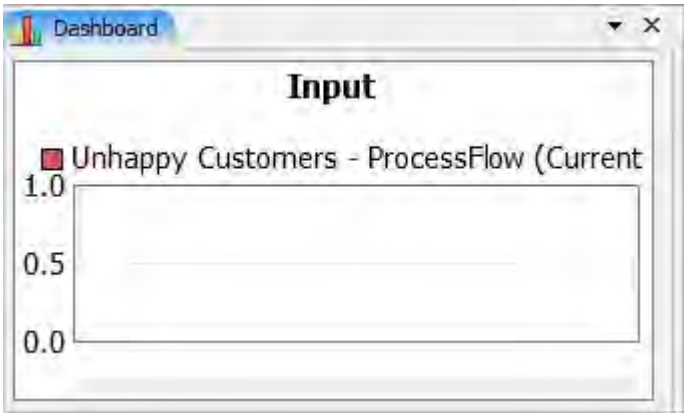
12. Notice in the chart that most customers will take approximately 91.5 to 93.5 seconds to complete their order, but some might take a little more or less. Click outside the Distribution Chooser to save the changes. Now the Delay Time will read lognormal2(90, 3.1, 0.5, 0).

If you haven't done so already, consider saving your model.

Step 5 - Create a Process Flow Chart

Now that all the logic of your process flow is set up, you'll create a dashboard to record the number of happy customers and unhappy customers created by this process flow.

1. Click the Unhappy Customers activity to select it.
2. In Quick Properties, click the Statistics button button to open the statistics window for this activity.
3. You'll want to track how many tokens enter this activity. So, next to the Input box, click the Pin button to open a menu. Select Pin to New Dashboard to create a new dashboard and add a chart for this statistic, as shown in the following image:

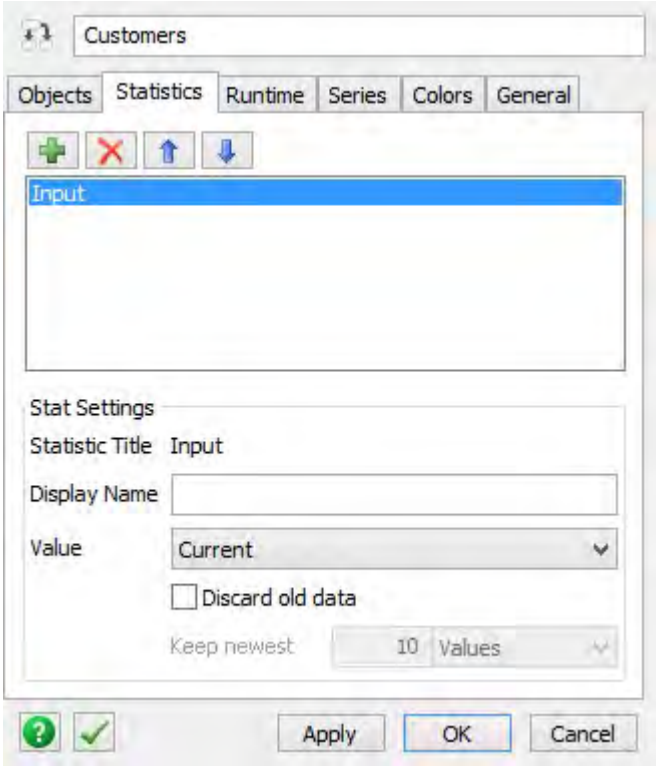


4. Now you'll add the input for the happy customers to this chart for comparison. Double-click the chart to open its Chart Properties window, as shown in the following image:

5. In the Name box at the top, delete the current chart name and type *Customers* instead.

6. In the Objects tab, click the Sampler button to enter sampling mode. Click the Happy Customers activity to add it to the chart. When you're finished, the list on the Objects tab should display both the Unhappy Customers and Happy Customers activities in its list.

7. Now you'll make sure that the chart is tracking the current input for both these Sink activities. In the Statistics tab, click Input in the statistics list to select it. Make sure that the Value menu below the list is set to Current.



8. If desired, you can make any visual changes to the chart in the Colors and General tabs.

9. Click the OK button to save the changes and close the window. Notice that the chart does not yet display any information because the simulation hasn't started running yet.

10. Now you'll add one additional chart to your dashboard that will track the average staytime that customers wait for service. Click the Get Service activity to select that block of activities.

11. In Quick Properties, next to the Name box for the Get Service activity, click the Statistics button button to open the statistics window for this activity.

12. Next to the Staytime box, click the Pin button to open a menu. Select Pin to Dashboard to add a chart for this statistic to the existing dashboard.

13. Now you'll change the statistic value and other visual properties for this chart. Double-click the chart to open its Chart Properties window.

14. In the Name box at the top, delete the current chart name and type *Average Wait Times* instead.

15. In the Statistics tab, click Staytime in the statistics list to select it. Then, in the Value menu, select Average.

16. In the General tab, clear the Show Legend checkbox.

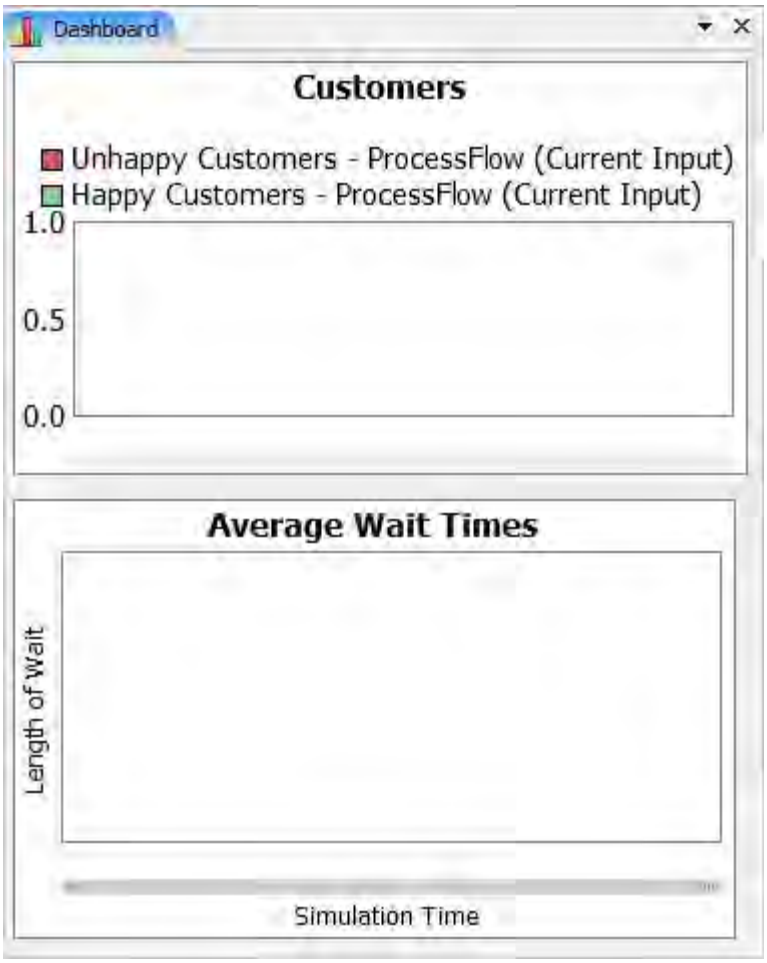
17. In the Y Axis Title box, type *Length of Wait*.

18. In the X Axis Title box, type *Simulation Time*.

19. Click the OK button to save the changes and close the window.

20. If needed, you can adjust the display size of the charts by clicking on a chart to select it and then using its sizer boxes.

When you're finished, your Dashboard should look like the following image:



Step 6 - Run the Simulation and Gather Information

Running a process flow simulation is just the same as running a simulation for a standard 3D model. You might want to make it so the dashboard and the process flow are split into two panes in the center pane. See Navigating in Process Flow for more information about how to split panes.

1. On the simulation control bar, press the Reset button to clear any data from the process flow.

2. Then press the Run button to watch the simulation at work.

3. As you watch the simulation, there should only ever be one token in the Service Time activity and there should be multiple tokens waiting in Get Service activity.

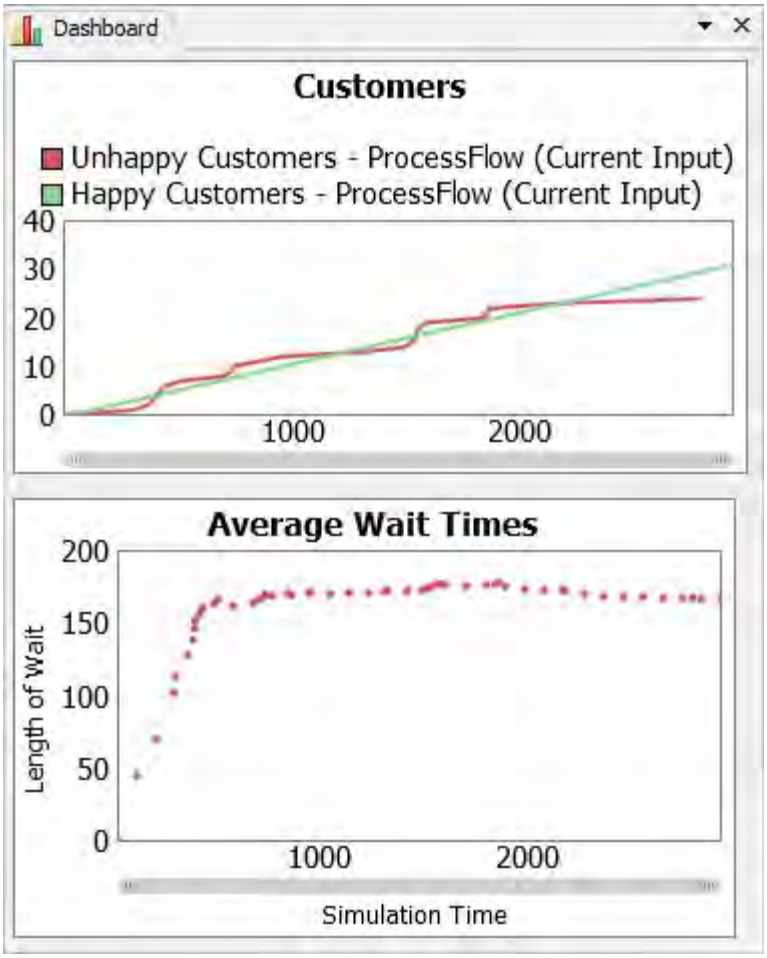
4. You'll also notice that some customers will make it to Happy Customers and some will move to Unhappy Customers. You'll notice that you'll only ever see tokens inside the Get Service activity and the Service Time activity. This is because all of the other activities perform their logic in 0 time with no created events. If you

want to see a token move through each activity you can use the Step button to step a token through each of its activities.

5. You'll notice there is a green and red circle that will appear on the Service Window resource as you run. The green icon shows how many tokens have acquired a resource. In this case we only have 1 resource available so this number will never be any higher than 1. The red icon is the number of tokens that are attempting to acquire a resource but cannot because none are available.

6. Lastly, notice that your dashboard updates its information.

The following image shows what the dashboard could possibly look like after running for 3,000 seconds:



Notice that there are only slightly more happy customers than unhappy customers, which isn't a good sign. Also notice that the average wait times are very close to the maximum wait time of 200 seconds. Clearly, this is a business process that could be improved.

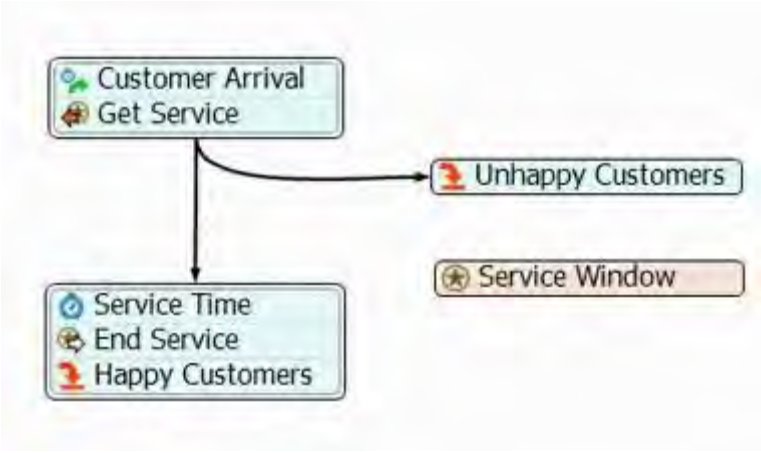
If you were using the Experimenter, you could experiment by increasing the number of available service windows to determine the ideal number of employees needed at the post office.

Step 7 - Use Labels and Decide Activities to Add Complexity

For the final step in the tutorial, you'll add some additional complexity to simulate customers who come into the post office to see how long the line for service is. If these customers see that the line for service has

more than 5 people in it, they will leave without service. You will add this complexity using a Decide activity that checks a label that is dynamically assigned based on the size of the service line.

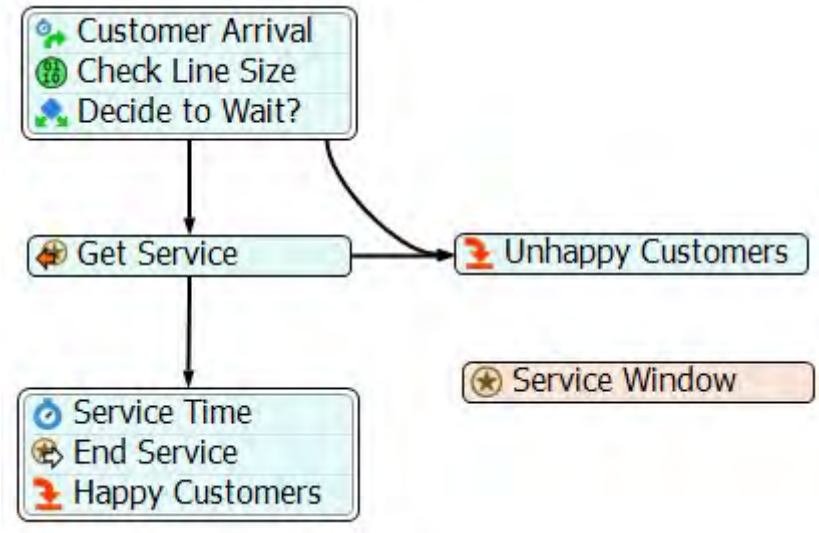
1. First, you need to separate the activities in the first stacked block so that you can add the eventlistening activity to it. Click the stacked block with the Customer Arrival activity to select it.
2. Click the Scissors to the left of the block to separate the two activities in the block, as shown in the following image:



3. Using your preferred method for adding an activity, add an Assign Labels activity below the Customer Arrival activity. Then add a Decide activity below the Assign Labels activity. Feel free to move the activities if you need more room in your process flow workspace.

4. Using your preferred method for renaming an activity, change the name of the Assign Labels activity to *Check Line Size*. Then change the name of the Decide activity to *Decide to Wait?*.

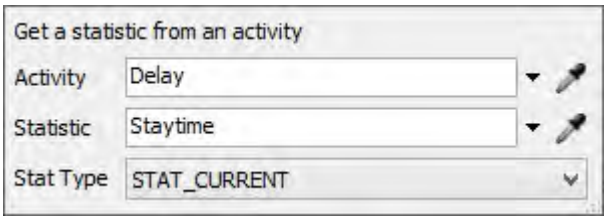
5. Create two outgoing connectors from the Decide to Wait? activity. One connector should go to the Get Service activity and the other should go to the Unhappy Customers activity. NOTE: Make sure there is also still a connector going from the Get Service activity to the Unhappy Customers activity.



6. Now you'll edit the Check Line Size activity properties so that it will assign a dynamic label that will check the current line size. Click the Check Line Size activity to select it.

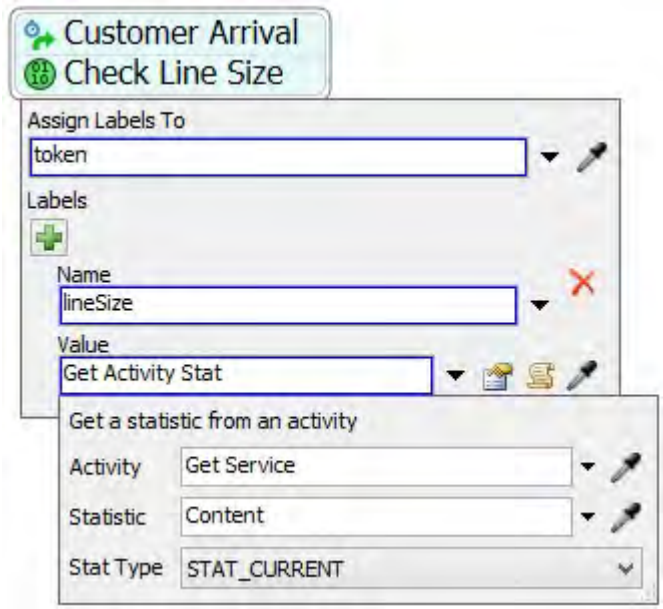
7. In Quick Properties, click the Add button under Labels to add a new label.

8. In the new label's Name box, delete the existing text and type *lineSize*.
9. Click the arrow next to the Value box to open the picklist menu for this property. Select Get Activity Stat to open the following picklist options:

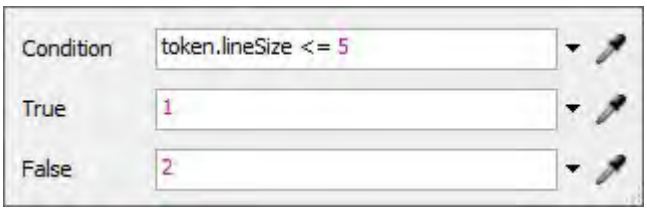


10. Next to the Activity box, click the Sampler button to enter sampling mode. Click the Get Service activity to sample it.

11. Next to the Stat Type box, click the Sampler button to enter sampling mode. Click the Get Service activity to open a menu of its available statistics. Select Content from the menu. Now the label will assign a value to the token based on the size of the line just before it goes to the Decide activity.



12. Now you'll change the properties for the Decide activity. In Quick Properties, click the arrow next to the Decision box to open the picklist menu for this property. Point to Conditional Decide to open the following picklist options:



13. In the Condition box, edit the default code so that it reads: `token.lineSize <= 5`. This code tells it to return a value of *true* if the value of the lineSize label on the token is less than 5. If it is true, the token will move to

the activity connected to the outgoing connector with the rank of 1 (the Get Service activity). If it is false, the token will move to the outgoing connector with a rank of 2 (the Unhappy Customers activity).

14. Lastly, you'll check to make sure that the outgoing connectors are ranked correctly. In the Connectors Out menu, make sure that 1: is selected. The To box should say it is connected to the Get Service activity. If not, click the Up arrow next to the Rank box to change it to rank 1.

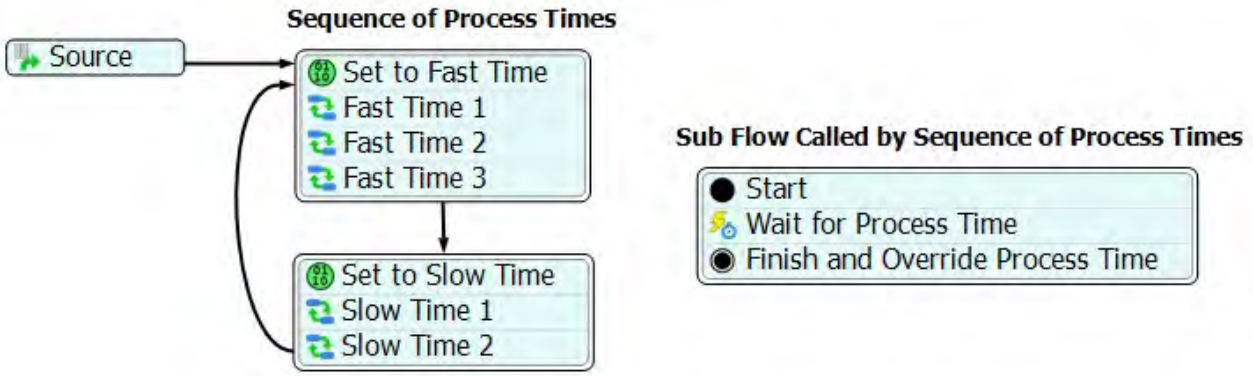
Reset and run your model and look at the change in your statistics. How do they change the Value of Check Line Size to be 3? or 10? Feel free to experiment with adding more service windows by changing the Count on the Service Window resource.

Tutorial - Linking Process Flows to 3D Models

The purpose of this tutorial is to demonstrate how you can link a process flow to a simple simulation model using an event-listening activity. It will also cover how to make a sub process flow for tasks that will be repeated multiple times during a simulation run.

In this scenario, you want to be able to easily change the process time for flowitems in a processor on a repeating pattern. The first three flowitems that enter the processor should have a fast processing time of 2 seconds. The next two flowitems will have a slower processing time of about 10 seconds. Then this pattern will repeat itself indefinitely for the all subsequent flowitems.

To create this kind of logic, you'll build a process flow that will loop in an indefinite pattern. And then you will create a sub flow with an event-listening activity that will override the processor's process time.



Each time the sequence needs to override the processor's process time, it runs the sub flow, shown in the preceding image. The token that's moving through Sequence of Process Times creates a child token, dispatches it to the Start activity of the Sub Flow, and then waits for the sub flow to finish before moving to its next activity.

Tasks Covered

This tutorial will cover the following tasks:

1. Building a simple 3D model
2. Building a looping process flow
3. Setting labels to control the process time
4. Building and linking a sub flow
5. Linking the process flow to the simulation model
6. Running the model

For More Information

For more in-depth information about the concepts covered in this tutorial, refer to the following topics:

- Linking Process Flows to Simulation Models

- Key Concepts About Event-Listening
- Event Types
- Sub Process Flows

Step 1 - Build a Simple 3D Model

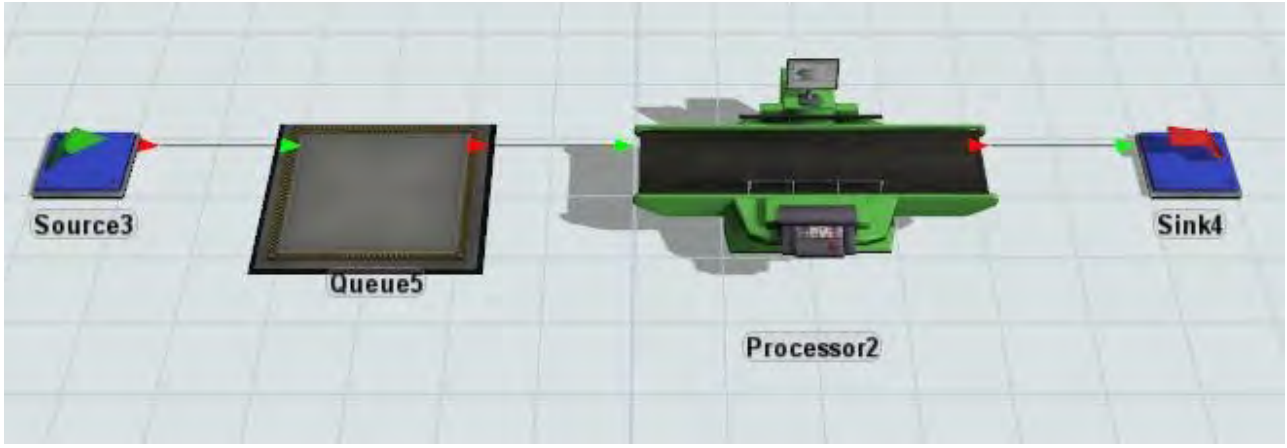
Since the focus in this tutorial is going to be on the process flow, you'll just build a very simple 3D simulation model in this step.

1. Create a new model.

2. From the Library, add:

    o 1 Source o 1
    Queue o 1
    Processor o 1
    Sink

3. Create port connections (A connects) between the objects in the order that you dragged them out, as shown in the following image:



Step 2 - Build a Looping Process Flow

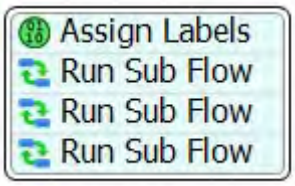
Now you will create the process flow that will control the processing time logic in the Processor. Because you want the processing times to loop in a sequence (3 fast processing times followed by 2 slow processing times), you'll create single process flow and then make it loop indefinitely.

NOTE: This step will include a brief review of the concepts covered in the Process Flow Basics Tutorial. It will also discuss the purpose of each activity for this process flow. If you don't need these step-by-step instructions for building a process flow, you can skip to the end of this step and view the final image to see how the final process flow should look. You can create your process flow based off that image if it is faster for you.

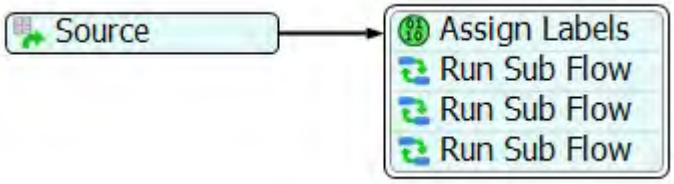
1. Click the ProcessFlow button on the main toolbar to open a menu. Select General to create a general process flow and open it as a separate tab in the center pane. You might want to split the center pane between the model view and the process flow view for your convenience.

2. Click a Schedule Source in the Library and add it to the process flow. This source will create the first (and only) token that will loop through the process flow as soon as the simulation begins.

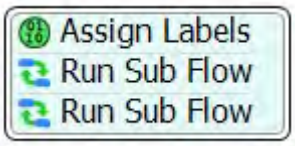
3. Now you'll create a sequence of activities that will make the process time run quickly for the first three flowitems that enter the Processor. The first activity will assign a label to the token and trigger a sub flow three times (one for each of the fast processing times). Using your preferred method for adding activities, create a stacked block with 1 Assign Labels activity and 3 Run Sub Flow activities, as shown in the following image:



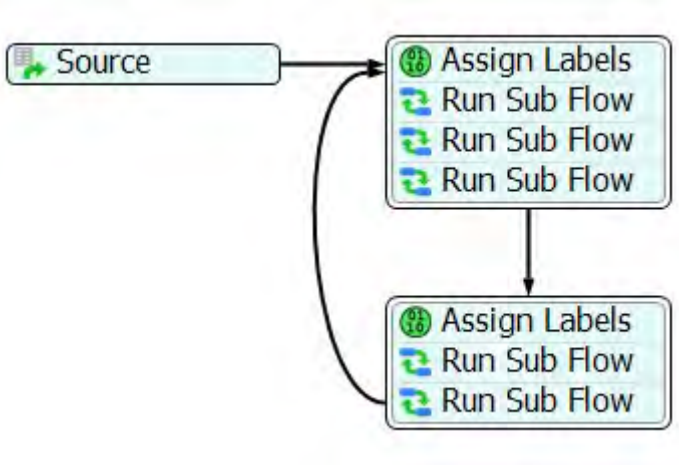
4. Create a connection between the Source and the Assign Labels activity. Point your mouse on the edge of the Source. The mouse icon will change to a chain link.  Click the edge of the block and, while holding down the mouse, drag it to the edge of the Assign Labels activity. Release your mouse to create the connection.



5. Now you'll create a similar stacked block for the two flowitems that will make the process time run slowly. Using your preferred method for adding activities, create a stacked block with 1 Assign Labels activity and 2 Run Sub Flow activities, as shown in the following image:



6. Create a connection going from the first stacked block to the second stacked block. Lastly, create a connection going from the second stacked block back to the first stacked block, creating a loop, as shown in the following image:

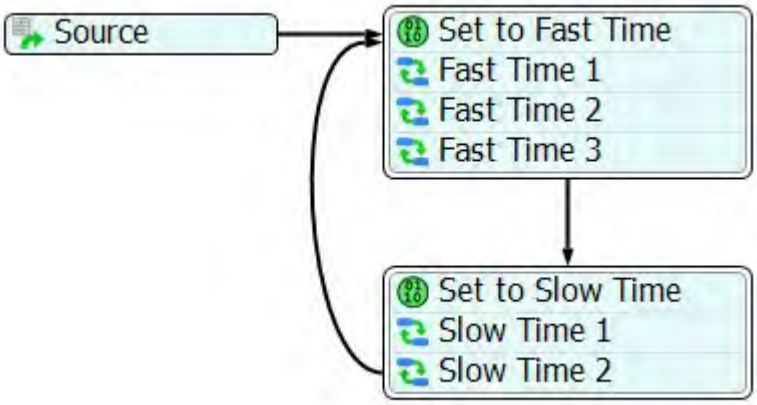


7. Lastly, you'll want to rename the activities for clarification. To rename an activity, double-click an activity name and type in a new name. Rename the activities as follows:

Which Block?	Current Name	New Name
First	Assign Labels	Set to Fast Time
First	Run Sub Flow	Fast Time 1
First	Run Sub Flow	Fast Time 2
First	Run Sub Flow	Fast Time 3
Second	Assign Labels	Set to Slow Time
Second	Run Sub Flow	Slow Time 1
Second	Run Sub Flow	Slow Time 2

8. If needed, you can resize the activities.

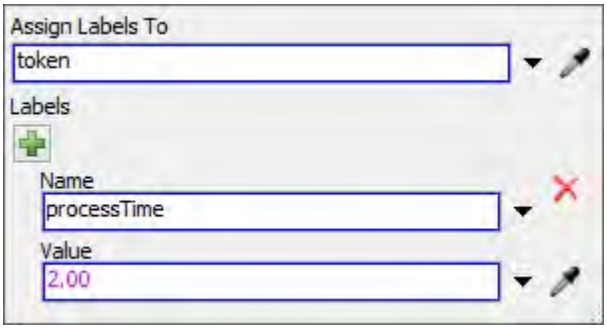
When you're finished, your process flow should look similar to the following image:



Step 3 - Set Labels to Control the Process Time

In this step, you'll edit the properties for the Assign Label activities so that they set the process time that will be given to the Processor. For this example, static numbers will be used but these numbers could be replaced with a statistical distribution, pull data from a global table or perform other logical operations.

1. Click the Set to Fast Time (Assign Label) activity in the first stacked block to select it.
2. In Quick Properties, click the Add button under Labels to add a new label.
3. In the new label's Name box, delete the existing text and type *processTime*.
4. Click in the Value box and type *2* to represent 2 simulation time units.



5. Repeat these steps for the Set to Slow Time (Assign Label) activity, except this time change the Value box to *10* to represent 10 simulation time units.

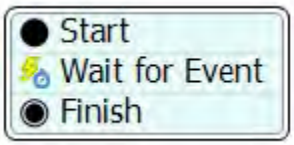
Step 4 - Build and Link a Sub Flow

In this step, you'll create a sub process flow (also called a *sub flow* for short). A sub flow is a separate process flow that begins running when it is triggered by another activity or event in the main process flow. You can possibly use sub flows for tasks that will be repeated multiple times during a simulation run. In this case, you'll create a sub flow with an event-listening activity that will get triggered every time a flowitem is processed by the processor.

For the purposes of this tutorial, you'll create an *internal* sub flow, which is a sub flow inside of an existing process flow. Internal sub flows are basically an independent block of activities that begin with a Start activity and end with a Finish activity. (See Sub Process Flows for more information.)

NOTE: This tutorial model relies on a useful process flow feature that allows child tokens to inherit labels from their parent tokens. In this tutorial, you'll set the labels on a parent token in the main process flow. Every time the main process flow runs a sub flow, it will create a child token that will move through the sub flow itself. The child token will automatically inherit the *processsTime* label from its parent token. It will then use that label as the override value for the process time in the child token. (See Sub Process Flows - Linking Parent and Child Labels for more information.)

1. Using your preferred method for adding activities, create a stacked block with a Start activity, a Wait for Event activity, and a Finish activity.



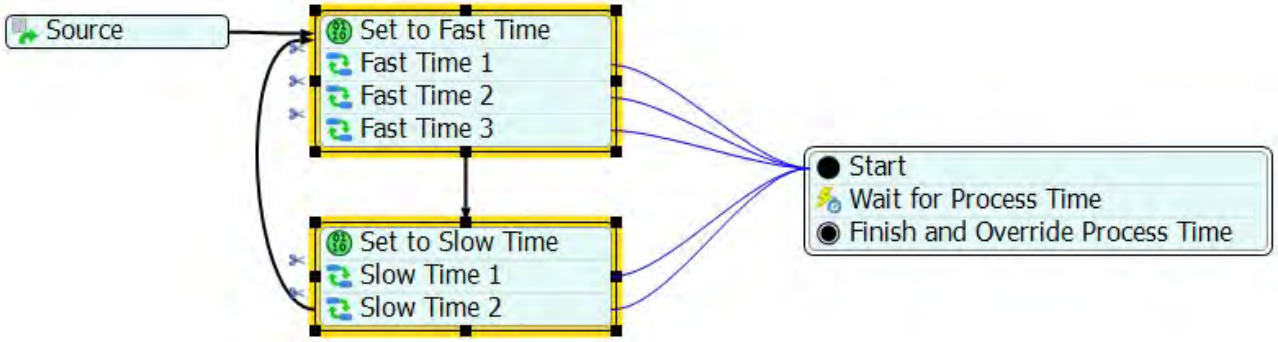
2. Rename the activities in this block for clarification. Rename the activities as follows:

Current Name	New Name
Wait for Event	Wait for Process Time
Finish	Finish and Override Process Time

3. Now it's time to link the Run Sub Flow activities to the Start activity on the sub flow. Click on the first stacked block to select it. You'll notice that each of the Run Sub Flow activities have an Exclamation icon next to them, which indicates they are not linked to a sub flow. Click the Exclamation icon to enter sampling mode, then click the Start activity on the sub flow. A blue line will appear to show they are linked.

4. Repeat this process until all five of the Run Sub Flow activities are connected to the sub flow.

When you're finished, your process flow should look similar to the following image:



Step 5 - Link the Process Flow to the Simulation Model

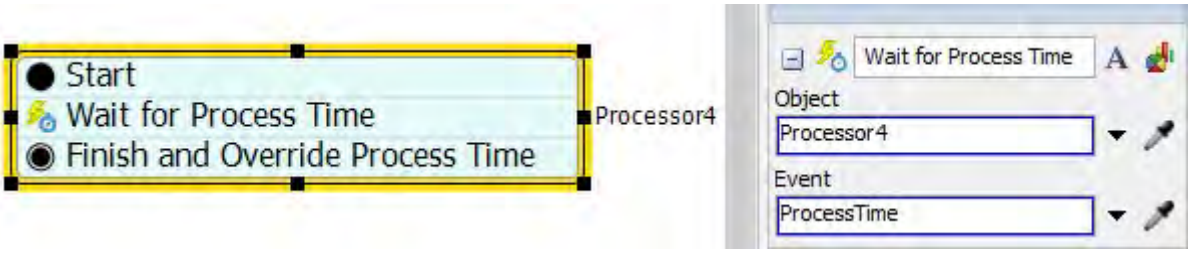
In this step, you'll link the event-listening activity on the sub flow (Wait for Process Time) to the processor so that it will override its process time. You'll also get to experiment with a Universal Edit keyword. (See About the Universal Edit Feature for more information.) NOTE: You might want to split the center pane between the simulation model and the process flow view before you start this step.

1. Click the Wait for Process Time (Wait for Event) activity to select it. You'll notice that the Wait for Process Time activity has an Exclamation icon next to it it, which indicates it is not linked to an event in the simulation model yet.

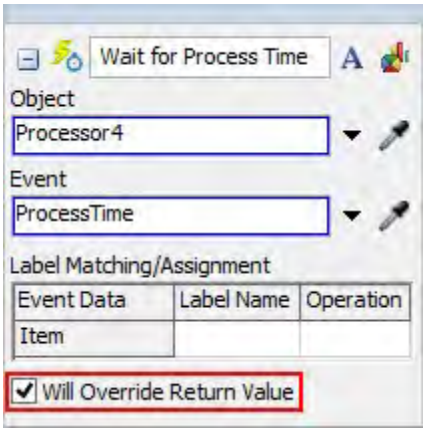
2. Click the Exclamation icon to enter sampling mode, then click the Processor in the simulation model. NOTE: If you didn't split the center pane, point your mouse to the Model tab while in sampling mode to switch to the 3D view.

3. Clicking on the Processor in sampling mode will open a menu listing all the events that might occur on the processor. Select the Processor: ProcessTime event. NOTE: The exact name display of the processor might differ in your model.

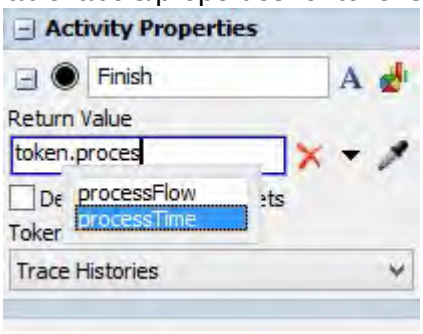
4. Notice that the Wait for Process Time (Wait for Event) activity now displays the name of the processor next to it. In Quick Properties, the Object box also displays the name of the processor and the Event box displays the name of the event you selected, as shown in the following image:



5. Now you will change the settings so that the current value of the token's processTime label will override the processor's process time. In Quick Properties, check the Will Override Return Value checkbox.



6. Lastly, you'll get a chance to see how a property that uses the Universal Edit feature works. The Universal Edit feature is designed to make it easy to add complex functionality to certain properties without needing to know FlexScript. Using keywords, you can set a property to access data that could possibly change dynamically during a simulation run such as labels, groups, variables, and more. In the Quick Properties for the Finish activity, click in the Return Value box. Type the keyword `token.` to open a menu of all the available labels/properties for tokens in your process flow, as shown in the following image:

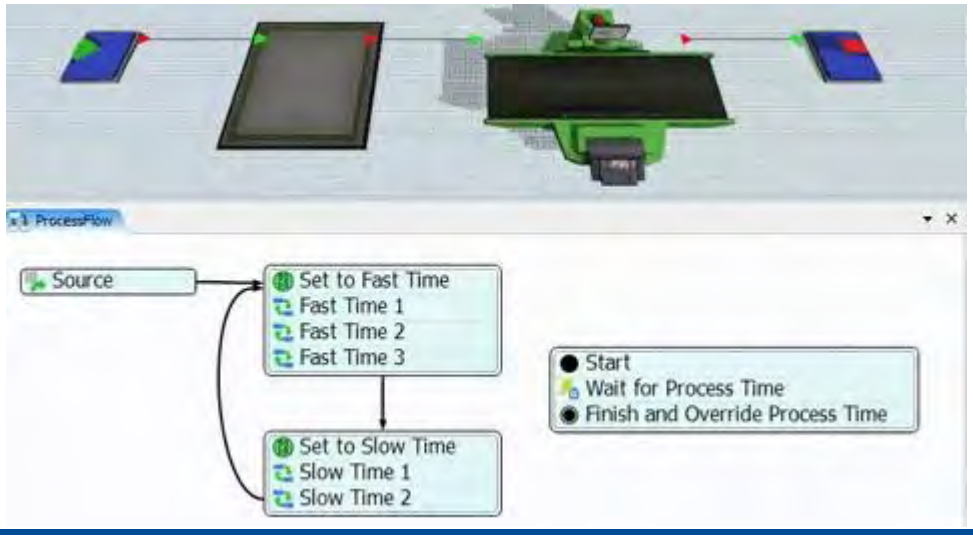


▾ Alternatively you can click on the to the left of the Return Value box and select *processTime* from the Labels sub menu.

7. Double-click processTime or use the arrow keys and the enter key to select that label. Now, the sub flow will automatically override the process time of the processor with the value of this label.

Step 6 - Run the Model

Reset and run the model. You'll notice that Source creates a token at time 0, the token enters the Set to Fast Time activity and its *processTime* label is set to 2. The token then moves to the first Fast Time 1 activity where a child token is created and sits at the Wait for Process Time activity in the sub flow. As soon as the processor in the model receives an item, its Process Time will be evaluated and instead of using the value defined by the token, the child token will be released from its activity and move to the Finish activity evaluating the Return Value which will be returned to the Process Time. The child token is then destroyed and the parent token moves to the next Run Sub Flow activity to start sub flow over again. This will continue as each item enters the processor. Once the token finishes Slow Time 2, it will move back to the top to continue the pattern again. Not only was the logic easily created without any coding, but you can also visibily see where the processor is in its set of process times. This can be a valuable debugging tool or it can be useful for collecting statistics. Statistics could be gathered separately or collectively for an an activity or area to record how long the processor spent in any given state.



Tutorial - Task Sequences

This tutorial will teach you how to create task sequences in the Process Flow module and link them to a 3D model. For this tutorial, you will create a model with two processors that are manned by two operators. Each operator will use the same task sequence to transport flowitems to the next downstream object. You'll begin by creating a simple task sequence inside a sub flow that will simply tell the operator to load and unload the flowitems. Then, you will make the task sequence more complex by adding tasks that will make the operator clean the processors after they finish transporting the flowitems to the next destination.

Tasks Covered

This tutorial will cover the following tasks:

1. Building a simple 3D model
2. Building a task sequence in a sub flow
3. Attaching the processors and operators to the sub flow
4. Editing the activity properties to add dynamic references
5. Viewing instances during a simulation run
6. Adding more activities to the task sequence
7. Editing the properties of the new activities

For More Information

For more in-depth information about the concepts covered in this tutorial, refer to the following topics:

- General Process Flow Properties
- Sub Process Flows
- Task Sequences
- Editing Activity Properties - About the Universal Edit Feature
- Process Flow Instances

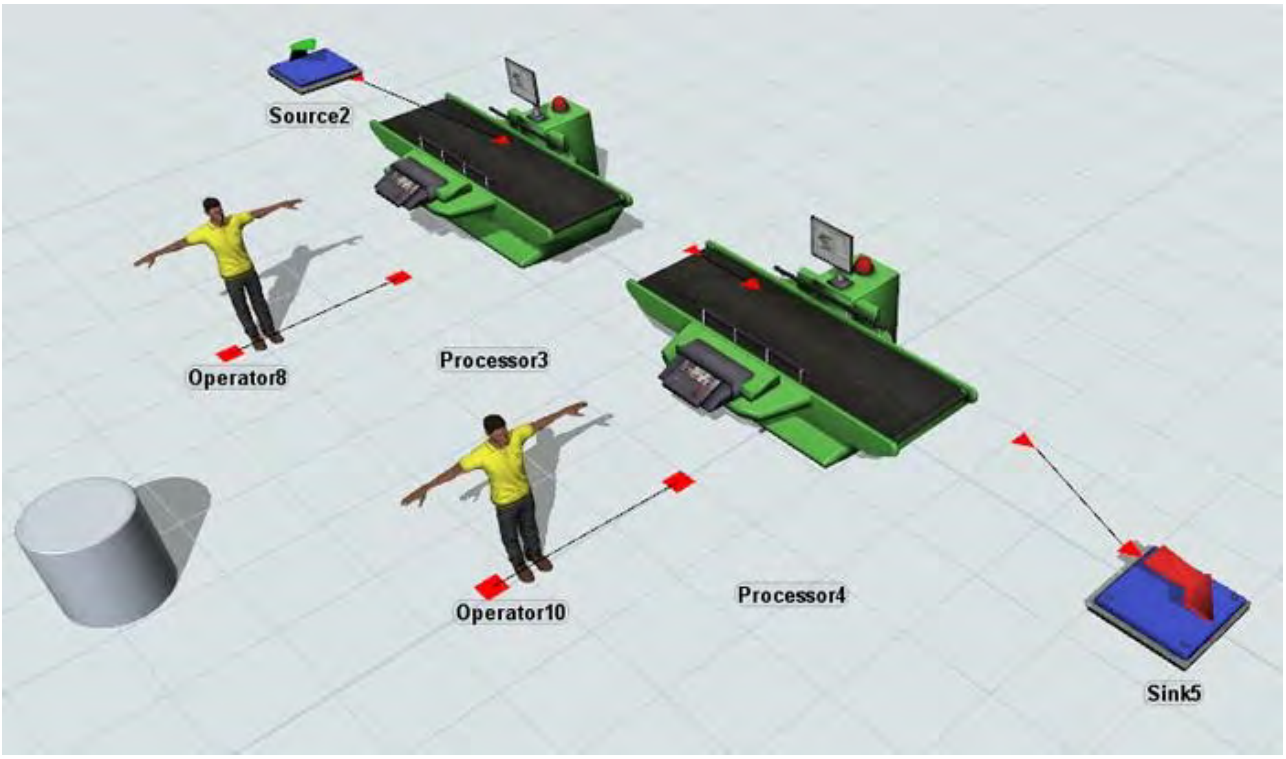
- Troubleshooting Process Flows

Step 1 - Build a Simple 3D Model

First, we'll create the 3D simulation model that will use a task sequence in a process flow.

NOTE: This step will include detailed instructions about which types of objects and port connections you should make while building the model. If you don't need these step-by-step instructions, you can skip to the end of this step and view the final image to see how the simulation model should look.

1. Create a new model.
2. From the Library, add:
    - 1 Source
    - 2 Processors
    - 1 Sink
    - 2 Operators
    - 1 Shape (found under the Visual group in the Library)
3. Create port connections (A connects) between these objects (in this order):
    - The Source
    - The first Processor
    - The second Processor
    - The Sink
4. Create a center port connection (S connect) from the first Processor to the first Operator.
5. Create a center port connection (S connect) from the second Processor to the second Operator.
6. You might need to move the objects so that they look similar to the following image:



NOTE: The Shape object will eventually represent a cleaning supplies storage closet in this simulation model.

Step 2 - Build a Task Sequence in a Sub Flow

Now you'll create a simple task sequence that both operators will use whenever they need to load and unload a flowitem. In this step, you'll set the process flow to run a separate instance each time an operator runs the sub flow. Process flow instances will be explained a little bit in each of the following steps, you can also refer to Process Flow Instances for more information.

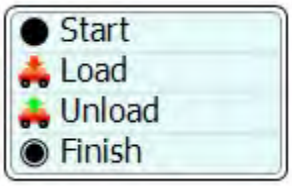
1. Click the ProcessFlow button on the main toolbar to open a menu. Select Sub Flow to create the kind of process flow that is best for running sub flows. It will automatically open as a separate tab in the center pane. You might want to split the center pane between the model view and the process flow view for your convenience.
2. Next, you'll change some of the sub flow's general properties. Start by renaming the sub flow to *ItemTransport* for clarification. In Quick Properties, click in the Process Flow Name box. Delete the current name and type *ItemTransport*.

3. Click the Instance Creation menu and select Per Instance. With this option selected, each operator that runs the task sequence in the sub flow will be considered a separate instance. By default, when you select this option all shared assets will be local to each instance, meaning the two operators won't share assets.

4. Now you'll build the actual task sequence in the sub flow. Using this task sequence, the operator will load and then unload a flowitem. Using your preferred method for adding activities, add the following activities to a single stacked block (in this exact order):

o Start o
Load o
Unload o
Finish

When you're finished, your process flow should look like the following image:



Step 3 - Attach the Processors and Operators to the Sub Flow

Now you'll link this sub flow to the simulation model. On the processor, you'll use a special picklist option that will allow you to dynamically run the task sequence every time the processor uses an operator to transport a flowitem.

1. With the model view open, double-click one of the Processors to open its Properties window.

2. In the Flow tab, check the Use Transport checkbox. This will activate the box next to this property in which you select the task executer that will transport the flowitem. Click the arrow next to this box to open the picklist menu for this property. Select ProcessFlow: Use Task Sequence Sub Flow from the menu to open the options for this picklist.

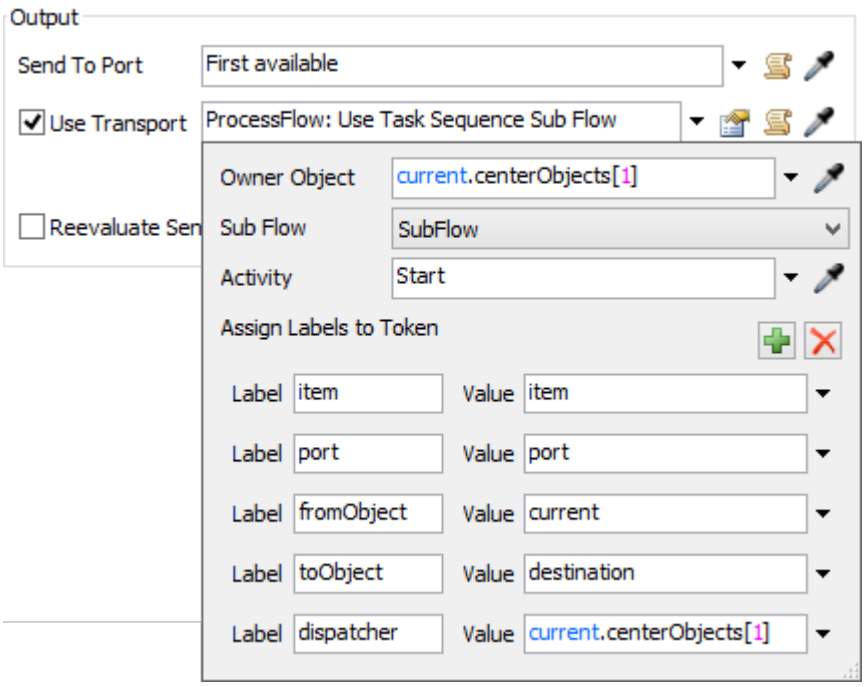
3. Click the arrow next to the Dispatch Object box to open a menu. Select current.centerObjects[1] to attach the operator that is attached to the processor's center port to this sub flow.

4. None of the other properties on this picklist will need to be changed:

    - The Sub Flow menu should display the name of the sub flow you created earlier. (If you didn't change the name, the sub flow would have been named ProcessFlow by default.) You don't need to change this property because it should have automatically selected the sub flow that you created earlier.

    - The Activity box should also have the Start activity selected by default, which is the first activity in your sub flow.

    - You should use the default labels that are assigned to the flowitems when it goes through the processor. The fromObject label assigns the current processor as the pickup location for the flowitem. The ToObject label assigns the object in the outgoing port connection as the destination for the flowitem. Be aware that you'll dynamically reference both of these labels in your sub flow in the next step.



5. Repeat these steps for the second Processor.

Step 4 - Edit the Activity Properties to Add Dynamic References

Now you'll edit the activities in your sub flow so that it is dynamic, meaning it will change based on conditions that are unique to each instance of the sub flow. As a reminder, each time an operator picks up a flowitem from the processor and transports it to its destination, that will act as a separate instance of a sub flow. In this step, you'll also get some experience with properties that use the Universal Edit feature. See Editing Activity Properties - About the Universal Edit Feature for more information.

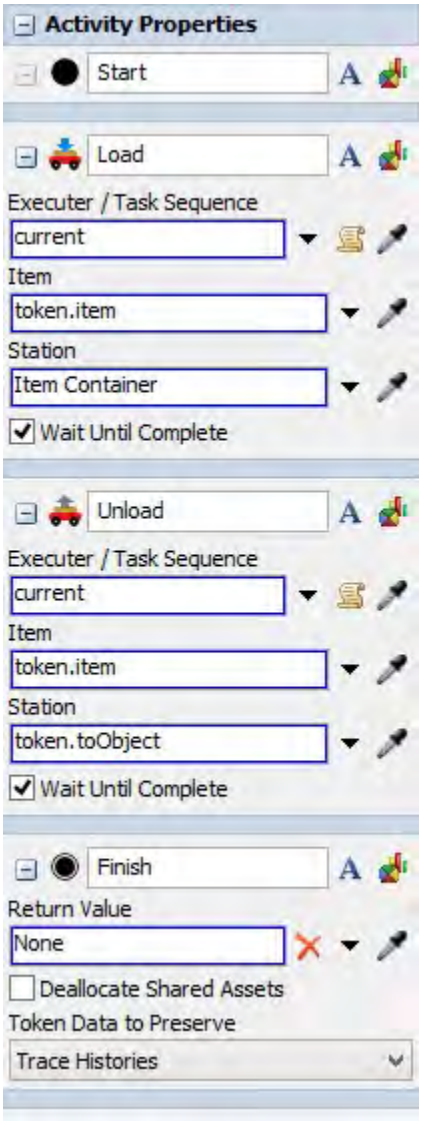
1. With the process flow view open, click the stacked activity block to select it.

2. You'll start by creating a dynamic reference to the operator that is assigned to the task sequence. In the Quick Properties for the Load activity, look for the Executer/Task Sequence box. This property determines which task executer will be assigned to complete this task. In this case, you'll want to replace the text in this property with the keyword *current*. This keyword creates a dynamic reference to the object attached to the current instance of the sub flow. In other words, any time this sub flow is run, it will assign this task to whichever operator is currently attached to that instance of the sub flow.

3. Repeat this step for the Unload activity by typing *current* in the Executer/Task Sequence box.

4. Lastly, you'll create a dynamic reference for the flowitem's destination by referencing the *toObject* label. (This label assigns the object in the outgoing port connection as the flowitem's destination. See Step 3.4 as a reminder.) In the Quick Properties for the Unload activity, click in the Station box and type the following text: `token.toObject`.

When you're finished with this step, the Quick Properties for the stacked block should be the same as the one in the following image:

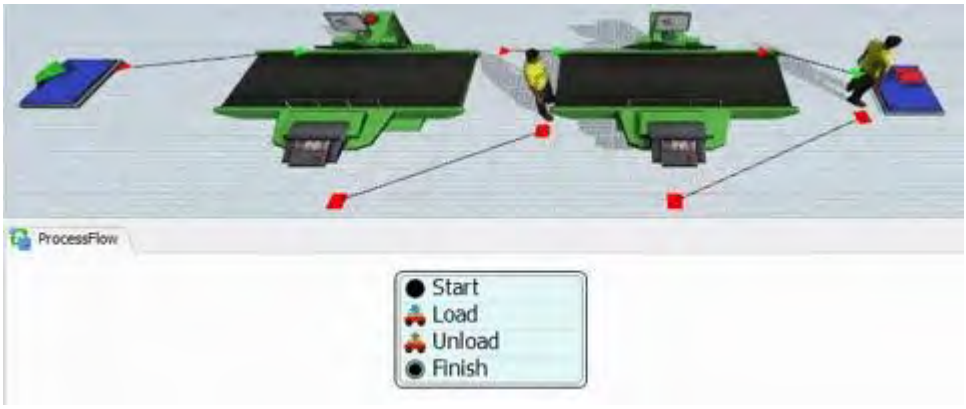


Step 5 - View Instances During a Simulation Run

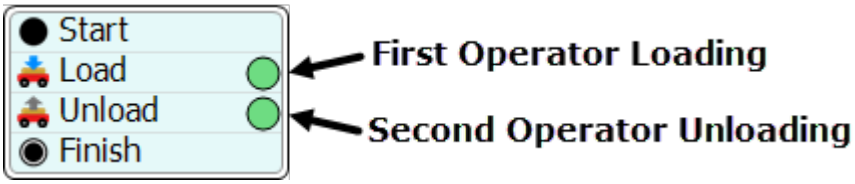
In order to help you get a better understanding of how instances work, in this step you will run the model and view the two different instances of that process flow in action.

1. Run the model by pressing the Reset and Run buttons on the simulation control bar.

2. As the model runs, you should see that any time a flowitem needs to be transported, it appears as a task in the sub flow. Each token represents an Operator performing the task:

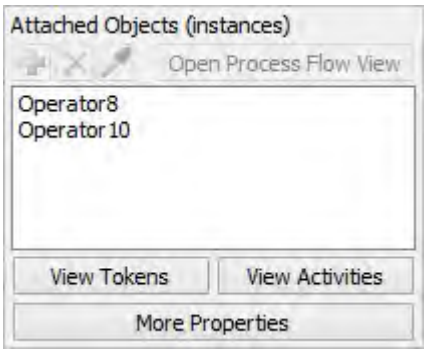


3. Press the Stop button on the simulation control bar to pause the similation. Press the Step button repeatedly to advance the simulation to the next step until you can see two tokens in the process flow at the same time, as shown in the following image:

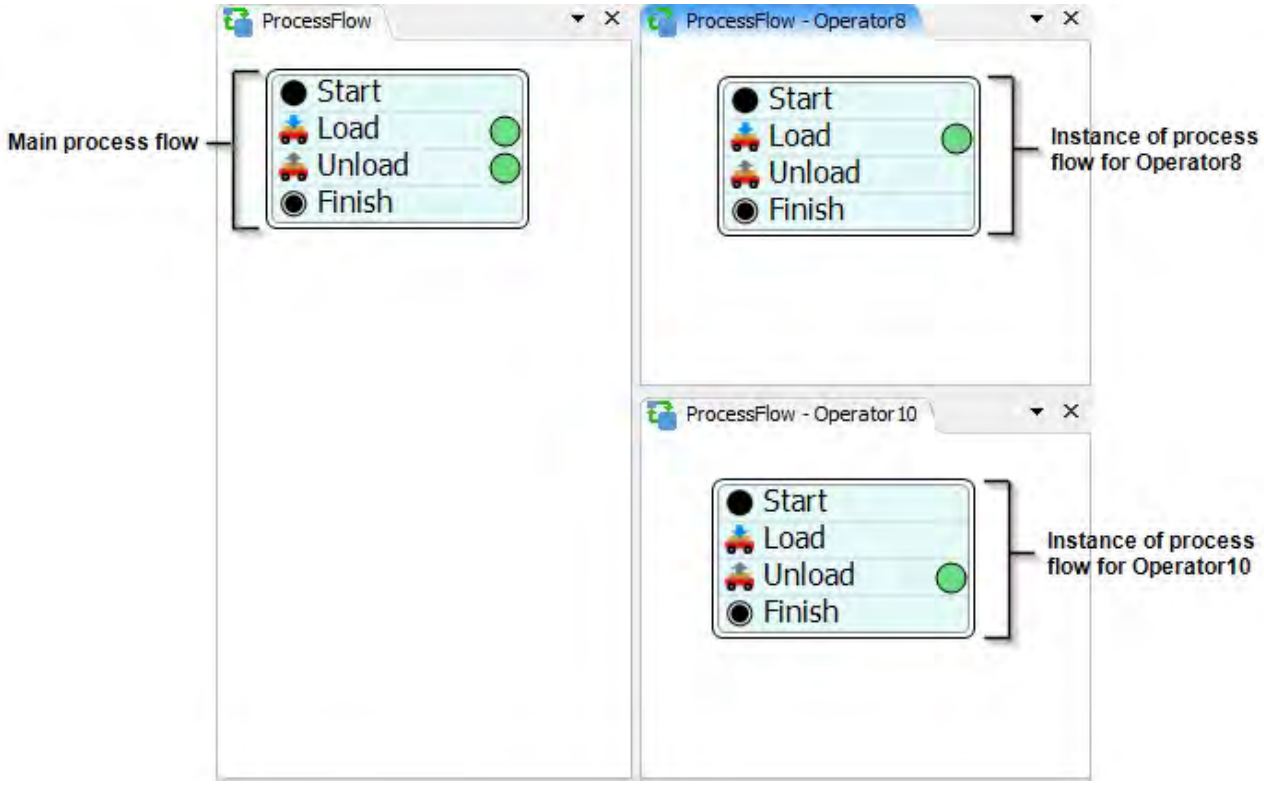


4. Click in a blank spot in the process flow to ensure nothing is selected and to display the process flow's general properties in Quick Properties.

5. In Quick Properties, find the Attached Objects (instances) group. Notice that this lists the two operators that are attached to this process flow, as shown in the following image:



6. Click one of the operators in the list to select it. Then click the Open Process Flow View button to open the process flow for that specific instance. You could also do the same for the second operator. The following image shows what it would look like if you were viewing the main process flow and both instances at the same time:



7. Notice that both tokens are in the main process flow. However, the first token belongs to the first Operator and so it is the only one that appears in the instance for that Operator. The same is true for the second token, which belongs to the second Operator.

Step 6 - Add More Activities to the Task Sequence

In this step, you'll edit the task sequence to make it a little more complicated. You'll add activities that will make the operators clean the processors after they finish transporting the flowitems to the next destination.

1. Press the Reset button on the simulation control bar to stop the current simulation run.
2. Now you need to separate some of the activities in the stacked block. With the process view open to the main sub flow (not the instances), click the stacked block of activities to select them.
3. Click the Scissors to the left of the the block in between the Start activity and the Load activity to separate them.
4. Now click the Scissors to the left of the block in between the Unload activity and the Finish activity to separate them.

5. You can move any of the activities around if you need to put some extra space between the activities as you work on the next few steps.

6. After the Start activity, add a Custom Code activity.

7. Click the stacked block with the Load and Unload activities and drag it to the bottom edge of the Custom Code activity to link the activities together.

8. After the Unload activity, add the following activities (in this order):

   o Travel o Acquire Resource o Travel o Delay o Custom Code o Travel o Release Resource

9. Click the Finish activity and drag it to the bottom edge of the Release Resource activity to link the activities together.

10. Add a Resource activity off to the side of the stacked block without connecting it.

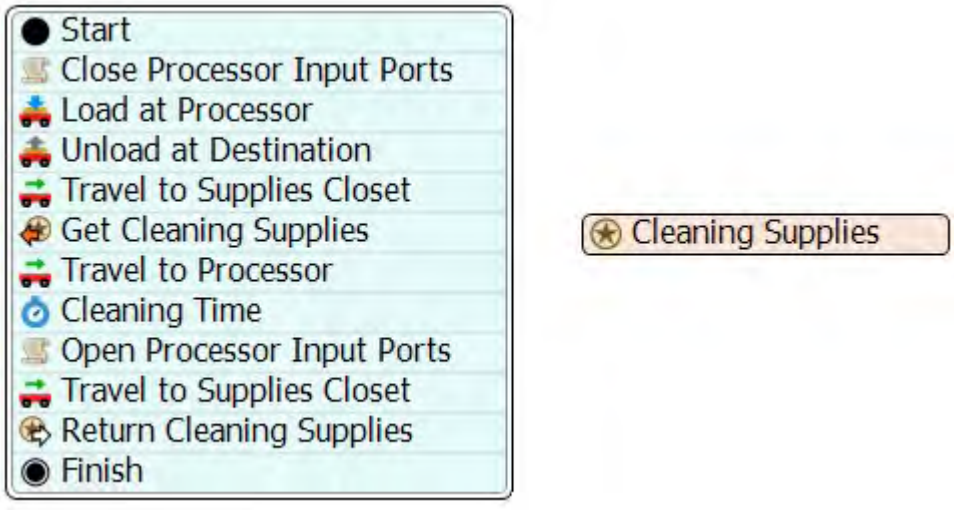
11. Lastly, you'll rename the activities to clarify their purpose in the task sequence. To rename an activity, double-click an activity name and type in a new name. Rename the activities as follows:

Current Name	New Name
Custom Code	Close Processor Input Ports
Load	Load at Processor
Unload	Unload at Destination
Travel	Travel to Supplies Closet
Acquire	Get Cleaning Supplies

Travel	Travel to Processor
Delay	Cleaning Time
Custom Code	Open Processor Input Ports
Travel	Travel to Supplies Closet
Release	Return Cleaning Supplies
Resource	Cleaning Supplies

12. If needed, you can resize the activities.

When you're finished, your process flow should look similar to the following image:



Step 7 - Edit the Properties of the New Activities

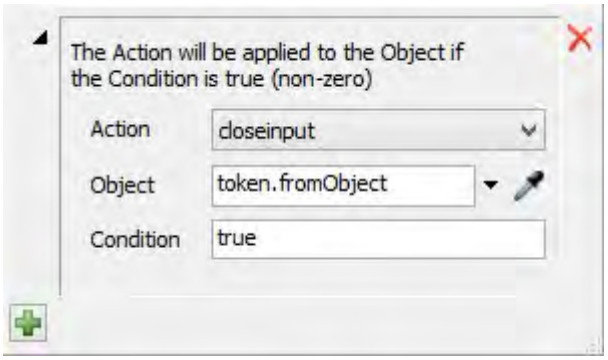
In this step, you'll edit the properties of all the new activities you added to the task sequence. The steps will be logically organized by the activity types rather than the order they appear in the task sequence.

1. Click the activity block to select it.

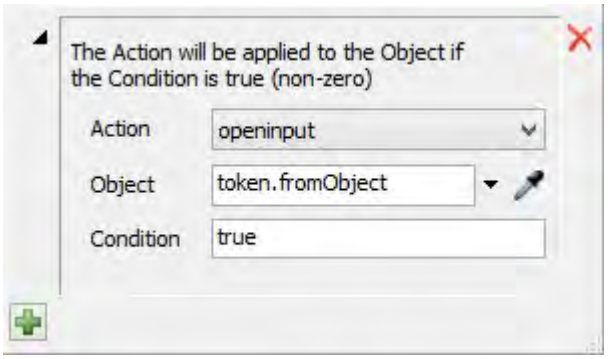
2. You'll start by editing the two Custom Code activities so that they close and open the processor's ports at the right point in the task sequence. In Quick Properties under the Close Processor Input Ports group, click the Add button next to the Custom Code box to open a menu of available picklists. Point to Control, then Close and Open Ports to open its picklist options.

3. Leave the Action menu set to closeinput so that it will close the incoming port of the current processor.

4. Click the arrow next to the Object box to open a menu. Select token.labelName. In this box, change *labelName* to *fromObject*. This setting will make a dynamic reference to the object listed in the token's *fromObject* label. (This label assigns the current processor to this property. See Step 3.4 as a reminder.) In other words, the incoming port from the current processor will be closed.



5. Repeat steps 2-4 for the Open Processor Input Ports activity with one small change: in the picklist options, select openinput in the Action menu. This will open the incoming port of the current processor after the operator finishes cleaning it.



6. Now you'll edit the properties for all the Travel activities so that they are assigned to the correct Operator. In Quick Properties for all three Travel activities, change the Executer/Task Sequence box to *current*.

7. Now you'll edit the properties for all the Travel activities so that they are assigned to the correct Destination. In Quick Properties under the first Travel to Supplies Closet group, next to the Destination click the Sampler button to enter sampling mode.

8. Select the Shape object in the 3D model to open a menu. Select Shape9 from the menu, although the exact number of the object might be different in your model. (This object will represent the supplies closet in the model.)

9. Repeat steps 7-8 for the second Travel to Supplies Closet activity.

10. In Quick Properties under the Travel to Processor activity, click in the Destination box and type token.fromObject to make a dynmaic reference to the object listed in the token's *fromObject* label. (This label assigns the current processor as the destination. See Step 3.4 as a reminder.)

11. Lastly, you'll link the Cleaning Material resource to the process flow. With the stacked block still selected, click the Exclamation icon  next to the Get Cleaning Supplies activity to enter sampling mode, then click the Cleaning Supplies resource.

You'll use the default settings for all the other activities. Be aware that the Cleaning Supplies resource is set to be globally accessible to all instances of the process flow. You also don't need to change the Return Cleaning Supplies settings because the system uses resource labels on the token to determine which resource needs to be returned.

Reset and run your model. You should see the operators moving a flowitem, going to the closet and back to the processor, staying at the processor for a length of time, and then traveling back to the closet. Once they start traveling back to the closet, the processor should start processing again.

See Troubleshooting Process Flows for best practices about fixing a model if your operators are not behaving as expected.

You could possibly experiment with adding some queues in front of the processors so that you could monitor the accumulation of flowitems. You could also experiment with having more than one set of cleaning supplies.

Tutorial - Lists and Resources

Process Flow uses FlexSim's powerful Lists feature. Using Lists, you can:

- Simplify connections in your model
- Sync tokens or objects
- Organize groups of objects
- Track custom statistics
- Use search concepts like filtering and prioritization for making choices in the model

This tutorial will teach you how to use Lists in a process flow. It will also teach how to use Resources, which are another type of shared asset in the Process Flow module. You will build a model where items in a queue are transported downstream by operators, and those operators occasionally require water breaks.

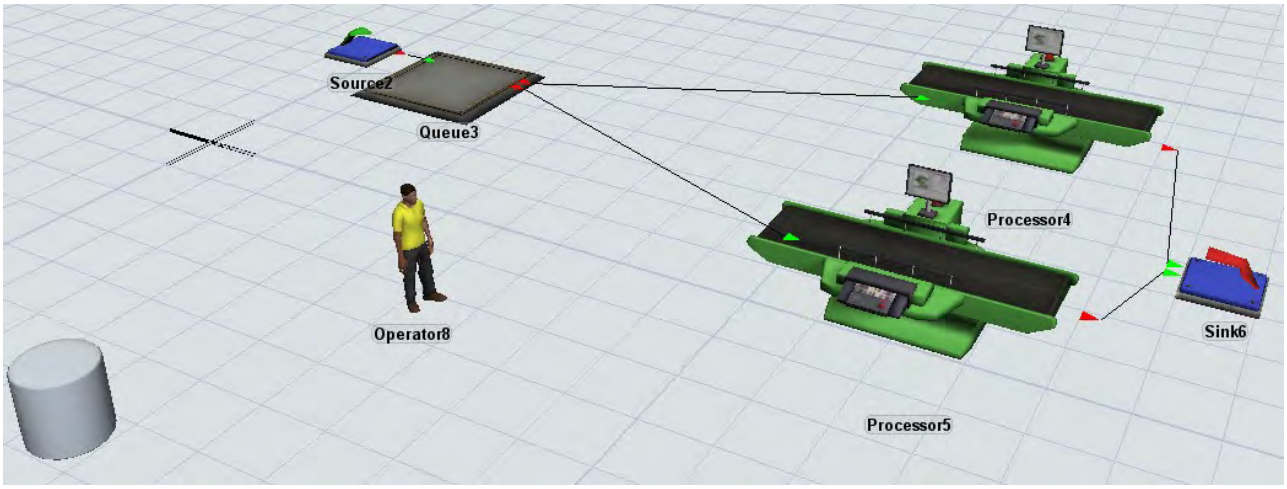
Tasks Covered

This tutorial will cover the following tasks:

- Using Lists to sort and select different objects
- Using Resources to handle Task Executers
- Advanced properties of Lists and Resources

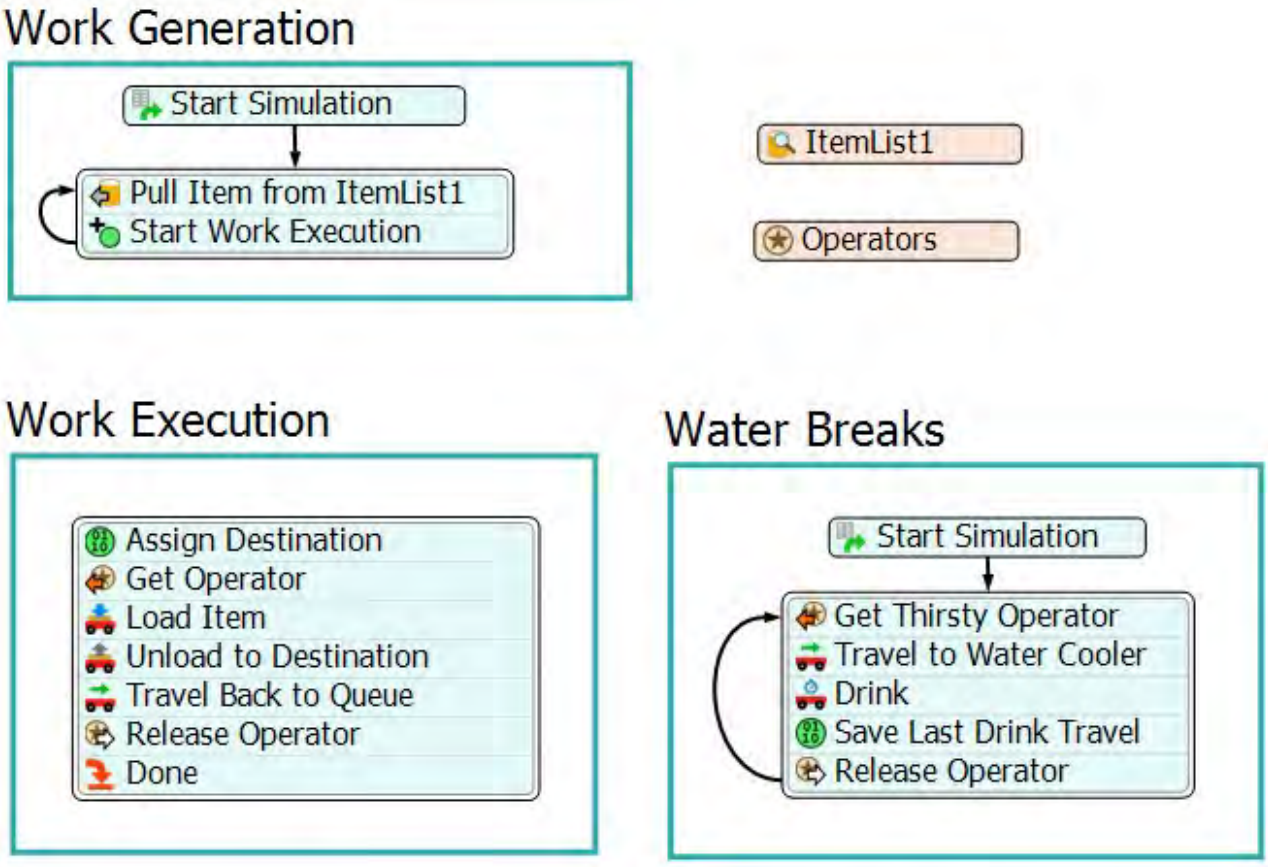
How It Will Work

The model you will build is shown in the following image.



In this model, two operators will be used to move items from the queue to the two processors. Moving items will be their primary responsibility. However, each time an operator travels a total of 100 meters, he or she will become thirsty and need to take a water break. The cylinder shape at the bottom of the model will represent the water cooler location where the operators will go for a water break.

The process flow that will drive the transport of items is shown in the following image.



Standard Functionality Can Do This

FlexSim's standard functionality can actually do much of this logic automatically. All you'd need to do is check the Use Transport box on the Queue and connect it to a Dispatcher. So why use Process Flow? Using Process Flow has several advantages, including:

1. The logic for doing water breaks (based on distance traveled), would require custom code on an operator's On Resource Available trigger, which might be difficult for some users.
2. Using Process Flow gives you better visibility into the state of the model at any given time, and it gives you more detailed control in directing the operators without having to write custom task sequence code (in this model the operator will travel back to the queue after unloading, rather than staying at the processor).
3. Using Process Flow can give you much deeper statistical insight. For example, let's say you want to know how long it takes on average to acquire an operator. This would be difficult to do using the standard mechanism, but it is relatively simple in Process Flow.

That said, if the standard, non-Process-Flow mechanism works for you, by all means use that method. Or you may choose to mix and match between the two methods as needed. Process Flow was designed to allow you the flexibility to choose the method that best meets your needs and capabilities.

Key Concepts

This model will several of the List's features. In order to understand what's going on in the model, you'll need to know a few basic list concepts. See the main list topic for more detailed information on how Lists work.

Lists

At the most basic level, a list is just that: a list of values. In this model you will use two lists:

- A list of items that need to be transported.
- A list of operators that are available to do work.

Lists are dynamic. Their content changes during the simulation run. For example, the list of items to be transported will contain only those items that are in the Queue awaiting transport. Items are added to the list when they become ready to be transported from Queue to Processor. They are removed from the list by the process flow, which then creates the logic that will control the transportation.

Pushing/Pulling and Back Orders

The Process Flow module uses the terms push and pull to represent logic that adds and removes values from a list. Why, you might ask, is it not simply called *adding* or *removing*? The answer is that the pull operation represents much more than just the removal of a value:

- A pull may include a search of the list based on a filtering criteria and/or a prioritization rule.
- Also, a pull operation, if it does not immediately find a value to remove, will create a back order on the list. Once a matching value is then pushed onto the list, the back order will be fulfilled and the value will be immediately removed.
- Additionally, the pull operation includes a return value, namely the value that was pulled from the list.

So a pull operation involves much more than just the removal of a value from the list. For that reason, the specialized terms of *push* and *pull* are more accurate.

List Fields

Lists can also include fields. A field is a defined value associated with the entry on the list. Fields can be used in pull operations to filter and prioritize what should be pulled. In this model you will use list fields to only get "thirsty" operators to take a water break.

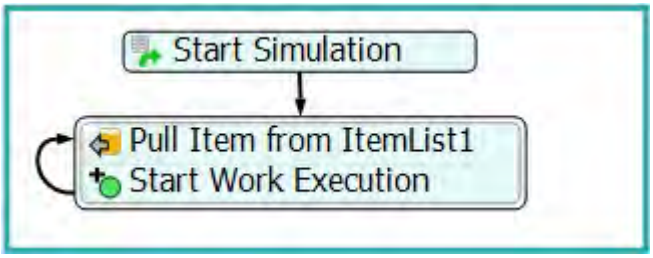
Resources and Lists

Resources represents a limited supply of some resource that can be acquired and released. It can be used to simulate a supply of goods, services, time, materials, employees, etc. In this model you will use a Resource to manage the availability of the operators. When an item needs an operator, a token will acquire the Resource (one of the operators), and then the operator will be released when those tokens are finished with the operator. When the resource operates in object mode, it can internally use a list to manage which operators are available. It will push each of the operators onto this list when the simulation starts. Then, each acquire will pull an operator from the list, and each release will push the operator back onto the list. This means that you can use the same filtering and prioritization features of a list when acquiring a resource.

Process Flow Logic Areas

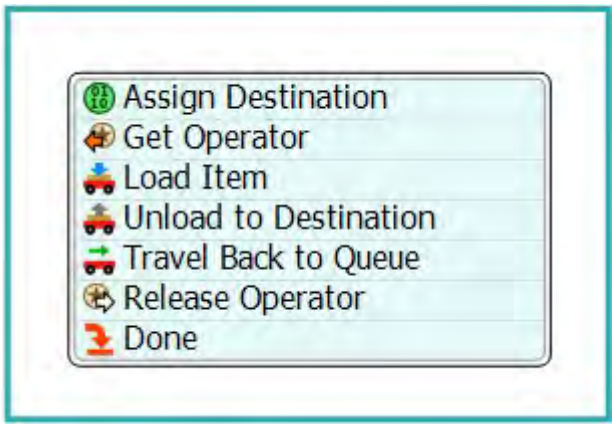
The Process Flow will be separated into three main logical sections: Work Generation, Work Execution, and Water Breaks.

Work Generation



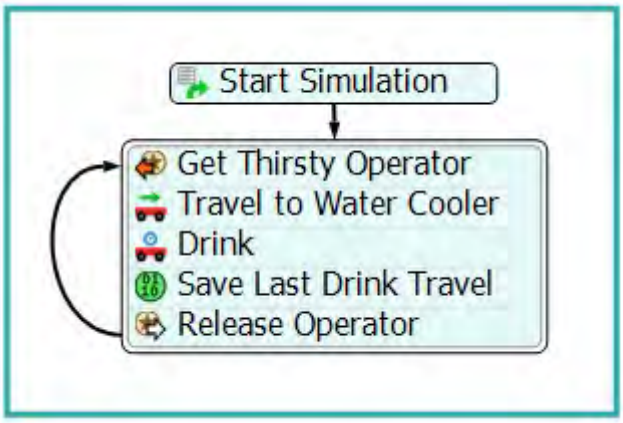
The work generation section is what is generating the transport work that needs to be done. Here you will add a Global List called ItemList1. When an item is ready to be transported from the queue to a processor, logic on the queue will push the item onto ItemList1. Then within the Process Flow, you will pull from the list. Once pulled, you will create a separate token to perform the transport (Work Execution), and then loop back around and pull the next item from the list.

Work Execution



The work execution section is spawned by Work Generation, and defines the control logic for the operator to transport the item. It primarily acquires an operator, tells the operator to load the item from the queue, then unload it to the processor, then travel back to the queue. Finally, it releases the operator.

Water Breaks



The water breaks section manages when and how to perform water breaks. It acquires a "thirsty" operator. This is where you will use the resource's list features to only acquire an operator who is "thirsty", i.e. an operator who has traveled at least 100 meters since his last water break. Once acquired, you will tell the operator to travel to the water cooler and "drink", i.e. delay for 30 seconds. Then you will save off his travel distance (for use in figuring out when he will become thirsty again), and release the operator. Finally you will loop back and do the whole thing over again.

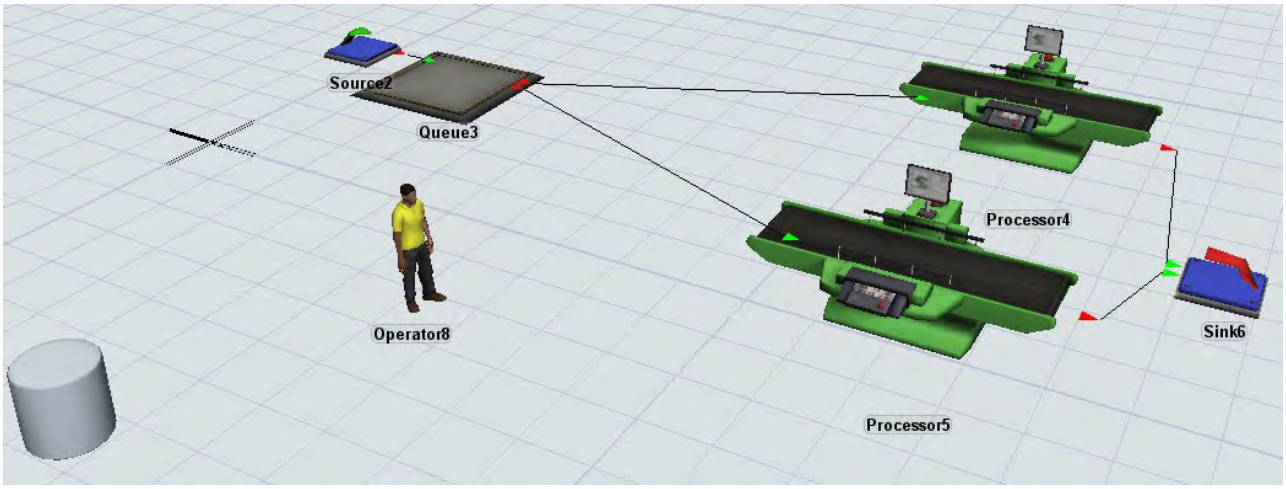
Perspectives

Process Flow logic is often implemented from "perspectives". This model, for example, is implemented from the perspective of the work that needs to be done. When items need to be moved, they acquire operators and tell the operators to move them. Even water breaks are implemented from this perspective. The "water breaker" acquires a thirsty operator and tells him to go get water. While you will use this method for this model, that doesn't mean it cannot be implemented from a different perspective. You may, for example, think of the operation through the eyes of the operator. He may look for items and transport them, and then, every time he finishes a transport operation, he checks if he's thirsty and branches to a separate "water break" logic. Each of these perspectives is perfectly valid. While you will use this tutorial model's "work perspective" because it lends itself well to demonstrating List and Resource features, the perspective you use will depend on what seems most intuitive and easy to implement. As you become more acquainted with Process Flow's features, you will become more confident in making this judgment call.

Build the Model Objects

First, you'll create the model's 3D objects.

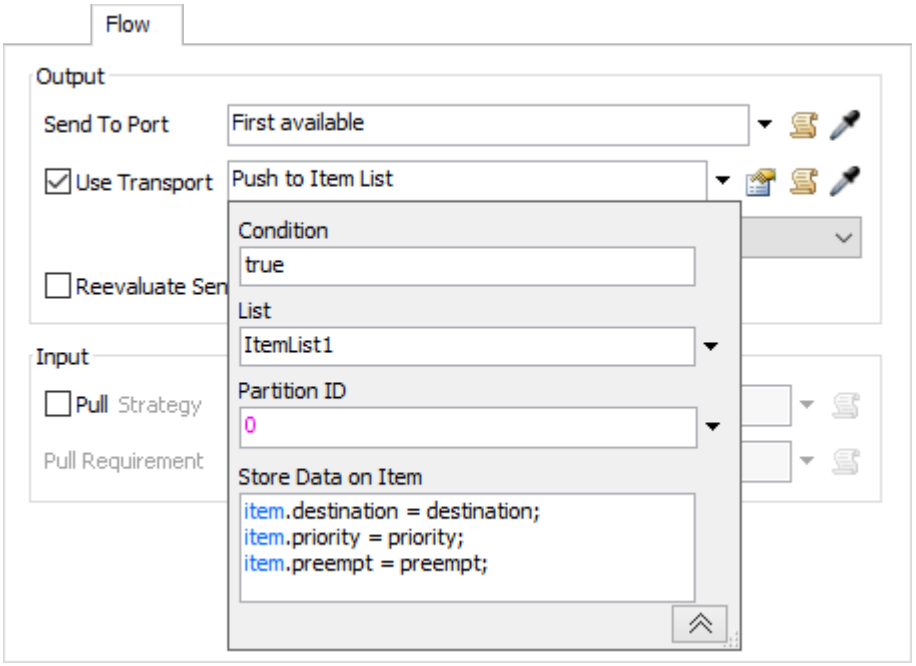
1. From the Library, drag out a source, a queue, an operator, two processors, a shape object, and a sink. Connect them as shown in the following image.



button and

2. Open the queue's properties. In the Flow tab, check Use Transport, then click on its select Use List > Push to Item List (No Task Sequence).

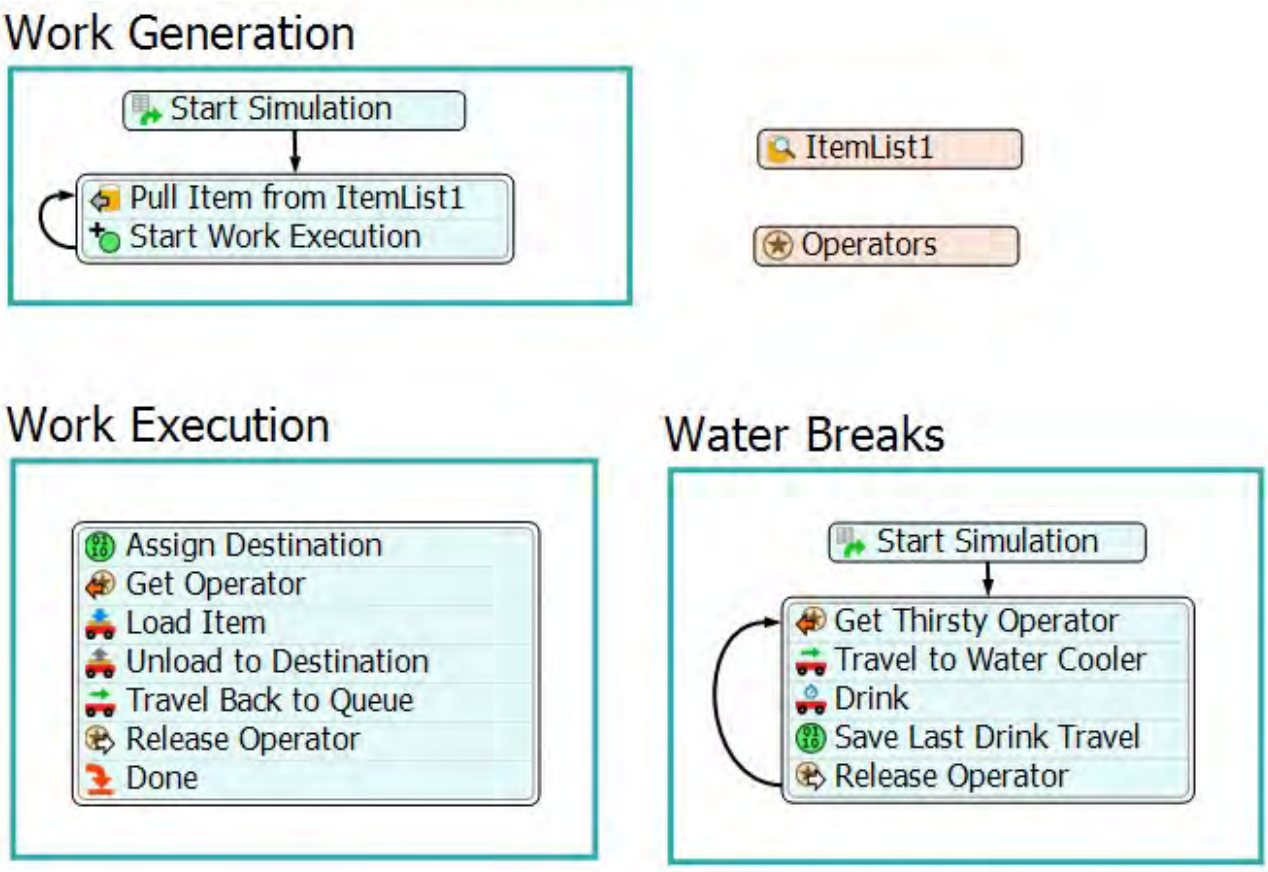
3. In the settings pop-up, under List, press the button and select Add New List.... This will add a new global list named ItemList1 and open its properties. You can close the properties. The Use Transport settings should be as follows.



The Push to Item List (No Task Sequence) option will push the item onto an item list. It will not create a transport task sequence (you want to create the task sequence ourselves in the Process Flow). Instead, it stores required data on the item's labels (the Store Data on Item code snippet), and pushes the item onto the list. Note that the destination (the processor that the item should to be sent to) is stored on the *destination* label of the item: `item.destination = destination`. This will be used in our Process Flow to figure out where to send the item.

Build the Process Flow

Next, you'll define the Process Flow. Create a new General Process Flow from the Toolbox. It will open up in a separate pane to the right. Recreate the layout shown in the following image.



Shared Assets

The Process Flow requires the following shared assets:

1. A List

   Add the list, name it *ItemList1*, and in its properties, under List, press the button. Then move the cursor over ItemList1 in the Toolbox and click on it (you can move the cursor over the Toolbox tab to select the tab). Choose Tools/GlobalLists/ItemList1.

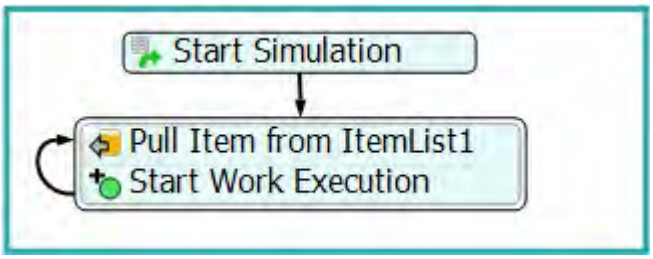
   This makes the list block into a "surrogate" for the global list ItemList1. Any push or pull operations on the list in the Process Flow will operate on the global list.

2. A Resource

   This resource will manage the operators. Add the resource, name it *Operators*, and in its properties, under Reference, press the button. Then click on the operator in the 3D model. This associates the resource with that operator.

   Next, under Count, enter *2*. This will make 2 operators available (an additional operator will be created the next time you reset the model).

Work Generation



The image shows the activities surrounded by a square Flow Chart display object. This does not affect simulation execution, but visually organizes activities. Add this object if desired and place subsequent activities inside it.

The work generation section requires the following activities.

1. A Schedule Source activity

   Add the activity and name it. Its default settings will create a single token at simulation start. This is exactly what you want, so you can leave its properties as their defaults.

2. A Pull From List activity

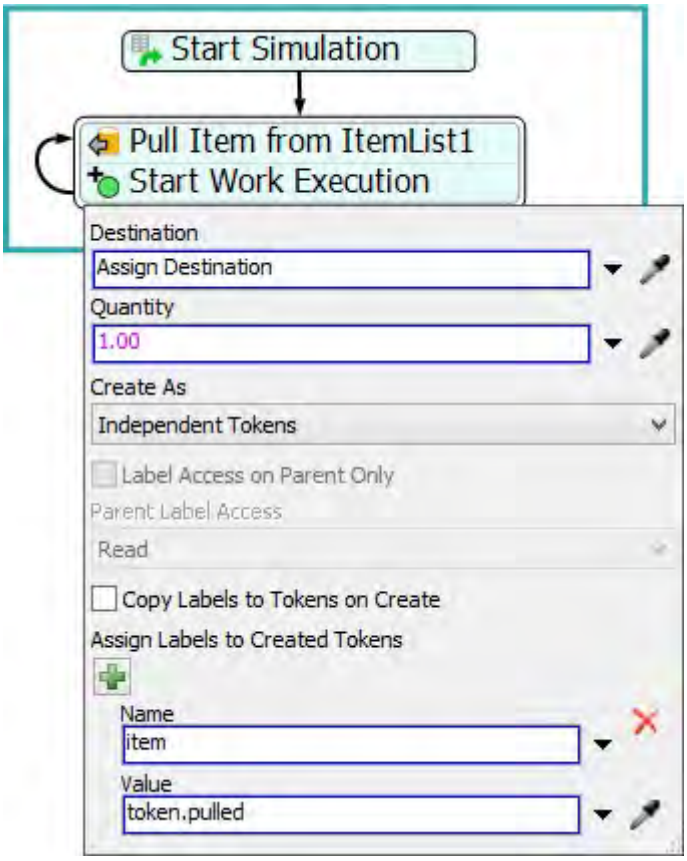
   Add the activity and name it. Then click on its icon and then click on ItemList1 to connect it to the list. You can leave all other properties as their default, but note that the result of the pull will be assigned to the token's *pulled* label (its Assign To field). You will use this in the next activity. The Pull From List activity will pull an item from the list and assign it to the token's *pulled* label. If an item is not on the list, a back order will be created and the activity will hold the token until the back order is fulfilled.

3. A Create Tokens activity

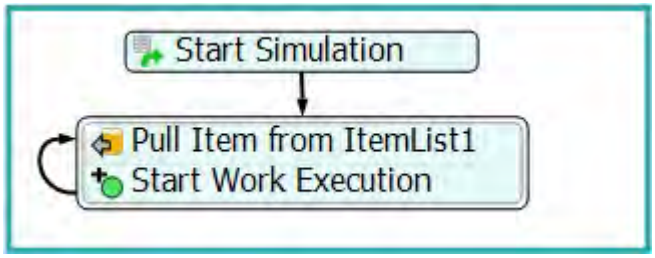
   Add the activity and name it. In this case you'll need to look ahead and create the first activity in the Work Execution section, because this activity will create a new token and send it to that activity. So, add an Assign Labels activity, to be defined later in Work Execution, then click on the Create Tokens activity's icon and then click on the activity you created.

   In the activity's properties, you won't need a reference back to the parent token, so for Create As select Independent Tokens.

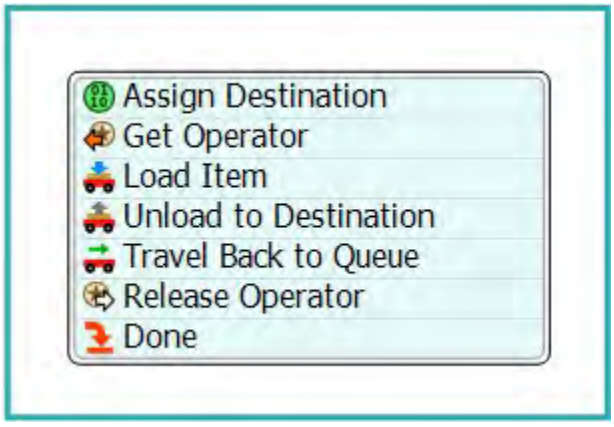
   Next, you want to copy the reference to the item from the parent token to the new token. Under Assign Labels to Created Tokens, remove any default label assignments, and then add one. Define it's Name as *item*, and for its Value enter *token.pulled*. This will assign a label named *item* on the new token to the parent token's *pulled* label value, which is the reference to the item that was pulled from the list in the previous activity.



Once you've created the activities in the Work Generation section, connect them with connectors as shown in the following image.



Work Execution

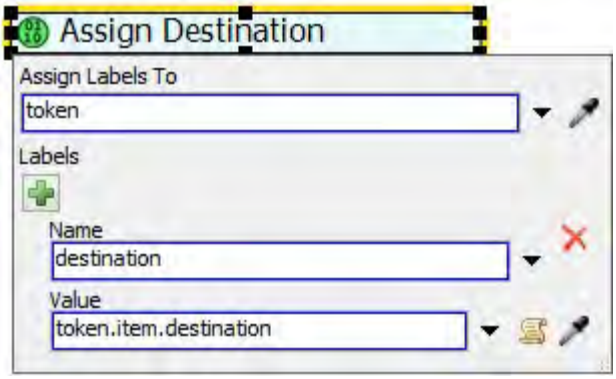


The image shows the activities surrounded by a square Flow Chart display object. Add this object if desired and place subsequent activities inside it.

The work execution section requires the following activities.

1. An Assign Labels activity

   You should have already created this activity when building the work generation section. Name it. In its properties, add a single label assignment, and give it the Name of *destination*, and for its Value, enter token.item.destination.



   This assignment first gets access to the item in the queue (remember you assigned the *item* label on the newly created token to reference the item in the queue). Then it gets that item's *destination* label. Remember when you defined the queue properties you noted that the destination was assigned on the item's *destination* label. Finally, it assigns that destination value to the token's *destination* label.

   To put it more simply, this label assignment is just pulling the destination label from the item to a label on the token. Having labels stored directly on the token can make later operations simpler to define.

2. An Acquire Resource activity

   Add the activity and name it. Click on the activity's icon and then click on the Operators resource to connect it to the resource.

   All other properties can be left as their defaults. Note that the acquired resource will be assigned to the token's *resource* label. You will use this label to reference the acquired operator.

3. A Load activity

   Add the activity and name it. In its properties, under Executer / Task Sequence, enter *item.resource*.

   For Station, click the button, and then click on the queue in the model. Choose Queue3 from the menu.

   Make sure the Item value is *token.item*

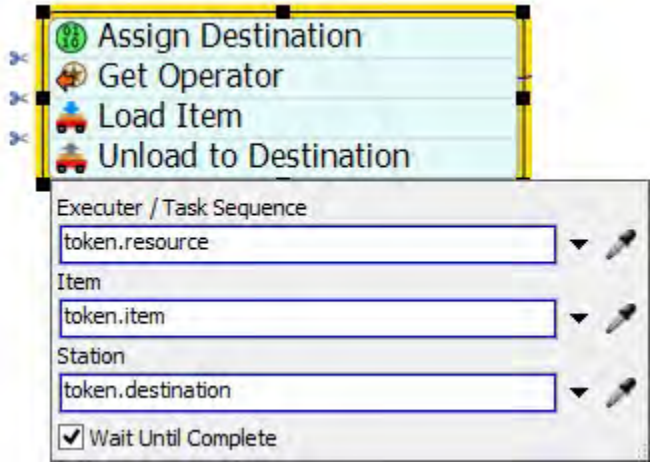
This load activity will tell the operator (the allocated object you assigned to the *resource* label) to load the item (the object assigned to the token's *item* label) from Queue3.

4.  An Unload activity

   Add the activity and name it. In its properties, under Executer / Task Sequence, enter *token.resource*.

   For Station, enter *token.destination*.

   Make sure the Item value is *token.item*

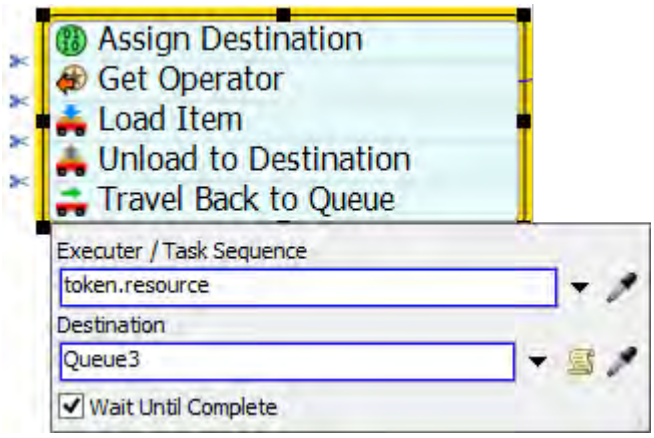


This unload activity will tell the operator (the allocated object you assigned to the *resource* label) to unload the item (the object assigned to the token's *item* label) to a destination processor (the object that was assigned originally to the item, and then pulled onto the token's destination label).

5.  A Travel activity

   Add the activity and name it. In its properties, under Executer / Task Sequence, enter *token.resource*.

   For Destination, click the button, and then click on the queue in the model. Choose Queue3 from the menu.



This travel activity will tell the operator (the allocated object you assigned to the *resource* label) to travel back to the queue.

6. A Release Resource activity

   Add the activity and name it. In its properties, make sure that the value for Resource(s) Assigned To is *token.resource*.

   This activity will release the operator (the allocated object you assigned to the token's *resource* label).

7. A Sink activity

   Add the activity and name it.

   This activity will finish and destroy the token.

Test the Model

Before moving on to the final section, you should be able to test this model now. Reset and run. Items should enter the queue, and then be transported by the two operators to one of the destinations. You should also see tokens moving through the Process Flow corresponding to events in the 3D model.

Operators Resource Configuration

Before you implement the water breaks section, you need to do some configuration on the Operators resource. Remember that the rule for water breaks was that only "thirsty" operators will take water breaks, i.e. operators who have traveled at least 100 meters since their last water break. You need to add fields to the Operators resource's internal list, to allow us to filter for "thirsty" operators.

1. Click on the Operators resource in the Process Flow view. In its properties, press the Advanced button. This will open the properties window for the resource's internal list.

2. In the Fields tab, press the button and select TaskExecuter > totalTravel.

This will add a field call totalTravel to the list. This represents the total travel distance of the operator on the list. Once the field is added, you can query its value when you are acquiring the operator.

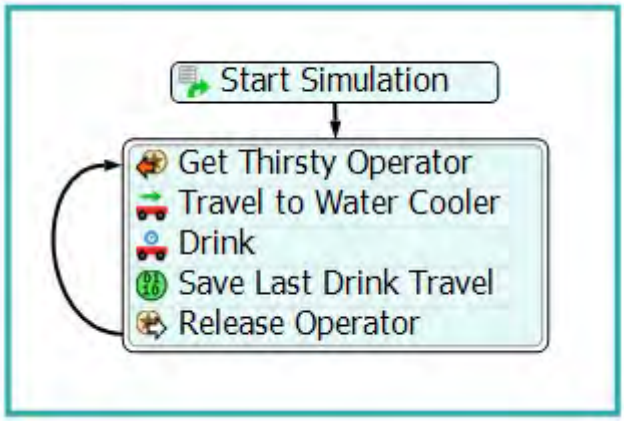
Notice that the expression for the field's value is `getvarnum(value, "totaltraveldist")`. This gets the variable named *totaltraveldist* on *value* (the operator on the list). You will use this same expression later when you want to store off the total travel distance after the operator is finished taking a water break.

3. In the Fields tab, press the  button again, and select Label. In the added Label Field, enter *lastDrinkTotalTravel*. This adds a field to the list corresponding to a label on the operator named *lastDrinkTotalTravel*. You will assign this label to the operator's total travel each time the operator finishes a water break.

4. Press OK to close the list properties.

Water Breaks

Now you'll build the final section, the logic for performing water breaks.



Like in the other sections, the activities are surrounded by a square Flow Chart display object. Add this object if desired and place subsequent activities inside it. The water breaks section requires the following activities.

1.  A Schedule Source activity

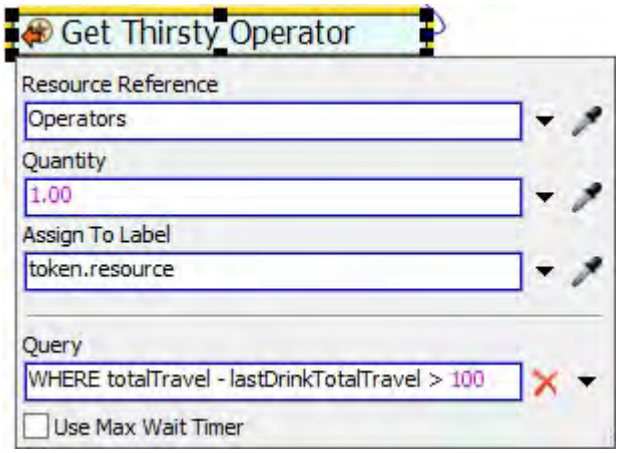
   Add the activity and name it. You can leave its properties as their defaults. This will create a single token when the simulation starts.

2.  An Acquire Resource activity

   Add the activity and name it. Click on the activity's icon and then click on the Operators resource to connect it to the resource.

   Under Query, enter `WHERE totalTravel - lastDrinkTotalTravel > 100`.

   This where clause tells the activity to only acquire an operator whose total travel since his last water break (`totalTravel - lastDrinkTotalTravel`) is greater than 100.



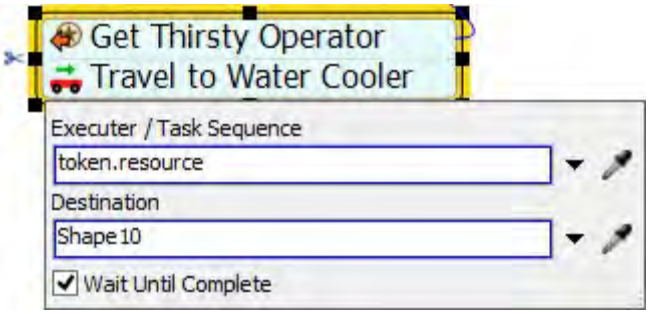
A Travel activity

3.

Add the activity and name it.

Under Executer / Task Sequence enter *token.resource*. button, and then click on the

For Destination press the cylinder object in the model.



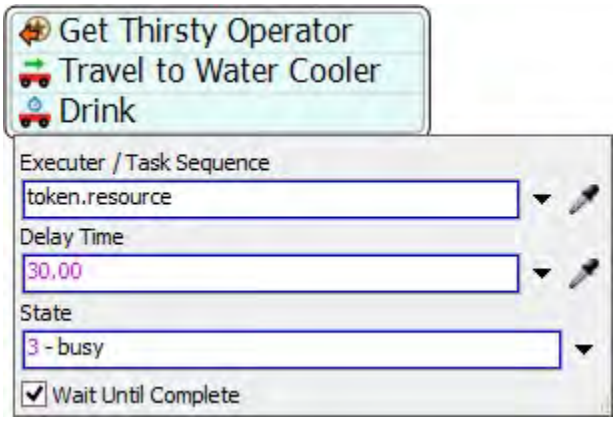
This travel activity will tell the operator (the allocated object you assigned to the *resource* label) to travel to the water cooler shape.

4. A Task Sequence Delay activity

Add the activity and name it.

Under Executer / Task Sequence enter *token.resource*. For

Delay Time enter *30*.



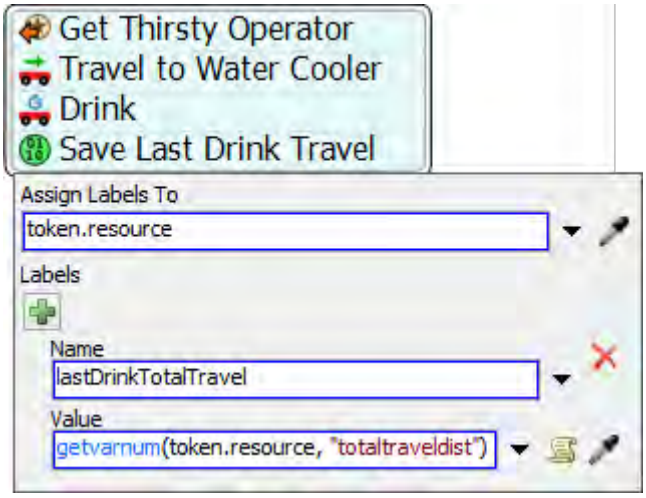
This activity will tell the operator to wait for 30 seconds while "drinking".

5. An Assign Labels activity

   This is where you will save off the operator's total travel distance to his *lastDrinkTotalTravel* label. Add the activity and name it.

   Under Assign To enter *token.resource*.

   Add a label assignment, and under Name, enter *lastDrinkTotalTravel*. For its Value enter getvarnum(token.resource, "totaltraveldist").



   This activity will assign a label named *lastDrinkTotalTravel* to the operator (the allocated object assigned to the token's *resource* label). The assigned value is the value of the variable named *totaltraveldist* on the operator (token.resource). Note that this expression is the same as the expression of the list's *totalTravel* field, expcept that in place of *value* (the value on a list), you reference an operator reference through the *resource* label on the token.

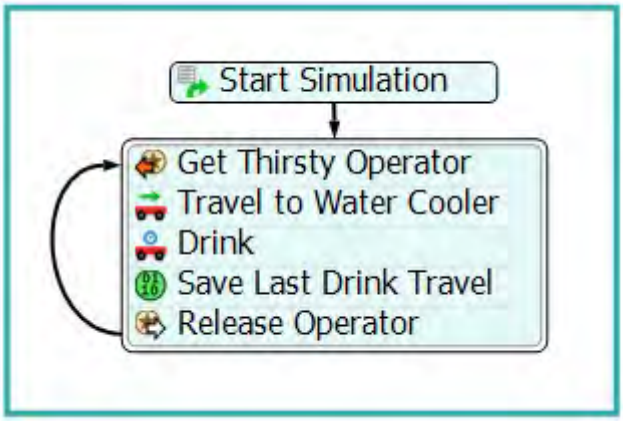
   When the operator is released and consequently pushed back on to the Operators resource's internal list, the *lastDrinkTotalTravel* field will be updated to the operator's new value.

6. A Release Resource activity

   Add the activity and name it.

   In its properties, make sure that Resource(s) Assigned To is *token.resource*.

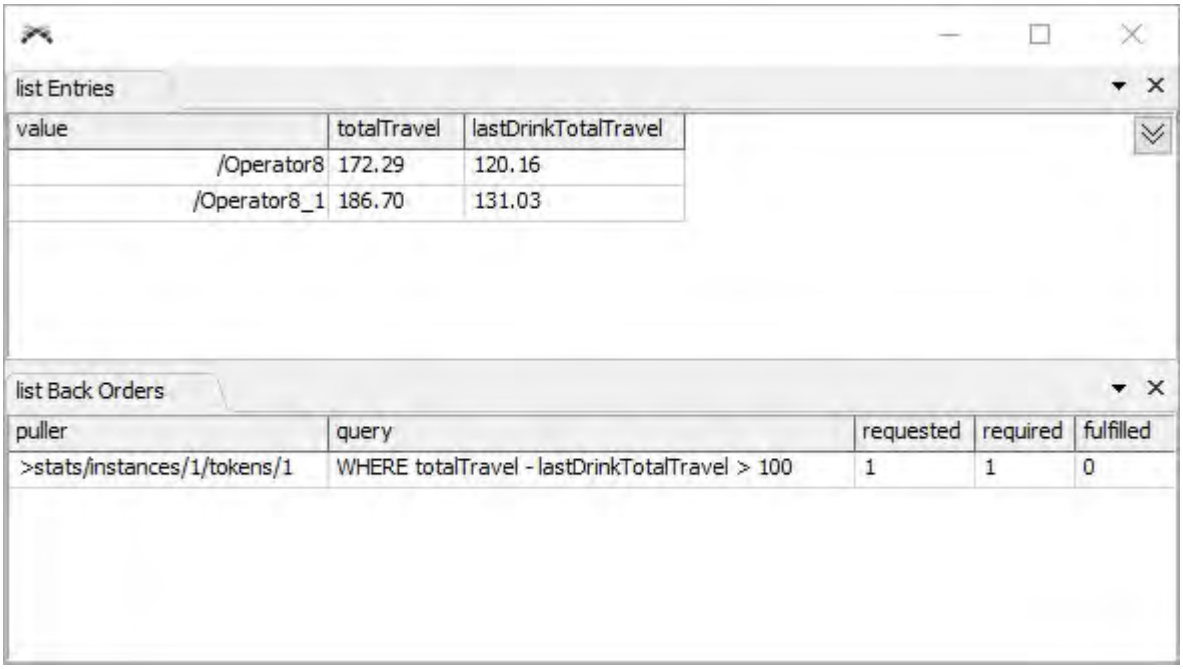
This activity will release the operator (the allocated object assigned to the token's *resource* label). Once you've added these activities, make sure they are connected as shown in the following image.



Run the Simulation

Your model should now be ready. Reset and run the simulation. Now when the operators travel 100 meters, they will travel to the water cooler and "drink" for thirty seconds.

If you want to see the list in more detail, you can view the Operators resource's live entries. While the model is running, click on the Operators resource, and in its properties, press View Entries. This will show a view of the operators that are currently on the list, and their respective *totalTravel* and *lastDrinkTotalTravel* values. You can also view back orders (outstanding acquire requests for the resource), by pressing View Back Orders in its properties.



For More Information

Resource Shared Asset

List Shared Asset

List Concepts

Adding and Connecting ProcessFlow activities

Linking a Process Flow to a FlexSim model

Tutorial - Kanban Labeler

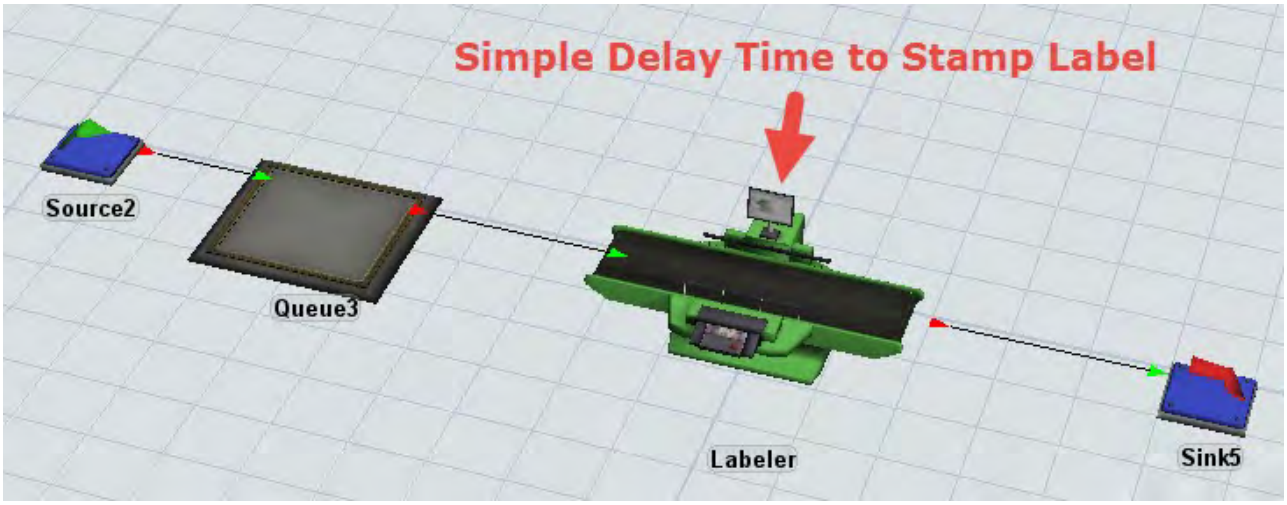
Process Flow includes features that allow you to easily customize logic on objects in the model. This tutorial will teach you how this works by implementing a kanban labeler, i.e. a processor that uses up material (labels that it stamps on items) as it processes items. Both material depletion and replenishment will be simulated.

This tutorial teaches the following Process Flow concepts.

- Using Fixed Resource Process Flow instances to define customized logic on objects in the model.
- Using Lists

The Problem

Quite often in manufacturing and other situations, machines use up material as they process products. Take for example a labeler machine that stamps sticker labels onto products. From a simulation perspective, you could implement this using a basic processor object. The core step is a delay associated with stamping the label onto the product.



However, using a simple processor to do this step does not simulate the machine's material usage, namely label paper, as it stamps products, nor the tasks required to replenish that material. Let's say the machine pulls labels off of a roll of sticker paper as it stamps products. Two rolls are stationed at the machine. When a roll is depleted, the machine automatically switches to the second roll, and a replenishment task is signaled to retrieve a new roll. The machine will pull from the second roll while the original roll is being replenished. If it happens that the second roll is depleted before a new roll is retrieved, the processor must stop until it has at least one roll. This scenario is a basic two-bin kanban system, where two "bins" (label rolls) are stored at the processing station, and depletion of one signals a replenishment to the station.

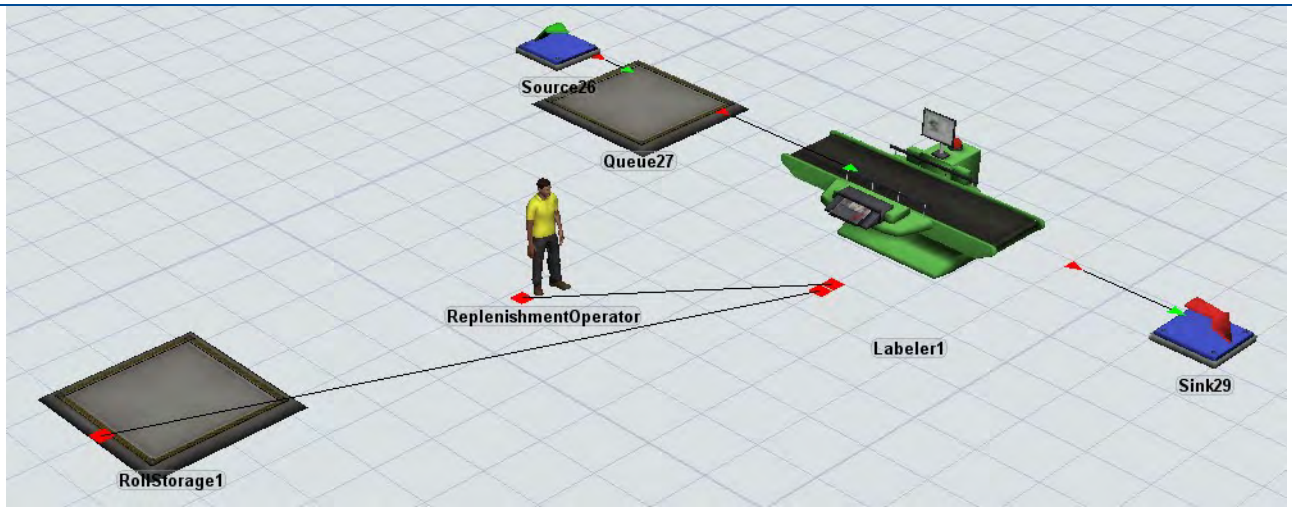
In standard FlexSim, you could simulate this as a combine operation. A combiner pulls the product from port 1, and then combines it with a label from port 2. However, this implementation would be very clunky. You'd have to split a roll into each label, and then do management of the two-bin-ness of the split roll and an extra roll. This would add a lot of extra objects to the 3D model that do not represent the real world system. The solution would feel like a work-around.

Process Flow makes simulating this scenario much easier. You can use a standard processor object, and then attach a Fixed Resource Process Flow to it, that listens for when the processor processes products, and performs the necessary kanban logic.

How It Will Work

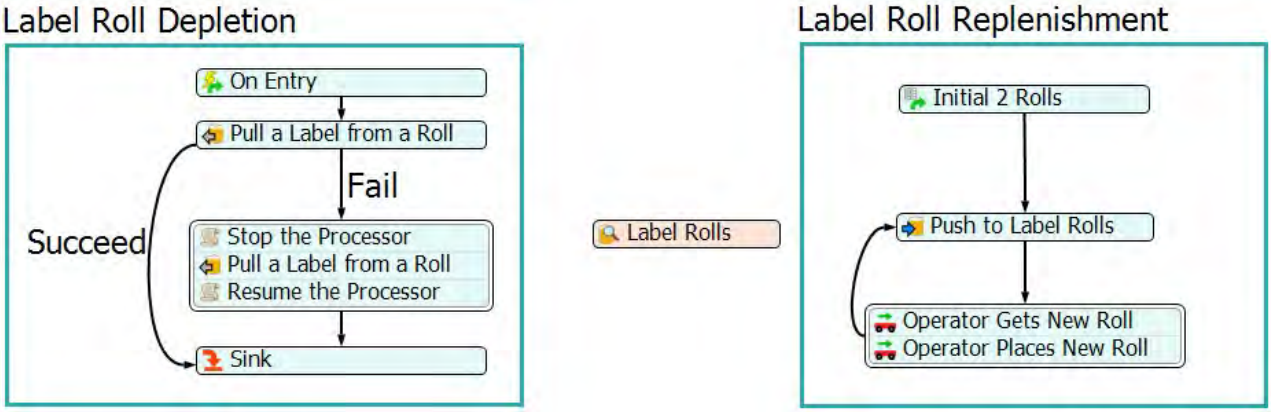
We will build this model in two phases. In the first phase, we will only simulate label depletion/replenishment logically, meaning there won't be physical 3D objects representing the label rolls. They'll only be represented as tokens in the Process Flow. In the second phase we will implement the physical label rolls.

The first phase of the model is shown in the image below.



The labeler machine is simulated using a processor object. There will also be an operator for replenishing the rolls, and a queue where the operator travels to pick up a new roll.

The Process Flow for phase 1 is shown in the following image



The Process Flow is separated into two sections: label roll depletion and label roll replenishment.

Label Roll Depletion

The label roll depletion section manages the processor's usage of material. When an item enters the processor, it will grab a label from a roll. If it is successful (there is at least one non-empty roll at the station), it's done. If, however, it can't immediately grab a roll, it must stop the processor, and then wait until it can grab a roll before the processor can resume.

Label Roll Replenishment

The label roll replenishment section manages keeping the machine supplied with label rolls. At the start of the simulation it will put 2 rolls at the machine. Then, each time a roll is depleted, it will tell the operator to travel to roll storage and bring a new roll to the machine. As mentioned before, in phase 1 we're not simulating physical rolls, so the operator isn't going to pick up an actual roll in the 3D model. He only travels to the storage location and then travels back to the machine *as-if* he were carrying a roll.

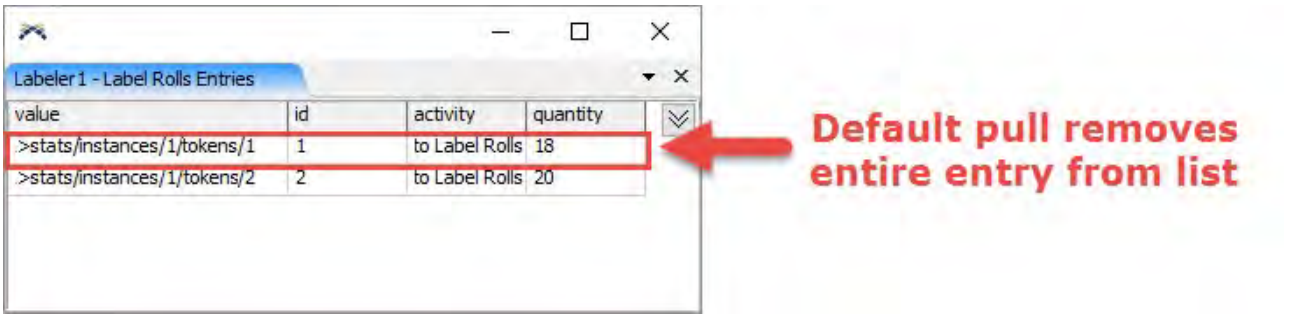
Key Concepts

Lists

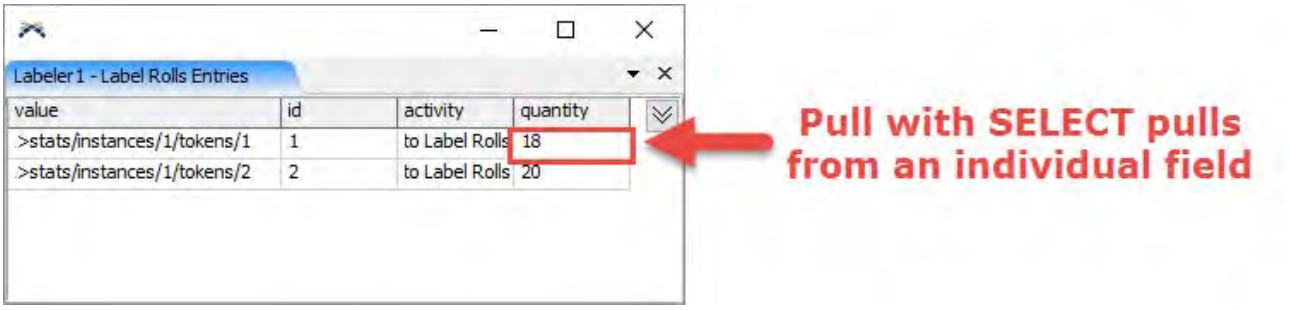
This model will use a list to represent the label rolls at a station. If you are new to lists you can refer to the Lists and Resources tutorial to learn more about them.

In the label roll replenishment section, when a roll is placed at the station, it is pushed onto the Label Rolls list. The label roll depletion section then pulls from the list of label rolls.

In this model, however, the pull operation is different than default pull operations. Usually a pull operation will pull/remove an entire entry from the list.



In this model, the list represents a list of label rolls, so a default pull would remove the full label roll. We don't want the label depletion to pull the full label roll. We only want to pull a single label from a roll on the list. Here we can use the SELECT clause in our pull query. Using the SELECT clause in a pull will cause the list to pull from a field on the list, instead of pulling the entire entry. The field value will be decremented with each pull operation. Once the field value reaches 0, the entry is removed from the list. In this model we will set a *quantity* field to be the total number of labels on the label roll. Each pull will decrement the field by 1. When the *quantity* field reaches 0, the label roll will be removed from the list.



See the List Functional Reference topic for more information on how this works.

Fixed Resource Process Flows and Instances

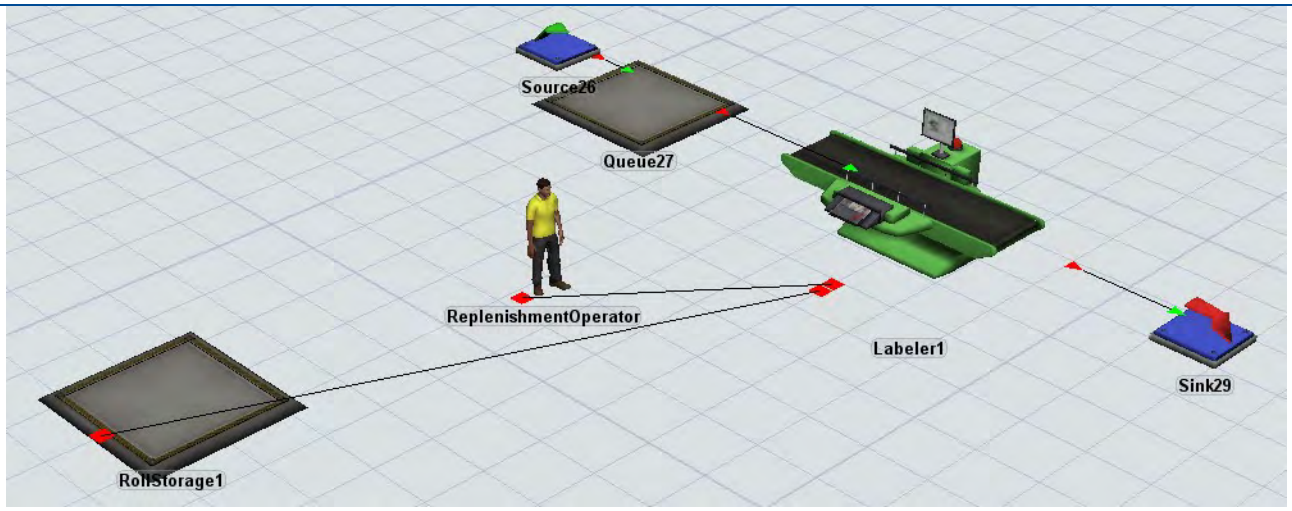
In this model we will use a Fixed Resource Process Flow. This is different than a General Process Flow in that it must be attached to objects in the model in order for any of its functionality to be executed. Any object that the Process Flow is attached to will become an instance of the Process Flow and will perform that ProcessFlow's logic. This means that, once we've defined the logic for the kanban labeler Process Flow, any object that we attach to the Process Flow will become its own kanban labeler.

Build the Model - Phase 1

Now let's build the model.

Model Layout

First we'll create the 3D objects. From a new model, create the layout as shown in the following image. This includes a source, two queues, a processor, a sink, and an operator.



Make sure you connect the processor's center port 1 to the operator, and center port 2 to the roll storage queue. We'll use these ports for referencing the objects in the Process Flow.

Process Flow

Now we'll implement the Process Flow. In the Toolbox press the button or click the Process Flow button from the main toolbar and select Process Flow > Fixed Resource > Blank. This will add a new, empty Fixed Resource Process Flow and open its modeling view.

Next, attach the Process Flow to the processor in the model. Click in the Process Flow view, and in Quick Properties under Attached Objects (instances), press the button and then click on the processor in the 3D view. This will make the processor into an instance of the Process Flow.

Add a List

Next we'll add a list to the Process Flow.

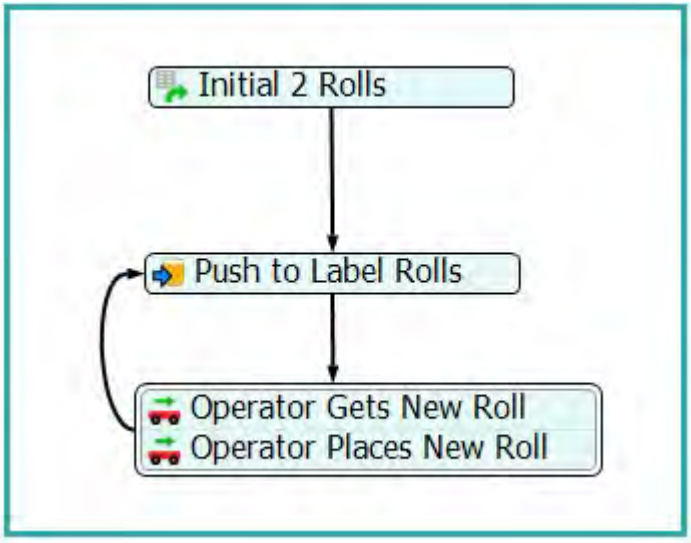
1. From the library drag a List into the Process Flow view and name it *Label Rolls*.
2. In its properties, under Type select Local.

   This means that each instance (labeler machine) will have its own set of label rolls at the station, instead of being shared globally.
3. In the list's properties, press Advanced.

   This will open the list's full properties window.
4. In the Fields tab, press the button and select Label.
5. For the Label Field enter *quantity*. Make sure that Dynamic is not checked.

The quantity field will track the number of sticker labels remaining on a roll.

Label Roll Replenishment



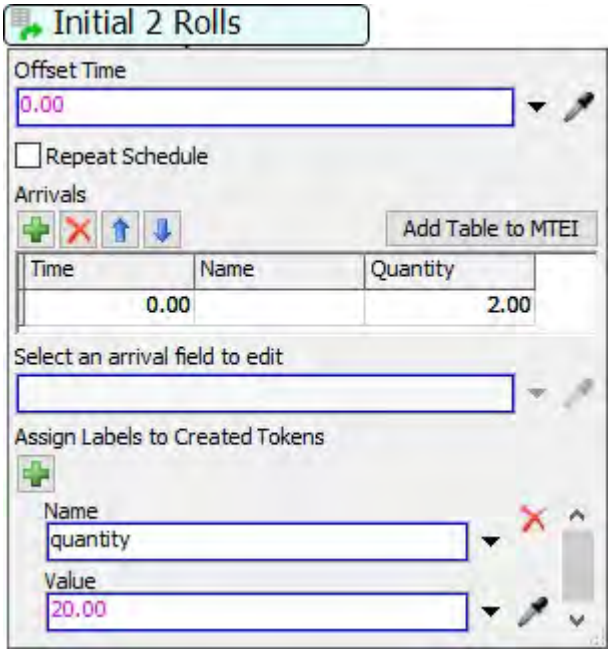
The label roll replenishment activities are surrounded by a square Flow Chart display object. This does not affect simulation execution, but visually organizes activities. Add this object if desired and place subsequent activities inside it.

Create and connect the activities in the label roll replenishment section. This includes:

1. A Schedule Soure activity

   Add the activity and name it *Initial 2 Rolls*. Go to its properties, and in its Arrivals table, in the first row under Quantity, enter *2*.

   Next, under Assign Label to Created Tokens, add a label with a Name of *quantity*, and a Value of *20*.



With these settings, the source will create 2 tokens when the simulation starts. Each token represents a single roll at the labeler machine, and will have a label called *quantity* with a value of 20. The quantity label represents the total number of sticker labels on a roll.

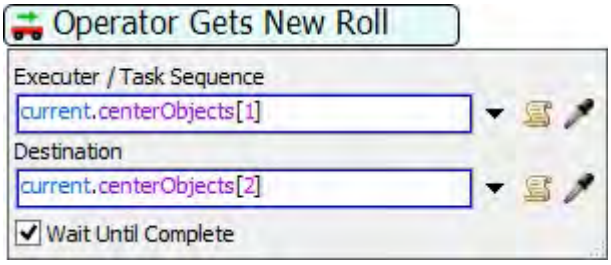
2. A Push to List activity

   Add the activity and name it *Push to Label Rolls*. Press its icon and then click on the Label Rolls list block.

   This will push the token (the label roll) onto the list of available label rolls at the processor. The token will remain in this activity until the label roll is pulled off of the list (by the label roll depletion section).

3. A Travel activity

   Add the activity and name it *Operator Gets New Roll*. In its properties, under Executer / Task Sequence enter `current.centerObjects[1]`, and under Destination enter `current.centerObjects[2]`.



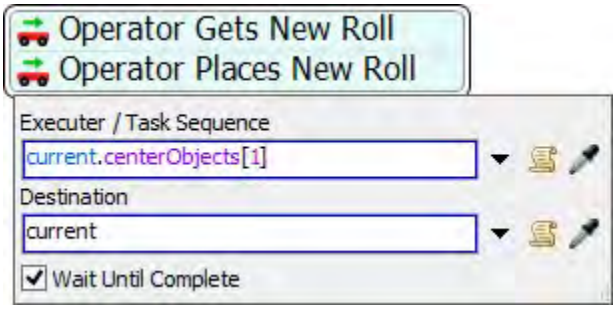
When using Process Flow instances, the `current` keyword is used as a reference to the instance object (in this case, the processor object). The expression `centerobject(current, 1)` gets a reference to the object connected to the processor's first center port, or the replenishment operator.

`current.centerObjects[2]` gets a reference to the object connected to the processor's second center port, or the roll storage queue.

These settings will cause tell the replenishment operator to travel to the roll storage queue.

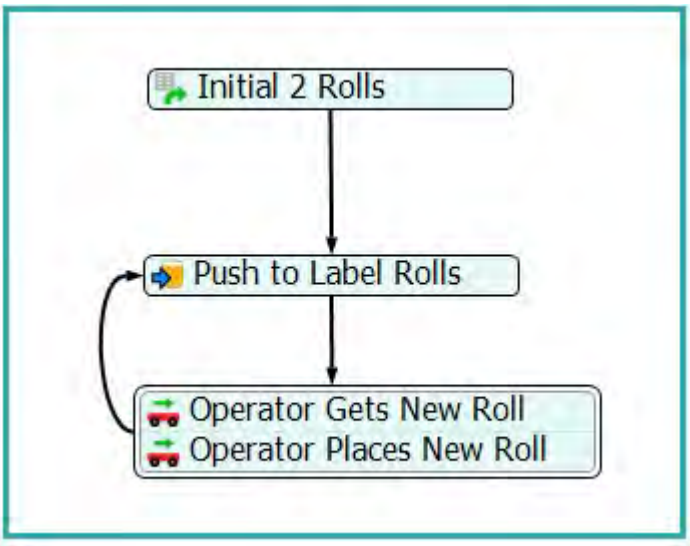
4. A second Travel activity

Add the activity and name it *Operator Places New Roll*. In its properties, under Executer / Task Sequence enter current.centerObjects[1], and under Destination enter current.

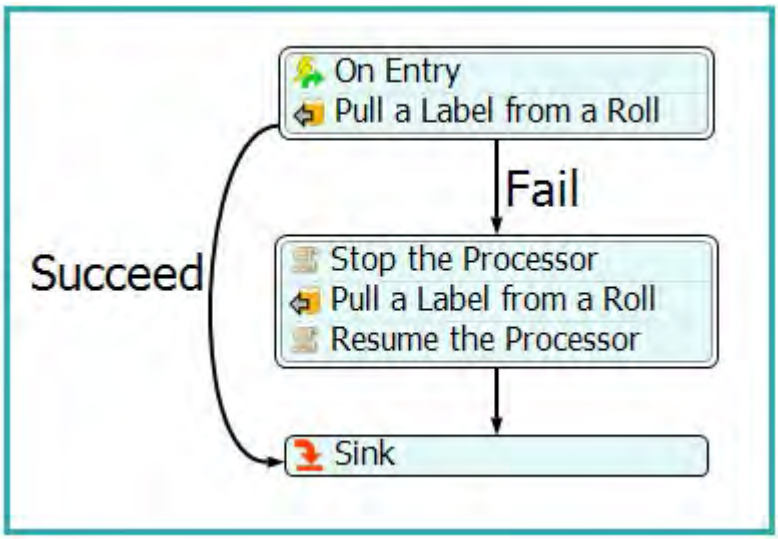


These settings will tell the replenishment operator to travel back to the labeler machine as-if he were carrying a label roll back to it.

Once you've added these activities, connect them together with connectors.



Label Roll Depletion

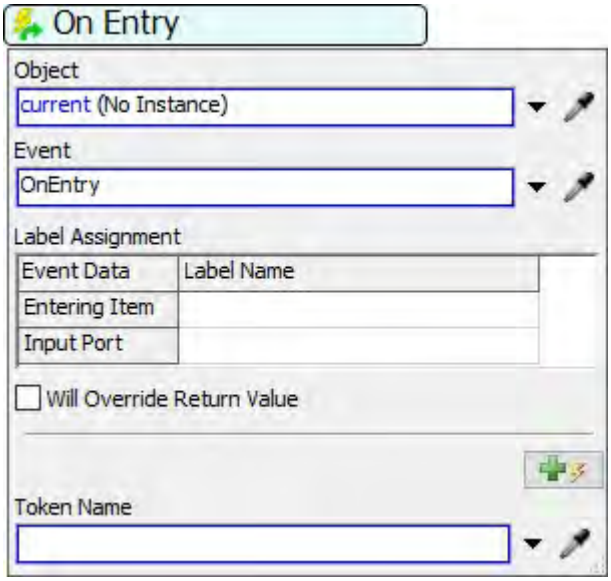


The label roll depletion activities are surrounded by a square Flow Chart display object. Add this object if desired and place subsequent activities inside it.

Create and connect the activities in the label roll depletion section. This includes:

1. An Event-Triggered Source activity

   Add the activity and name it *On Entry*. In its properties make sure Object has current in it, and then under Event, click the button and then click on the processor in the 3D model. Choose On Entry.

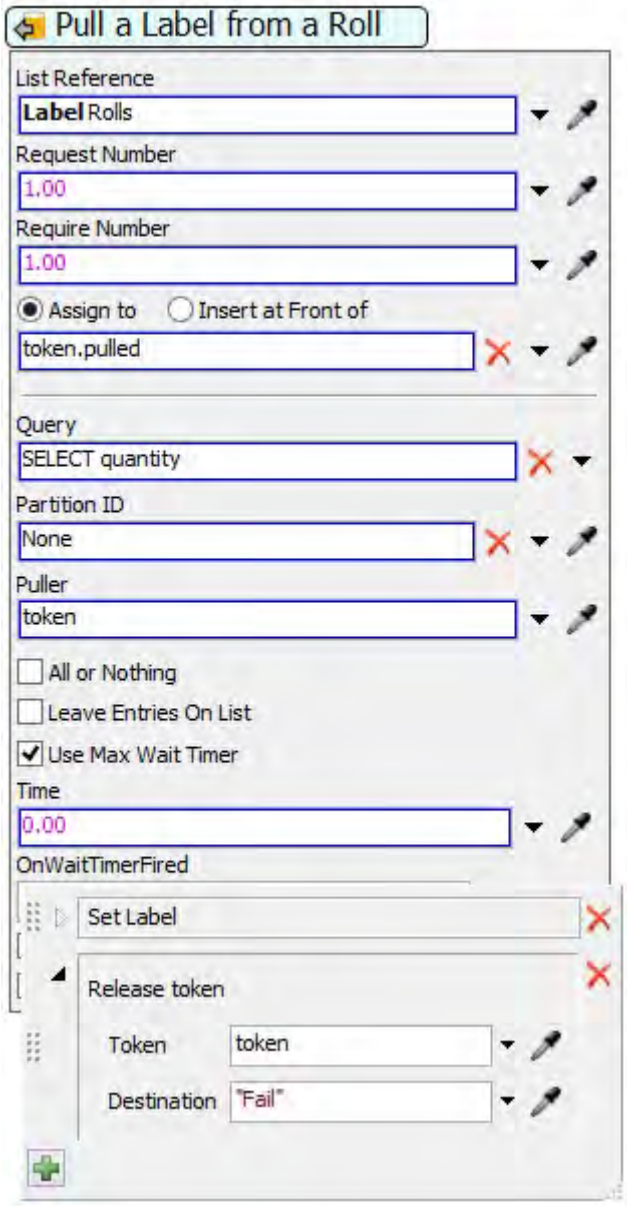


   These settings will make the activity create a new token each time an item enters the labeler machine.

2. A Pull from List activity

   Add the activity and name it *Pull a Label from a Roll*. Press its icon and then click on the Label Rolls list block.

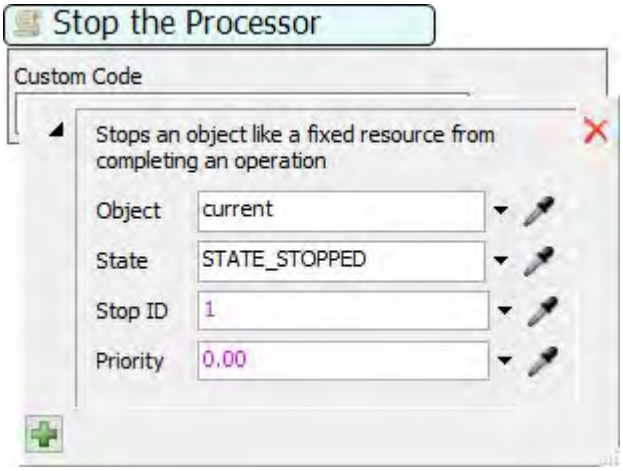
In the activity's properties under Query enter `SELECT quantity`.
Check Use Max Wait Timer, and in the trigger settings for OnWaitTimerFired, expand the Release token behavior. Under Destination enter `"Fail"`.



These settings will make the activity pull a single label sticker from the Label Rolls list by decrementing its *quantity* field by 1. Also, the Use Max Wait Timer settings make it so that, if the token cannot immediately pull from the label rolls list, i.e. there are no label rolls left, it will abort the operation and release the token through its "Fail" connector.

3. A Custom Code activity to stop the processor

Add the activity and name it *Stop the Processor*. In its properties press the button and select Control > Stop Object. Then under Object enter `current`.

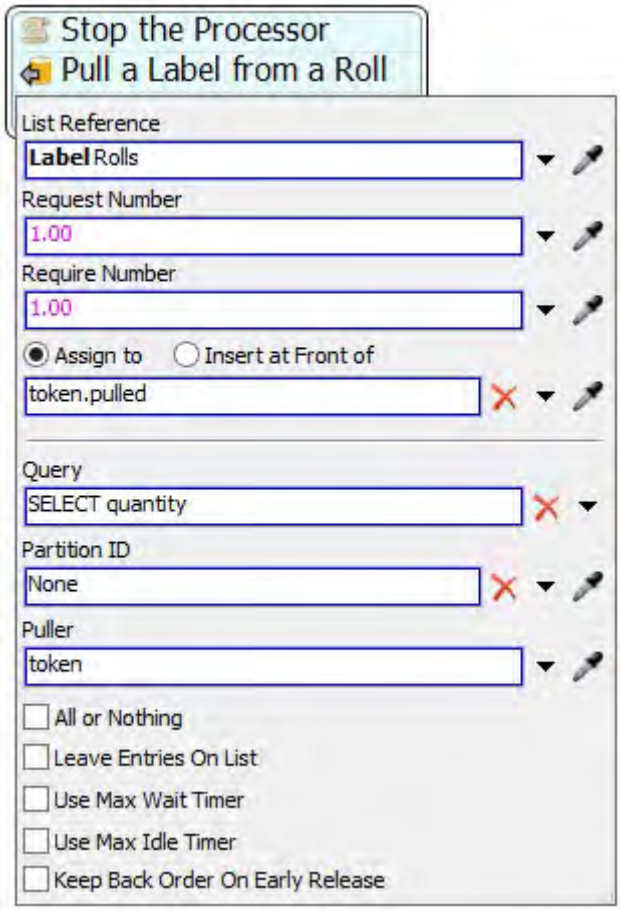


This will stop the labeler machine.

4. A second Pull from List activity

Add the activity and name it *Pull a Label from a Roll*. Press its icon and then click on the Label Rolls list block.

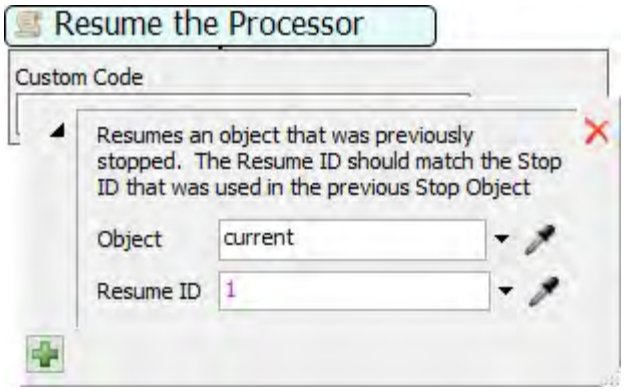
In the activity's properties under Query enter SELECT quantity.



These settings will make the activity pull a single label sticker from the Label Rolls list by decrementing its *quantity* field by 1. It will wait until there is a roll available to pull from.

5.  A second Custom Code activity to resume the processor

   Add the activity and name it *Resume the Processor*. In its properties press the ✚ button and select Control > Resume Object. Then under Object enter `current`.

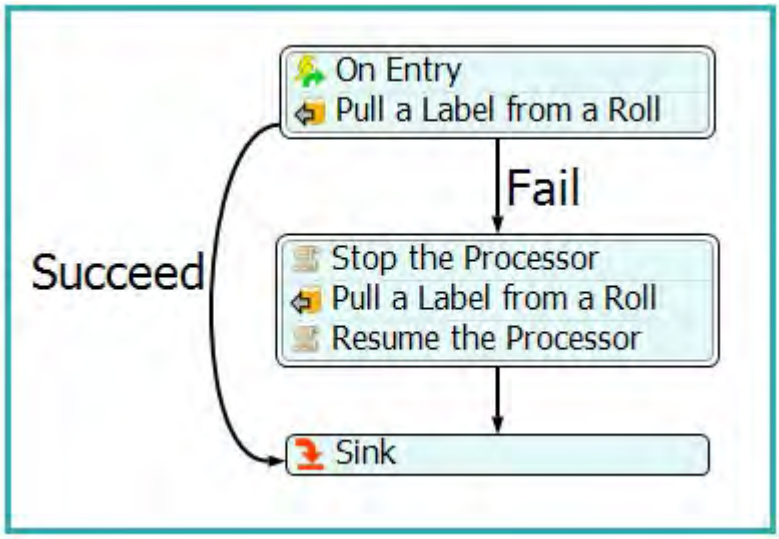


   This will resume the labeler machine.

6.  A Sink activity

   Add the activity. This will finish the token.

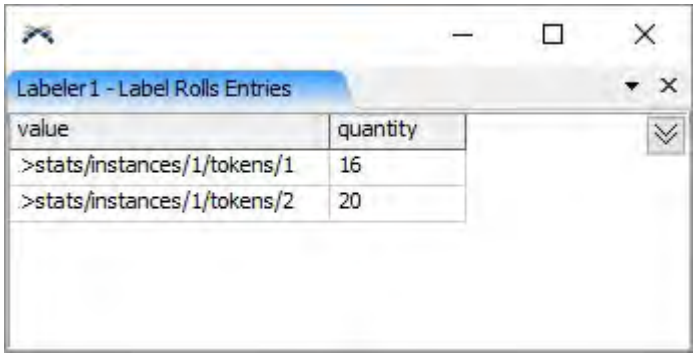
Once you've added these activities, connect them together with connectors.



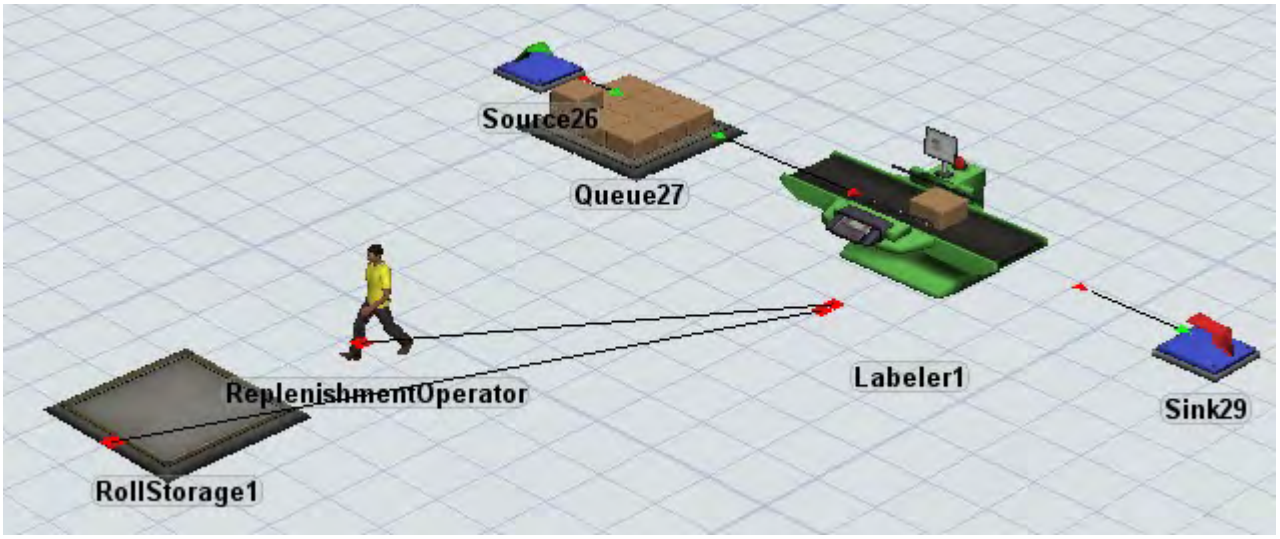
Make sure the Succeed connector is the first ranked connector from the pull from list activity (connect it first). This is important because if the pull succeeds, it will always send the token out its first connector. Make sure the Fail connector is named properly because we reference it by its name.

Run the Simulation

The model is now ready. Reset and run it. To view the Label Rolls list live as the model runs, click on the block and in its properties press View Entries...



You should see the quantity field decrease with each item that enters the processor. Once a roll's quantity reaches 0, it will be removed from the list, and the operator will travel to the roll storage queue and back.



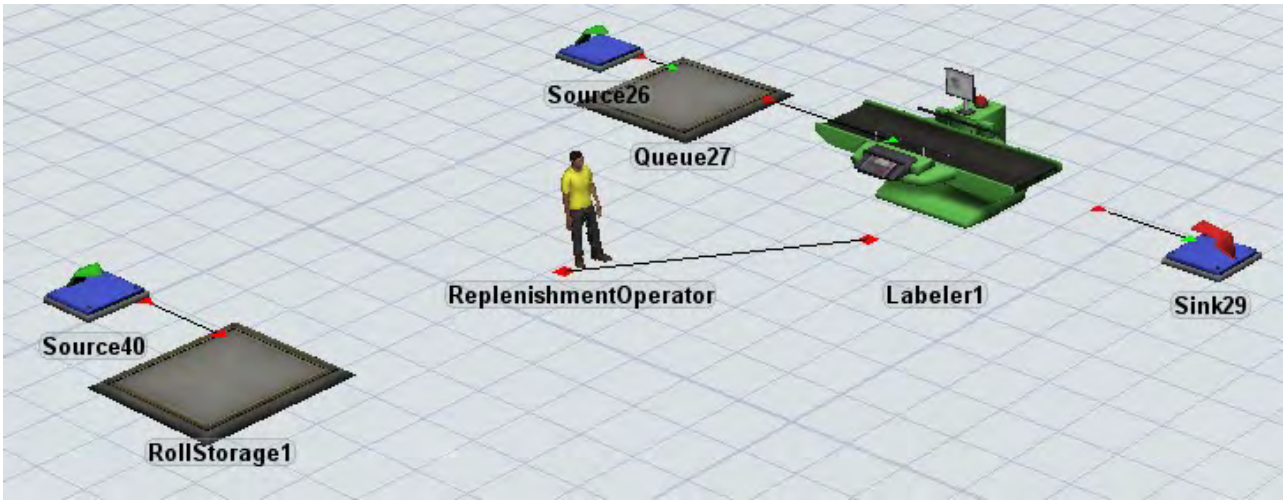
Now try to get the label rolls to be completely depleted, to see the processor stop. You can do this by either moving the roll storage queue really far away (so it takes a long time to retrieve the label roll), or by decreasing the operator's speed. When the label roll is completely depleted, the processor should stop working until a roll is replenished.

Phase 2 - Animation

Next we'll implement real rolls that are replenished to the labeler, and are animated as they are depleted.

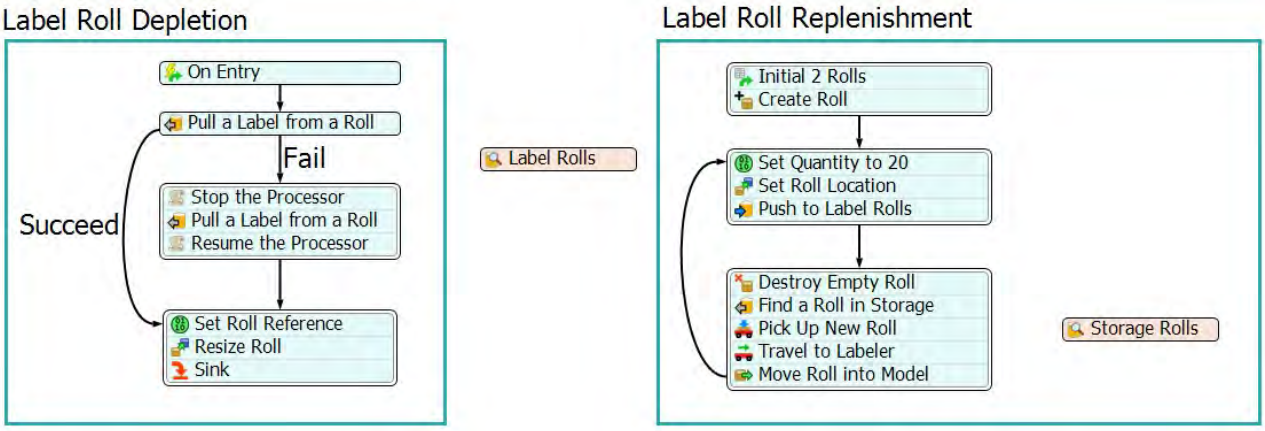
In phase 2 we will add a global list called *Storage Rolls*. This will represent the list of rolls that are available to be picked up from storage. The roll storage queue will receive items (rolls) from a source, and then push them onto the *Storage Rolls* list. Then the label roll replenishment section of the Process Flow will pull from that list to find an item to pick up. The item will be loaded by the operator and then placed beside the labeler.

Model Layout



The 3D layout for this model doesn't change much. Primarily we add a source to feed the roll storage queue, and then make the queue push to the *Roll Storage* list. Also, we won't use the labeler's 2nd center port to reference the roll storage queue. Instead we'll load the item pulled from the list.

Process Flow



The label roll replenishment section is where most of the changes will occur to the Process Flow. Here, the 2 initial tokens will create a roll in the 3D model. We'll also need to do a little more tracking of the quantity label, so that sizes can be properly updated. Also, when we replenish the roll, we need to pull a real roll from the queue, have the operator pick it up, and then place it next to the labeler machine.

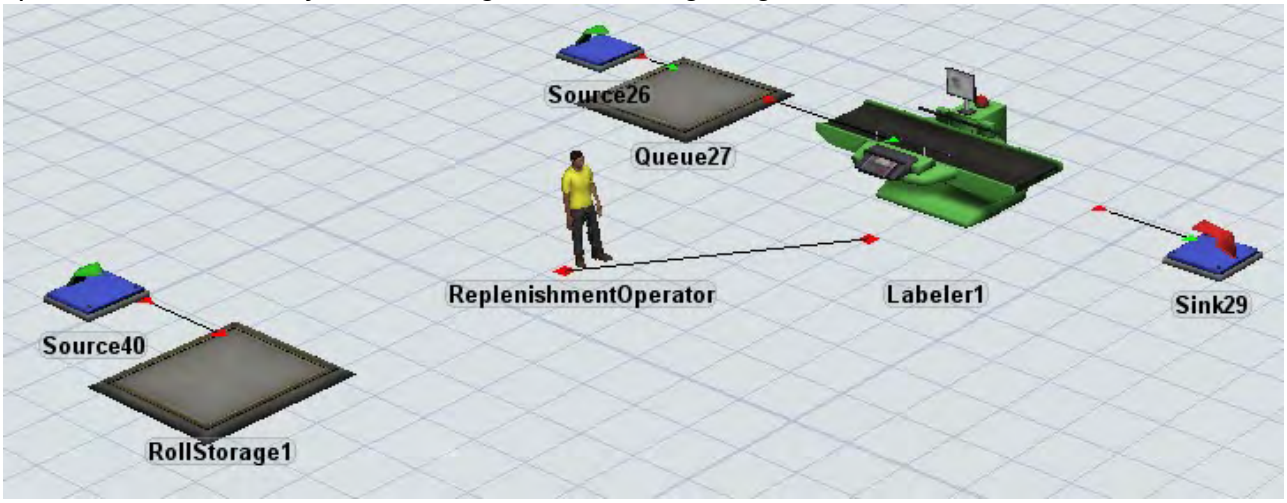
The label roll depletion section of the Process Flow will be adjusted. When it pulls a label from the label roll, it will update the roll's size based on its remaining quantity of labels. This will allow you to see the roll being visually depleted.

Build the Model - Phase 2

Now let's build the model.

3D Model Adjustments

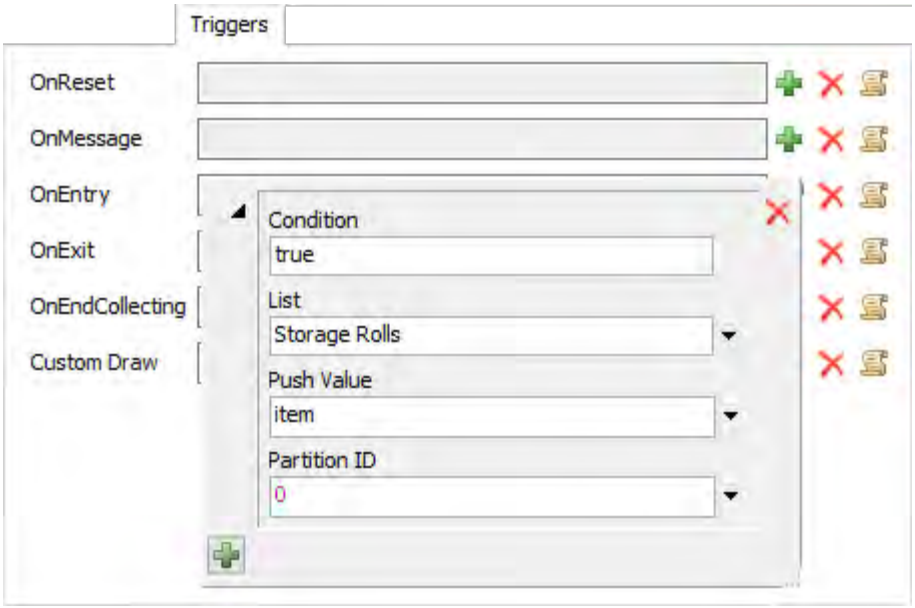
Update the model 3D layout according to the following image



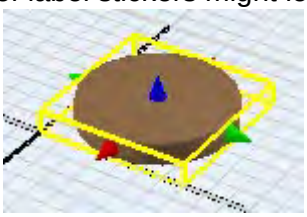
1. Add a source object and connect it to the roll storage queue. Open the source's properties, and under FlowItem Class, select Cylinder.
2. Disconnect the labeler's 2nd center port from the roll storage queue with a 'W' connect.
3. Open the RollStorage1 queue's properties. In its Triggers tab, under OnEntry press the button and select Lists > Push to List.
4. In the behavior settings popup, under List, press the button and select Add New List...

   This will add a new global list and open its properties (you could alternatively add the list through the Toolbox).

5. Name the list *Storage Rolls*
6. Go back to the queue's OnEntry behavior settings. Under List, press the button and select Storage Rolls.



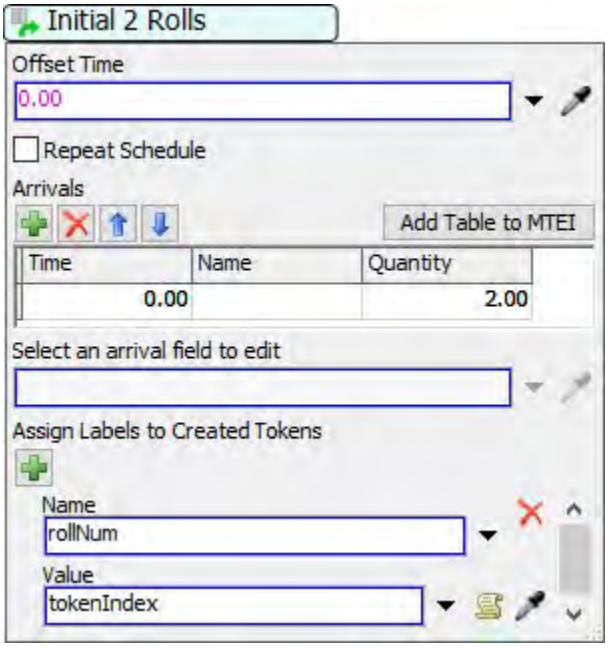
7. Open the FlowItem Bin. Select the Cylinder and decrease its z size so that it looks like what you think a roll of label stickers might look like. Changed its x and y size to 1.



Process Flow Adjustments

We'll need to change settings on some of the existing activities as follows.

1. In the Initial 2 Rolls activity's properties, under Assign Labels to Created Tokens remove the *quantity* label assignment (we'll assign the *quantity* label later on), and instead add a label with the Name of *rollNum* and for its Value press the ▼ button and select Token Index.



Here we are setting the rollNum label to the index of the token that will be created. The first token will get a value of 1, and the second a value of 2. We will use this in placing the roll in the correct location so the two rolls don't overlap in the 3D view.

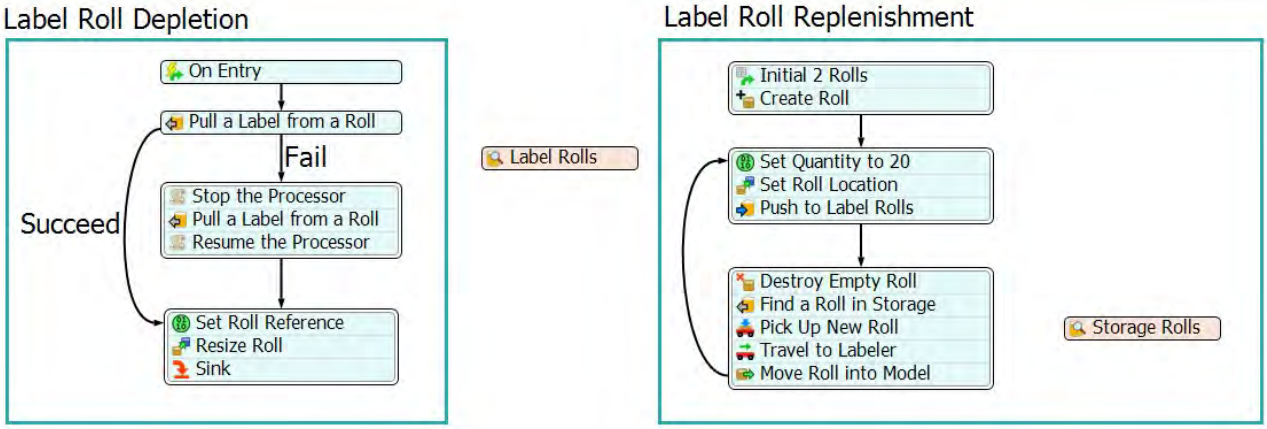
2. In the label roll replenishment section, delete the two operator travel activities. We'll recreate that whole section.

3. In the Label Rolls list properties, press Advanced to open the list's full properties window. In the Fields tab, beside the *quantity* field, check Dynamic.

Making this field dynamic will cause the list, in addition to decrementing the *quantity* field value on the list, to also decrement the *quantity* label value on the token itself. This means that we can know the remaining quantity of the roll so that we can properly update its size. See List Functional Reference for more information on how and why this works.

Process Flow Additions

Add and insert activities to the Process Flow according to following image.



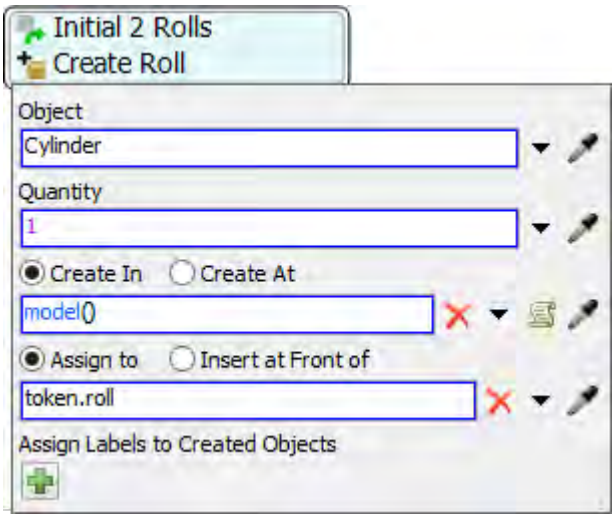
Label Roll Replenishment

We'll start with the label roll replenishment section. Required activity additions are as follows.

1.  A Create Object activity

   Add the activity, place it after *Initial 2 Rolls* and name it *Create Roll*.

Destination press the



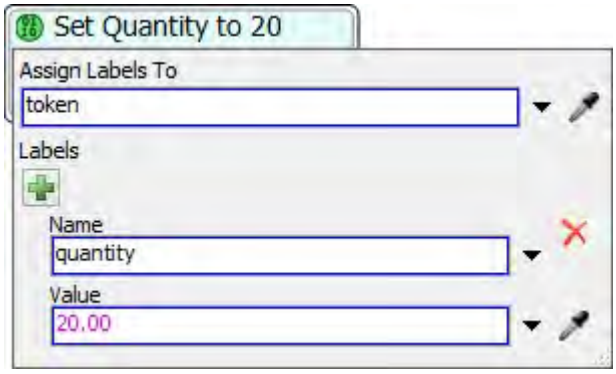
In its properties, under Object press the button and select Flowitems > Cylinder. Then for button and select Model. Finally, under Assign To enter *token.roll*.

These settings will create a cylinder object, place it in the model, and assign a label named *roll* on the token to be a reference to that object.

2. An Assign Labels activity

Add the activity, place it before *Push to Label Rolls* (add the activity then snap *Push to Label Rolls* below it) and name it *Set Quantity to 20*.

In its properties, add a label with a Name of *quantity* and a Value of *20*.



This will set the token's *quantity* label to 20. Since pull operations will decrement this label to 0, we need to reset it back to 20 each time before we push it back onto the list.

3. A Change Visual activity

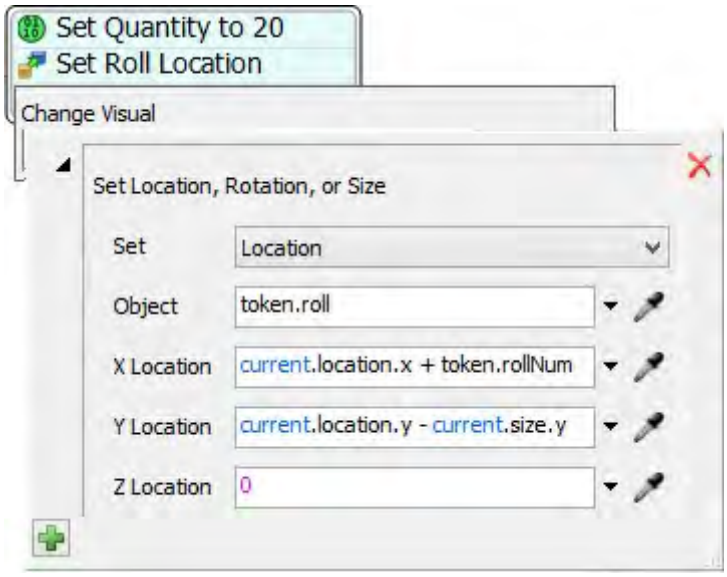
Add the activity, place it between *Set Quantity to 20* and *Push to Label Rolls* (you can add it in between the activities by double-clicking on the line between those activities and selecting the activity to add) and name it *Set Roll Location*.

In its properties, press the button and select Set Location, Rotation or Size.

In its properties, for Object enter token.roll.

For X Location enter `current.location.x + token.rollNum`.

For Y Location enter `current.location.y - current.size.y`. For Z Location enter `0`.

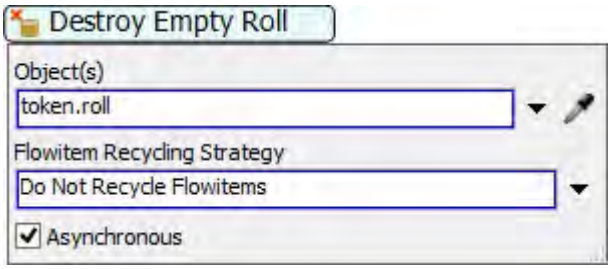


This will set the location of the roll object (the object referenced by the token's *roll* label) based on the location of the labeler machine (current). The first roll (rollNum 1) will be placed on the left, and the second (rollNum 2) will be placed on the right.

4. A Destroy Object activity

Add the activity, place it where the previous operator travel activities were, and name it *Destroy Empty Roll*.

In its properties, under Object(s) enter *token.roll*.



This will destroy the empty roll once it is depleted.

5. A List block

Add the block, place it beside *Destroy Empty Roll* and name it *Storage Rolls*.

In its properties, under List press the button, and then click on the *Storage Rolls* list in the Toolbox (you can hover the cursor over the Toolbox tab to activate that tab). Select Tools/GlobalLists/Storage Rolls Alternitavely you can select the Storage Rolls list from the Global List menu of the drop down.

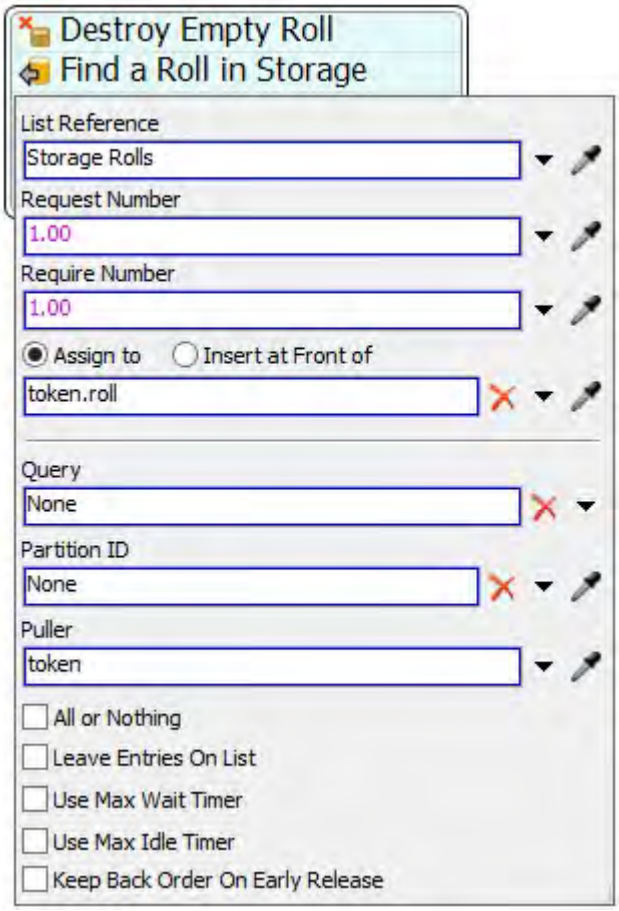
This will make the Process Flow list an alias for the global *Storage Rolls* list.

6.  A Pull from List activity

   Add the activity, place it after *Reset Roll Label* and name it *Find a Roll in Storage*.

   Press its  icon and then click on the *Storage Rolls* block.

   In its properties, under Assign To enter *token.roll*.

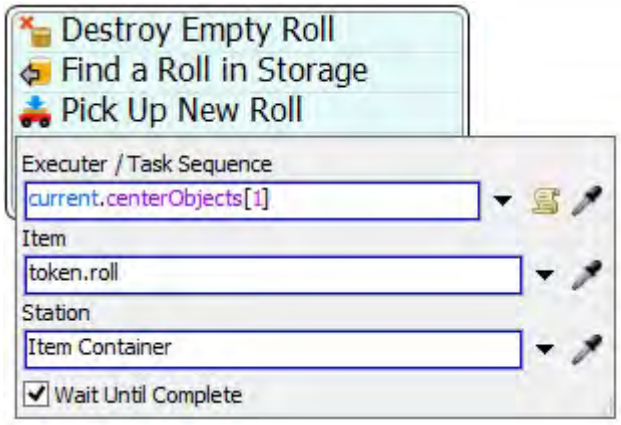


   These settings will make the activity pull a roll from the global *Storage Rolls* list and assign it to the token's *roll* label.

7.  A Load activity

   Add the activity, place it after *Find a Roll in Storage* and name it *Pick Up New Roll*.

   In its properties, under Executer / Task Sequence enter `current.centerObjects[1]`. Then for Item enter *token.roll*.

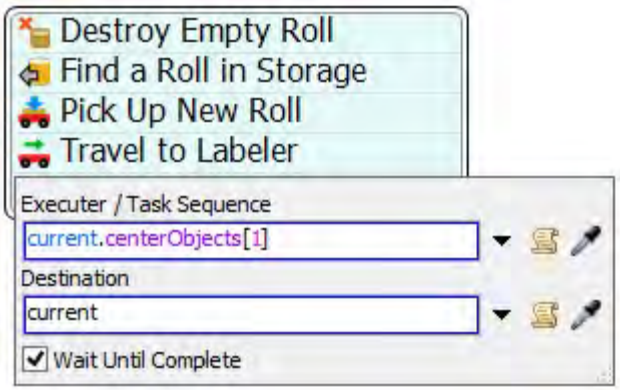


These settings will tell the operator (the object connected to the labeler's first center port) to load the roll (the object referenced by the token's *roll* label) from the item container (i.e. whatever queue the roll is currently in).

8.  A Travel activity

   Add the activity, place it after *Pick Up New Roll* and name it *Travel to Labeler*.

   In its properties, under Executer / Task Sequence enter `current.centerObjects[1]`. Then for Destination enter `current`.

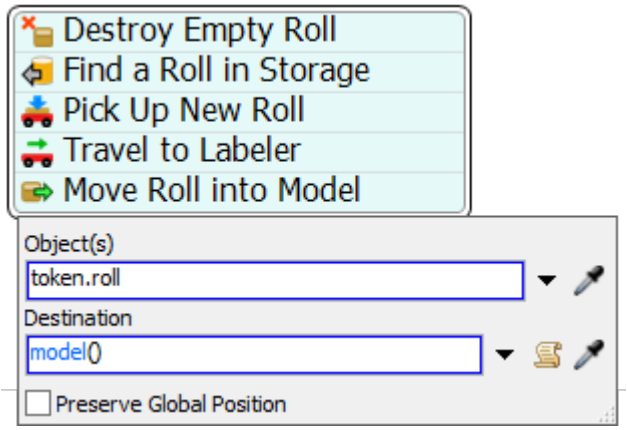


These settings will tell the operator (the object connected to the labeler's first center port) to travel back to the labeler (current).

9.  A Move Object activity

   Add the activity, place it after *Travel to Labeler* and name it *Move Roll into Model*.

   In its properties, under Object(s) enter *token.roll*, then under Destination enter `model()`.



This will move the roll object into the model. Note that we don't want to move the roll into the labeler itself. Since the labeler machine is a processor, moving the roll into it would cause the labeler to think it was an item that needed to be processed. Instead, we will put the roll into the model, and then set its location right next to the labeler.

Label Roll Depletion

Next we'll add activities to label roll depletion section. The primary change we make here is, when we pull a label from the label rolls list, we want to update the size of the roll based on how many labels are left. Required activity additions are as follows.

1. An Assign Labels activity
   Add the activity, place it before *Sink* (add the activity then snap *Sink* below it) and name it *Set Roll Reference*.

   This activity will add 3 labels to the token, as follows.

   1. A label with the Name *roll* and a value of `token.pulled.roll`

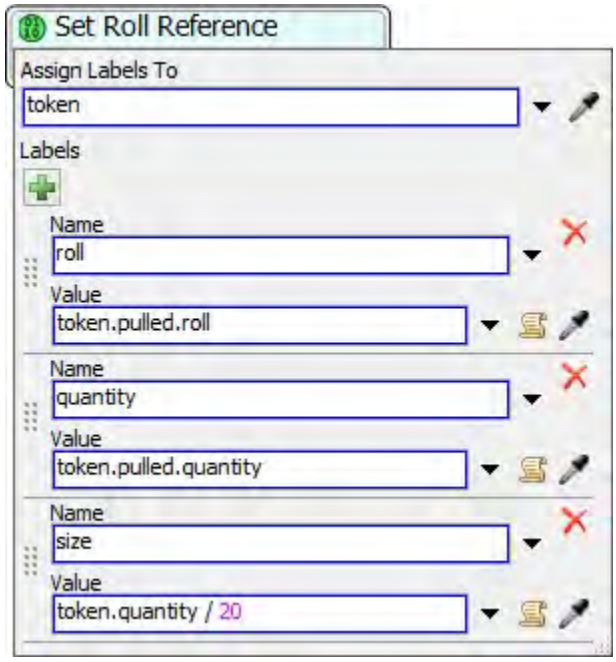
      This retrieves a reference to the 3D roll object from the roll token (the token in the label roll replenishment section that was pulled from the *Label Rolls* list) and assigns it locally to this token's *roll* label.

   2. A label with the Name *quantity* and a value of `token.pulled.quantity`

      This retrieves the current quantity label from the roll token (the token in the label roll replenishment section that was pulled from the *Label Rolls* list) and assigns it locally to this token's *quantity* label.

   3. A label with the Name *size* and a value of `token.quantity / 20`

      This sets the new size for the roll, which is the just-assigned *quantity* label divided by 20 (the max quantity of a label roll). This means a roll with 16 labels left will be assigned a size of 16 / 20 or 0.8.



 A Chan ge Visual activity

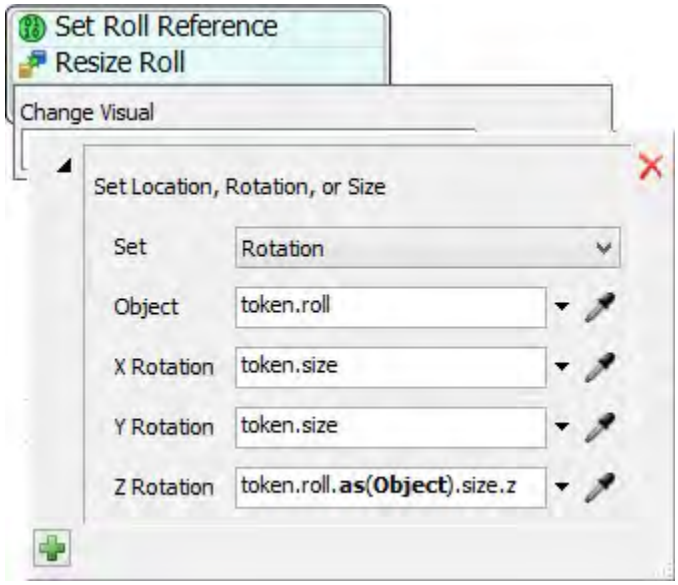
2.

Add the activity, place it between *Set Roll Reference* and *Sink* (you can add it in between the activities by double-clicking on the line between those activities and selecting the activity to add) and name it *Resize Roll*.

In its properties, press the button and select Set Location, Rotation or Size.

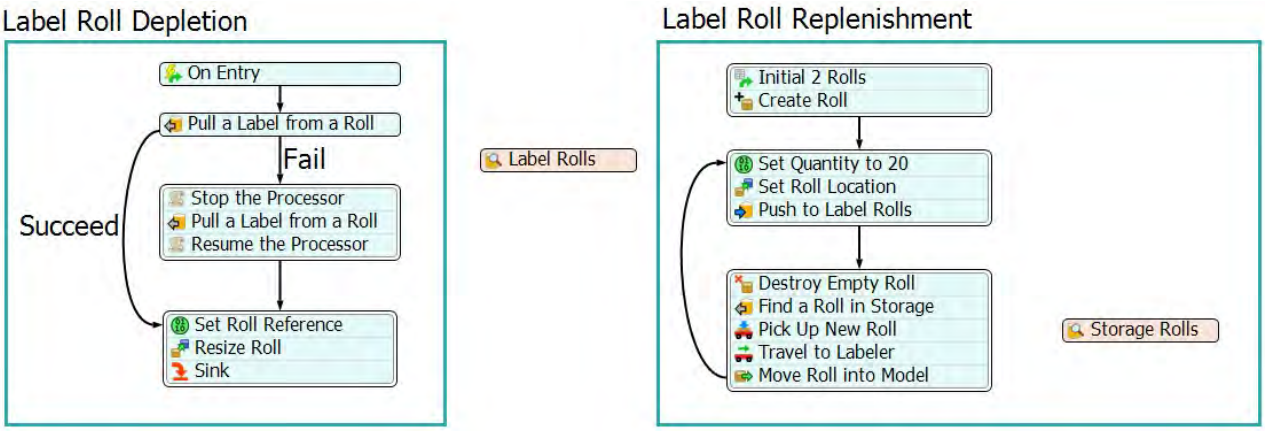
For Set select Size. For Object enter `token.roll`. For both X Size and Y Size enter `token.size`.

For Z Size enter `token.roll.as(Object).size.z`.



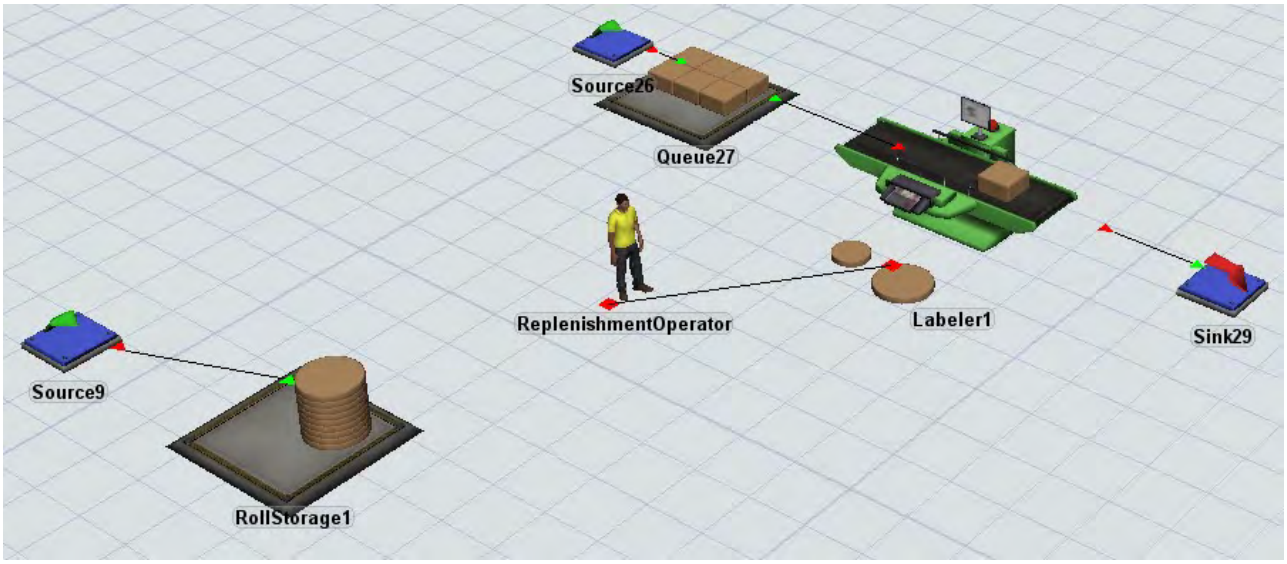
This will set the size of the roll (the object referenced by the token's *roll* label. The x and y size will be set to the token's size label, set previously. The z size will be set to whatever it was before (it will not be changed).

Once we've added all the new activities, make sure they are connected properly.



Run the Simulation

Now the model is ready. Reset and run. You should see two rolls appear at the labeler machine. As items are processed, one of the rolls should shrink. When it is completely depleted, it will be destroyed and a new roll will be retrieved from the roll storage queue.



Troubleshooting

With the current setup it can be difficult to see tokens moving through the Process Flow. This is because, for the most part, everything happens immediately at certain events in the model (mainly when an item enters the labeler). To better see the tokens moving through the Process Flow, you can place "breathe" points in the Process Flow by inserting zero-second delays. For example, you might put a zero-second delay after the *On Entry* source. Then stop the model and step forward until an item enters the labeler. Then you can step through the Process Flow and see the token moving through it.

Adding More Labelers

At this point it is pretty easy to add another labeler object. Just add another processor to the model, then attach the Process Flow to that object by clicking in a blank area in the Process Flow and in Quick Properties under Attached Objects (instances) press the button and then click on the newly added processor in the 3D view. Make sure the processor has a center port connection to an operator, and you're done. The processor will become a labeler machine.

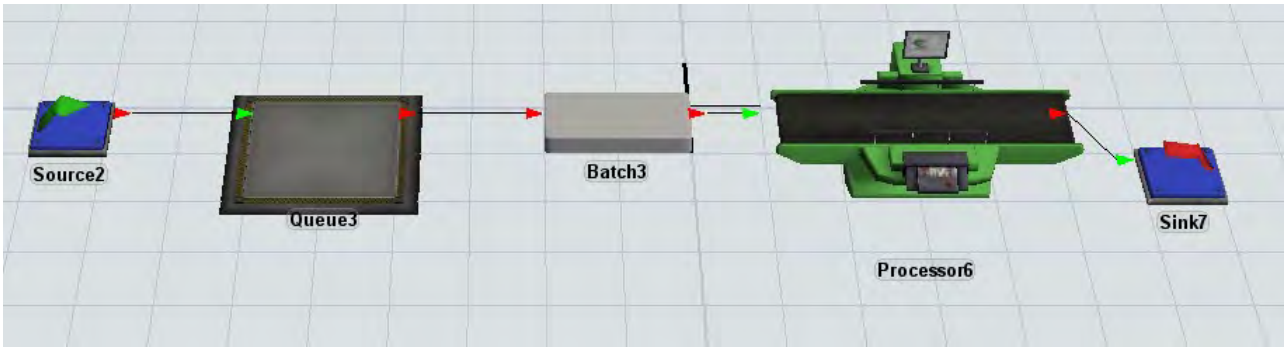
When you add an additional labeler instance to the Process Flow, the Process Flow view will show tokens for all instances (you may for example see 4 tokens in the label roll replenishment section instead of 2, namely 2 for each labeler machine). You can see a token's instance by clicking on the token in the Process Flow view. The instance (the labeler machine associated with that specific token) will be shown in Quick Properties. Also, if you only want to see the tokens associated with a specific instance, you can click in a blank area of the Process Flow view, and in Quick Properties under Attached Objects (instances) select the instance you want from the list, and then press Open Process Flow View. This will open a Process Flow window that only shows the tokens associated with that specific machine.

Tutorial - Custom Fixed Resource Process Flows

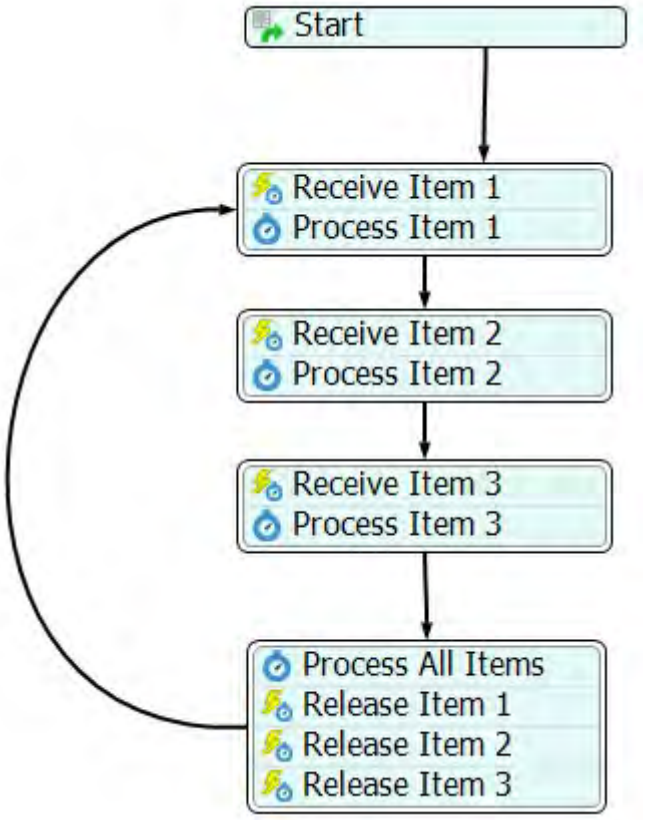
Process Flow makes it easy to define customized fixed resources, which is one of the purposes of the Fixed Resource Process Flow. This tutorial will teach you how to implement a customized fixed resource that batches and processes three items at a time.

Model Description

The following image shows the 3D setup for this model.



We will implement the logic for the Batch3 object. Batch3 will perform the following sequence:



Batch3 will sequentially receive 3 items, processing them each for 5 seconds. Then it will process all 3 items for an additional 10 seconds before releasing them to the next object. Once all items have exited, it will start over on the next batch.

Concepts Covered

This tutorial will cover the following concepts:

- Creating a Fixed Resource Process Flow
- Creating and using the Schedule Source, Wait For Event, and Delay activities.
- Receiving and Releasing items in a BasicFR using Process Flow.

Build the Batching Processor

1. Create the Fixed Resource Process Flow - Starting with a new model, go to the Toolbox, and click the button and select Process Flow > Fixed Resource. This will add a Fixed Resource Process Flow. Name it *Batch3*. Click back in the 3D view. You should now see in the Drag-Drop Library a new Process Flow category with the Batch3 Process Flow in it. This is an object that, when added to the model, will create a new instance of the Batch3 Process Flow. Each instance will execute the Batch3 Process Flow's logic.

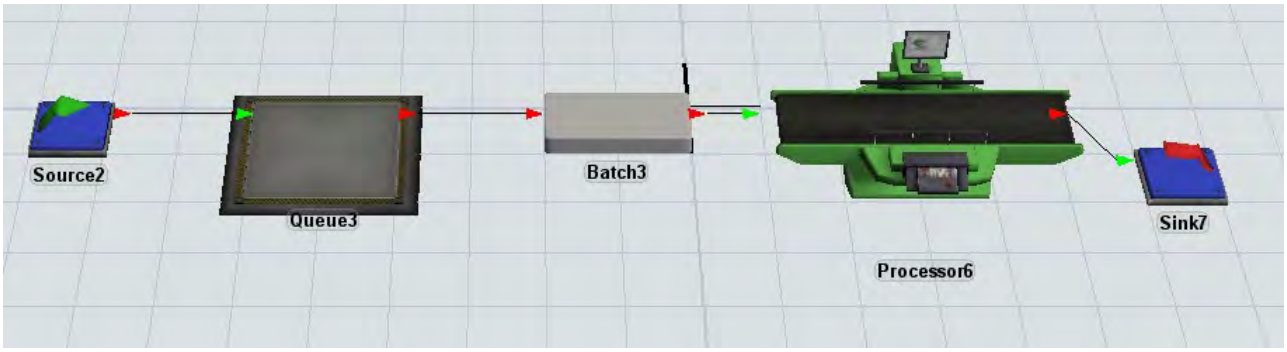
Using the BasicFR

The Batch3 instances that you create are, by default, instances of the BasicFR class. The BasicFR is a "blank slate" fixed resource. The BasicFR can receive items through its input ports and sends

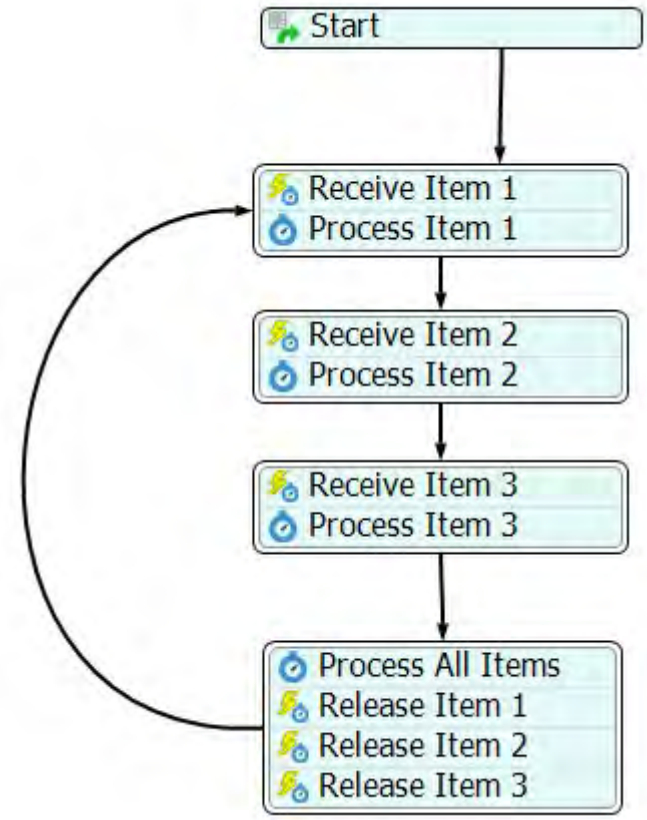
items through its output ports, just like any other fixed resource. However, the logic defining *when* to receive and release items is left up to the user, hence it is a "blank slate" fixed resource. This is just what we need for this tutorial.

In some other cases you may not want to use the BasicFR as the default, but instead want to use something like a queue or processor. You can define that by pressing the Edit Instance Object button in the Process Flow's General Properties, or by explicitly attaching existing objects to the Process Flow. In this tutorial, however, we will use the default BasicFR.

2. Create the 3D Layout - Create and connect the model 3D layout, namely a source, a queue, the Batch3 object, a processor, and a sink. Name the Batch3 instance *Batch3* and resize it so that items in it won't be covered by its shape.



3. Create the Process Flow Activities - Go back into the Batch3 Process Flow view. Create the Process Flow layout as shown in the following image.



This includes:

- o - A Schedule Source. o - 6 Wait for Events (3 to receive items, and 3 to release items). o - 4 Delays (1 for each received item and 1 for processing all items together).

4. Schedule Source Properties - By default, the Schedule Source creates a single token right when the simulation starts. This is exactly what we want to do in our logic, so we can leave it as it is.

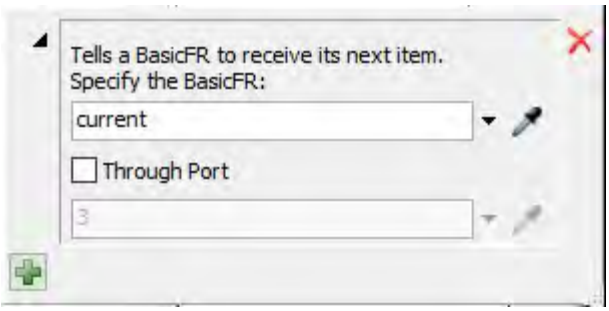
5. Receive/Process Activity Properties - For each of the 3 items, we want to, first, receive the item, and then process it for 5 seconds.

    1. Click on the *Receive/Process Item 1* block, and look in Quick Properties, under *Receive Item 1*.

    2. The Object field should be left as its default of current. This means we want to listen to an event on the instance object, namely the Batch3 object in the 3D model.

    3. For the Event field, press the button and then move the cursor over the Batch3 object in the 3D view. If the 3D view is behind a tab, hover over the 3D model tab to move to that tab, and then go to the Batch3 object. Click on the Batch3 object and select On Entry. This will make the activity listen for when an item enters the Batch3 object.

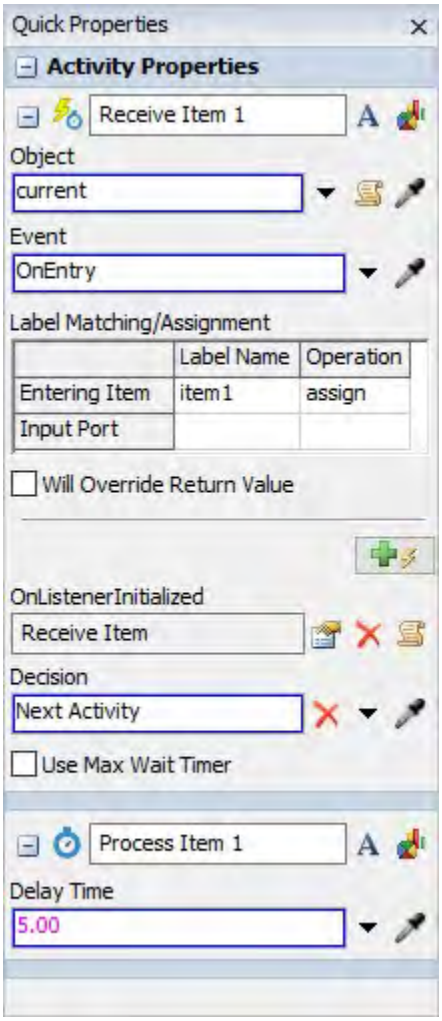
    4. Now you should see under Label Matching/Assignment a table with 2 rows. In the Entering Item row, enter *item1* for the Label Name column, and select *assign* for the Operation column. What we are doing here is saying that, when the OnEntry event happens, we want the token's *item1* label assigned to be a reference to the item that just entered as part of that OnEntry event. Later when we want to reference that item, we can use the token's *item1* label.
    5. Next, under OnListenerInitialized, press the button and select Control > Basic FR > Receive Item. Make sure the trigger properties are as follows:



This will make the object call the receiveitem() command, which makes it available to receive an item from the upstream queue.

6. Now go to the *Process Item 1* properties. Under Delay Time, enter 5.00.

The combined properties should look like the following:



Now implement the same properties on the other two Receive/Process blocks, except in those, define the Label Name as *item2* and *item3* respectively. This will make it so that each item will be assigned to a different label.

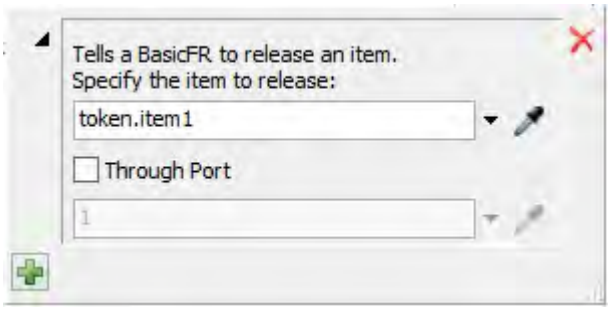
6. Process/Release Activity Properties - Once we have received and processed each individual item, we want to process all of the items together for 10 seconds, then release each item.

    0. Click on the Process/Release block, and go to Quick Properties.

    1. Give *Process All Items* a Delay Time of `10.00`.

    2. In *Release Item 1*, leave Object as `current`, and using the Event button, choose the On Exit event of the Batch3 object, just like we did for On Entry previously.

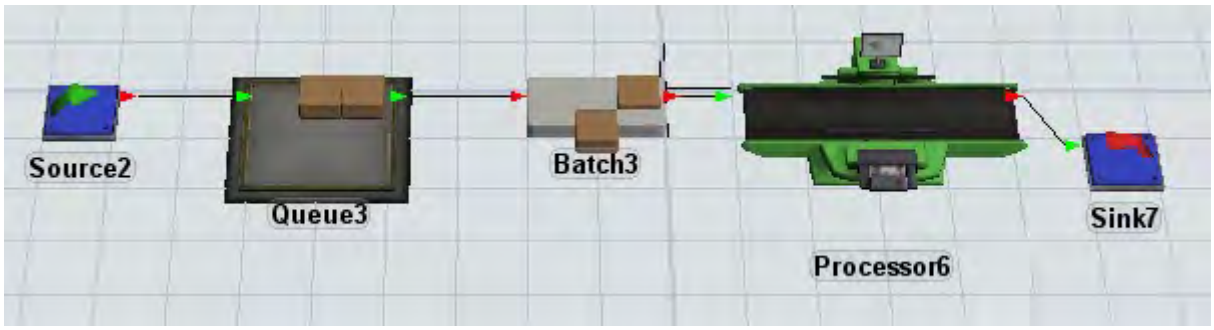
    3. Under OnListenerInitialized press the button and select Control > Basic FR > Release Item. Define the item to release as `token.item1`. The trigger properties should look like the following:

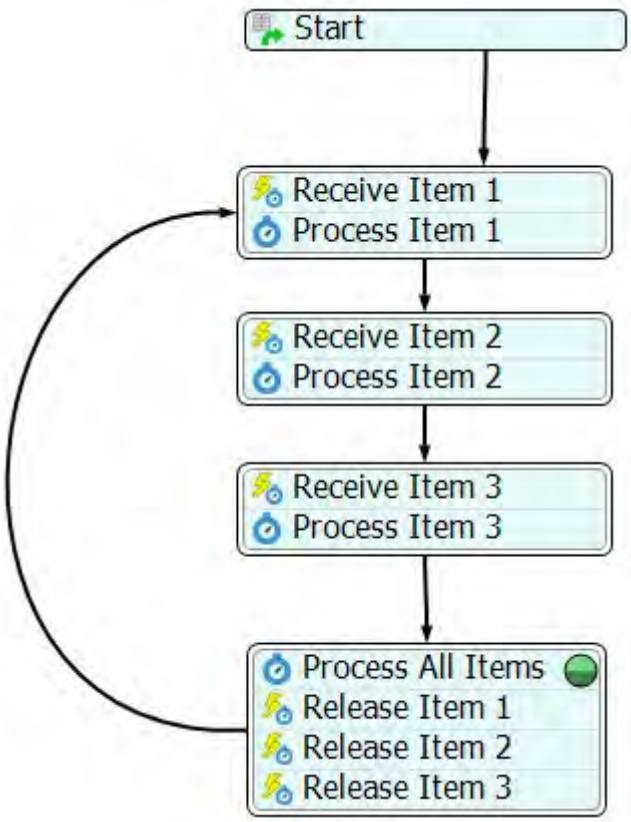


4. Do steps 3 and 4 again for the final two Release Item activities, except define the item to release as token.item2 and token.item3 respectively.

Run the Model

We can run the model now. Reset and run, watching the items in the 3D view and the tokens in the Process Flow.





Hey, It Doesn't Work!

First off, you may need to move the boxes around in the Batch3 to actually see all 3. When an item is moved into the Batch3, since we don't explicitly set its location, it will keep the same location that it had in the previous queue, except now in Batch3's coordinate space. Thus they may accumulate on top of each other. So, in the next phase we'll put in steps to set item location explicitly.

Second, and more importantly, you will notice that when the token gets to *Release Item 2*, it will actually release both items 2 and 3 at the same time and then jump to the beginning of the next batch. The processor then processes 2 items simultaneously. This is obviously the wrong behavior.

This is an artifact of the way FlexSim's event handling works. What is happening is that when the Processor is ready to receive its next item, it opens its input ports, which causes item 2 to exit Batch3. The OnExit is fired, which causes the next activity (*Release Item 3*) to be executed. Because all this functionality is triggered by the OnExit (which is called before the OnEntry of the processor) it all happens before the processor even knows that it received item 2 and can thus block further input. Consequently, item 3 is released and exits immediately as well, because it sees that the processor's input ports are still open.

This same problem can crop up in other scenarios. Within Process Flow it usually happens when a listened event is fired on a Wait For Event or Event-Triggered Source, and that is followed by one or more other activities that themselves may cause events in the model to fire, without any delays between the original triggered event and the subsequent event triggering logic. Affected logic may include operations like opening inputs or outputs, receiving or releasing items, moving objects, etc. The logic happens so fast that it doesn't give the original event enough time to "breathe" and finish what it was doing. In these cases, you can fix the problem by adding a Delay activity to delay for 0 seconds. This will allow the original triggering event to finish what it was doing before the token moves on. We will do this in the next phase of the model.

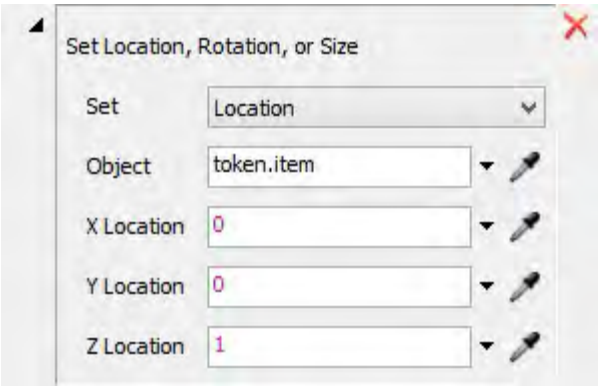
Let the Event-Triggering Object "Breathe"
If you have a Wait for Event or Event-Triggered Source that waits for an event and then immediately executes logic that may fire other events, it can be a good idea to put a 0 second delay after the eventwaiting activity. This will allow the object that triggered the event to finish its event logic.

Adjust the Model

Now let's go back and make adjustments to our Process Flow. First, we'll put activities in to set the item locations, so we don't get items on top of each other. Second, we'll put 0 second delays after release item activities.

1. Set Locations - For each of the 3 Receive/Process blocks, insert a Change Visual between the *Receive Item* and the *Process Item* steps. In their Quick Properties under Change Visual, press the  button and select Set Location, Rotation or Size. Set the respective locations to (0, 0, 1), (1, 0, 1), and (2, 0, 1). Make sure that the activities set locations on the items referenced by labels *item1*, *item2*, and *item3*, respectively.

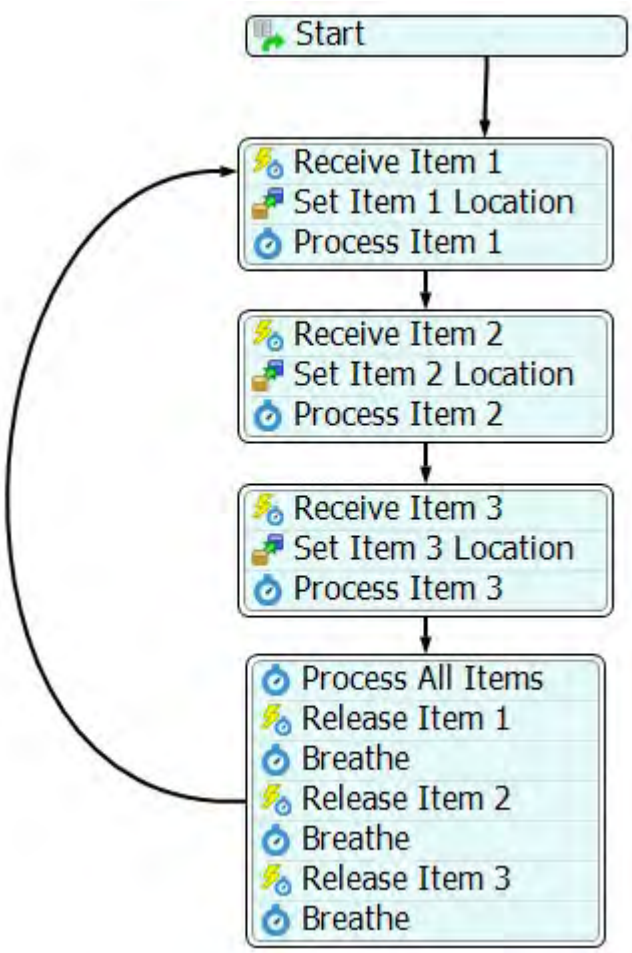
   Below is the trigger properties for the first Change Visual



Alternatively, you can set the Z Location value to `current.size.z`. This will put the item right on top of the Batch3.

2. Add Delays - Add a Delay of 0 seconds after each of the *Release Item* activities.

Below is an image of the modified Process Flow.



Run the Model

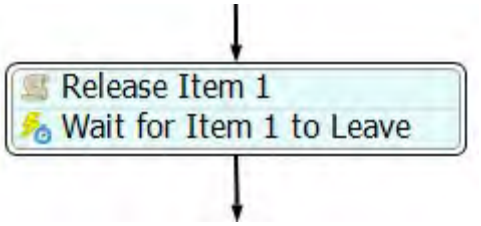
Now we can run the model again. Reset and run the model. You should see the items being collected 3 at a time, processed, then released.

Receiving/Releasing vs. Open Outputs/Inputs

In this model we used the Receive Item and Release Item options for receiving and release items in the OnListenerInitialized fields. We use these because our base object is the BasicFR "blank slate" class. Remember that this object does not have its own logic for receiving and releasing objects, so we have to put that logic in, and we do that using the Receive Item and Release Item options. If we were instead attaching our Process Flow to other objects like queues and processors, those objects already have their own logic for receiving and releasing items, so we would in those cases use options that close/open input/output on those objects. This acts as an "overlay" on top of the standard object logic, where you close output to block any released items from actually leaving, etc.

Using Wait for Event to Receive/Release Items

It may seem a little counter-intuitive that we are using a Wait for Event to actually call the functionality that receives and releases items. A more intuitive logic might look more like the following.



Here you first release the item, and then you wait for the item to exit. The problem with this implementation, though, is that the releaseitem() command may actually cause the item to exit before the command is even finished executing. This means that by the time you get to the *Wait for Item 1 to Leave* step, item 1 will already have exited, so the event it's waiting for will have already fired, and won't fire again. The token would thus get stuck in the wait. So, to alleviate this issue, the Wait for Event gives you an OnListenerInitialized trigger. This allows you to first set up the listening, and then execute the logic that receives/releases the item. Thus you can do it all in one activity.

For More Information

Adding and Connecting ProcessFlow activities

Linking a Process Flow to a FlexSim model

Tutorial - Custom Task Executer - Operator with Idle Breaks

Process Flow makes it easy to add functionality to existing Task Executers, which is one of the purposes of the Task Executer Process Flow. This tutorial will teach you how to modify an existing Task Executer by modeling an operator that returns back to a default location after performing a task. Additionally, if the operator doesn't get a new task to do within a given amount of time, he will go to a "break room" and wait for more work.

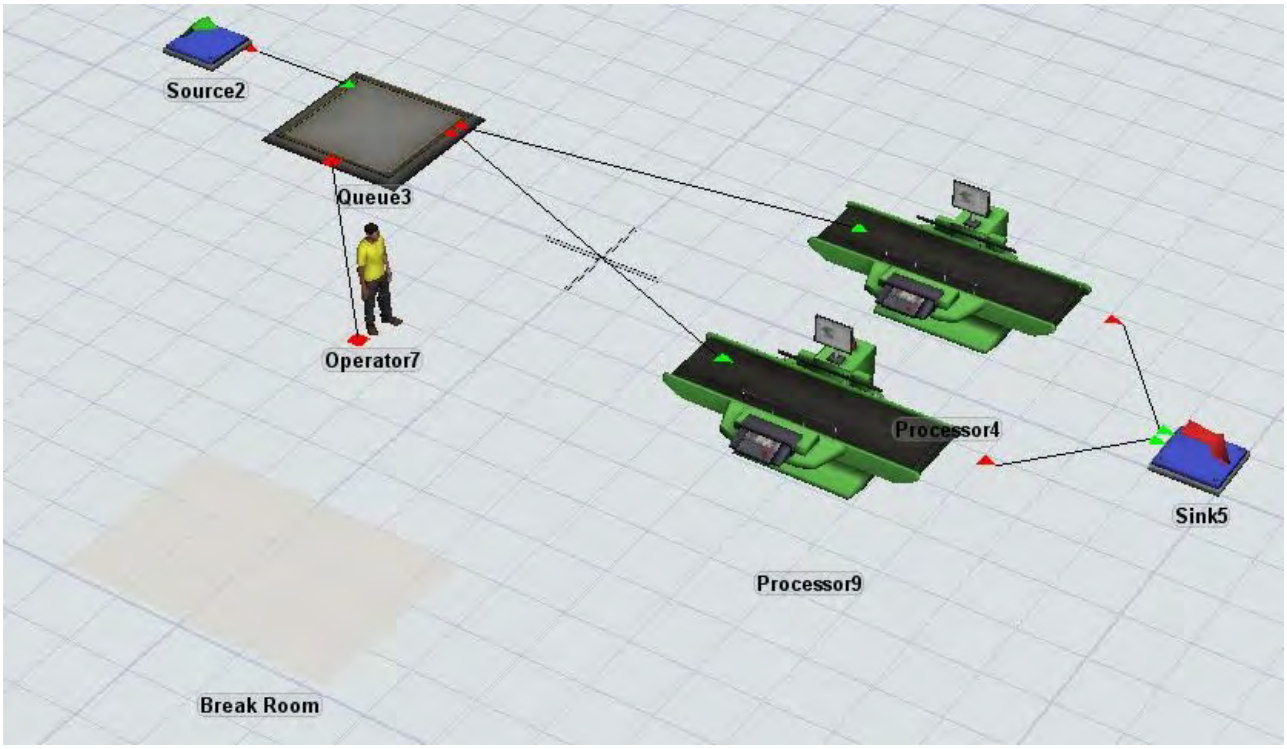
Tasks Covered

This tutorial will cover the following tasks:.

- Creating a Task Executer Process Flow
- Creating and using the Schedule Source, Decide, Travel, and Wait for Event activities.

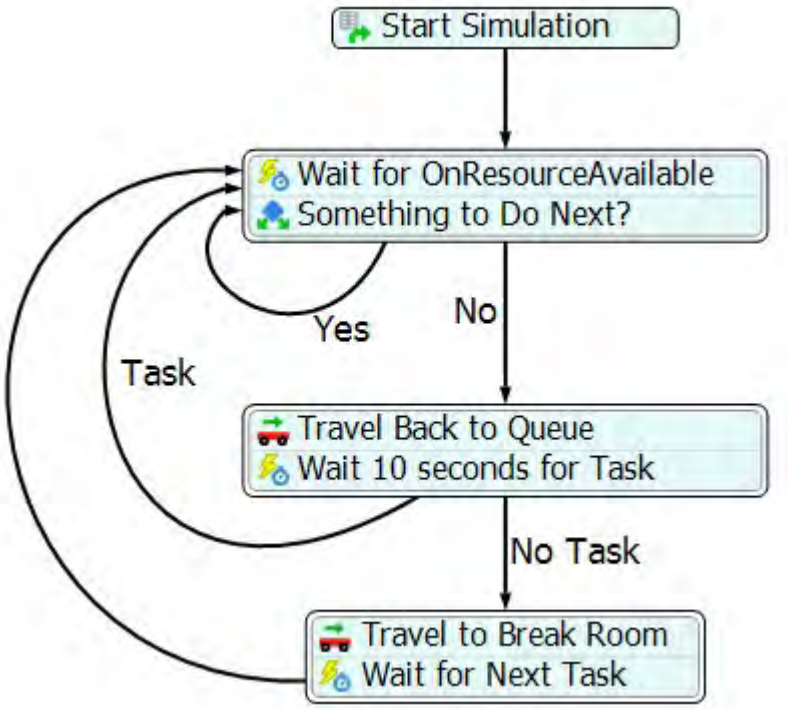
How It Will Work

The 3D model we will build is shown in the following image:



Here we will use standard modeling logic to make an operator transport items from a queue to one of two processors.

The process flow we will create is shown in the following image:

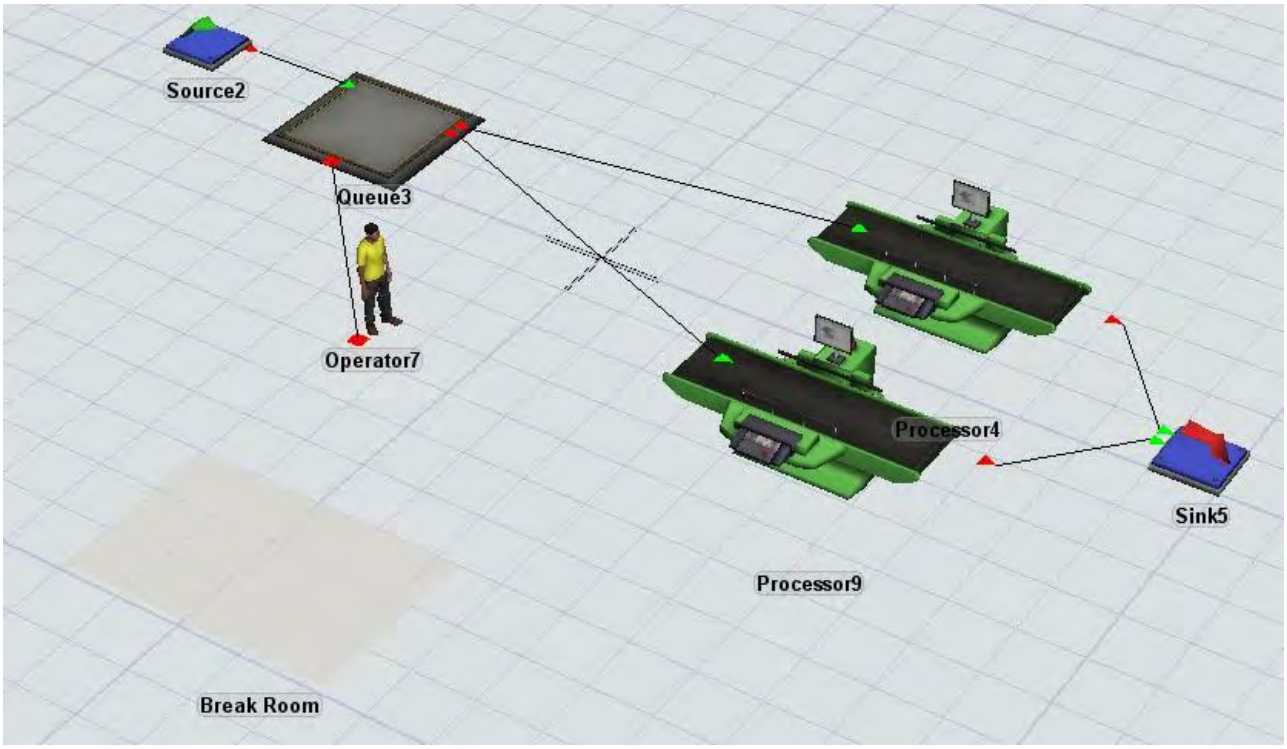


Since the standard model logic manages giving the operator work, the main task of the process flow will be to determine what to do when the operator doesn't have any standard work to do. We first wait for him to finish performing a task sequence. If upon finishing a task sequence the operator already has something to do next, then we will just wait again for him to finish that next piece of work. If there isn't something for him to do next, then we will tell him to travel back to the queue so he'll be right there when he does get his next task. Then, when he arrives at the queue, we'll wait to see if he gets a task within 10 seconds of arriving at the queue. If 10 seconds expires and he still doesn't get a task, then we'll tell him to go to the break room and wait there for more work.

Build the Model

3D Model Layout

First we'll build the 3D model layout. Create and connect objects as shown in the following image. This includes a source, a queue, two processors, a sink, an operator, and a visual plane.



Object Properties

Define object properties as follows:

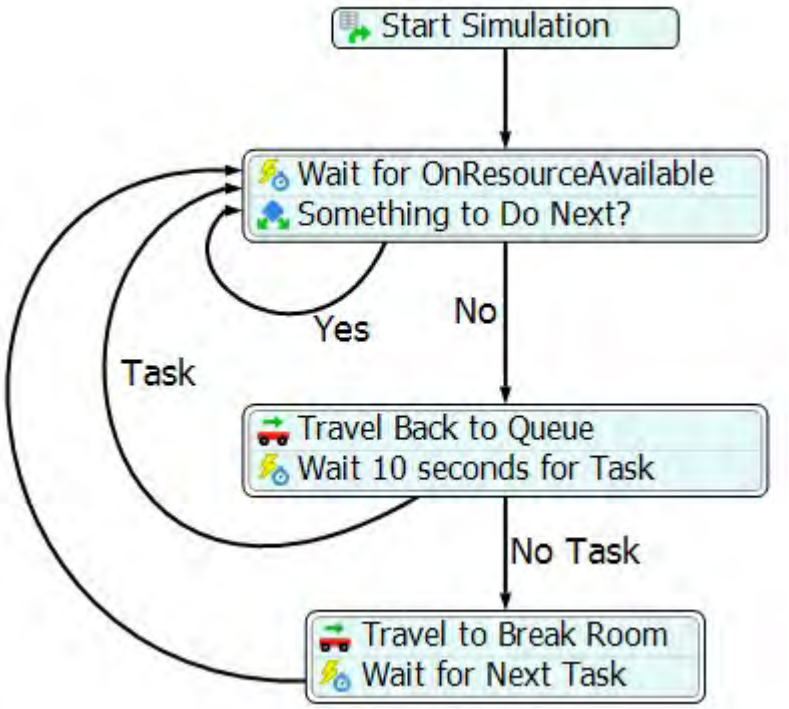
1. Change the source's Inter-Arrival Time to exponential(0, 30, 0)
2. In the queue's properties under Flow check the Use Transport box to tell the queue to get the operator to transport the item.

Process Flow

Now we can build the process flow. In the Toolbox press the button and select Process Flow > Task Executer. This will add a Task Executer process flow and open its modeling view.

Next, attach the process flow to the operator in the model. Click in the process flow view, and in Quick Properties under Attached Objects (instances), press the button and then click on the operator in the 3D view. This will make the operator into an instance of the process flow.

Create and connect activities as shown in the following image.



This includes:

1. A Schedule Soure activity

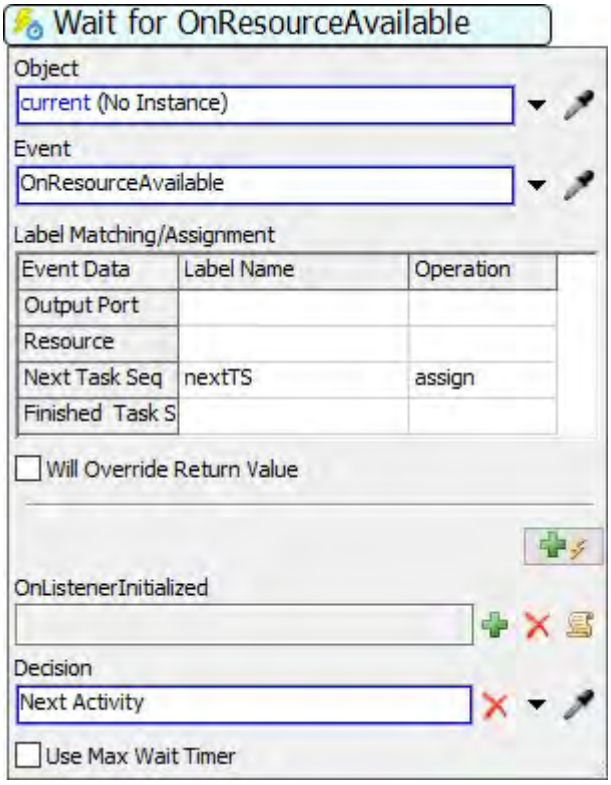
   Add the activity and name it *Start Simulation*. Its default properties will create a single token when the model starts. This is exactly what we want, so we can leave its properties as they are.

2. A Wait for Event activity

   Add the activity and name it *Wait for OnResourceAvailable*.

   In its properties, under Object press the button, then click on the operator in the model and choose current > On Resource Available.

   Next, under Label Matching/Assignment, in the Next Task Seq row, for the Label Name column enter *nextTS* , and for the Operation column select *assign* .

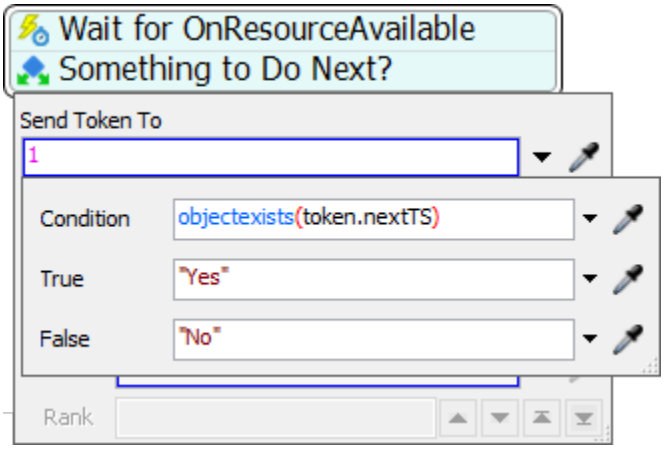


This will make the activity wait until the operator's OnResourceAvailable event is fired. This event is fired whenever the operator finishes a task sequence. Also, if when the event is fired the operator already has another task sequence to do in his task sequence queue, a reference to that task sequence will be assigned to the token's *nextTS* label. We will use this in the next activity to decide whether we need to tell him to travel back to the queue.

3. A Decide activity

Add the activity and name it *Something to Do Next?*.

In its properties, under Decision press the button and select Conditional Decide. Then in the behavior settings popup, for Condition enter `objectexists(token.nextTS)`. For True enter `"Yes"`. For False enter `"No"`.

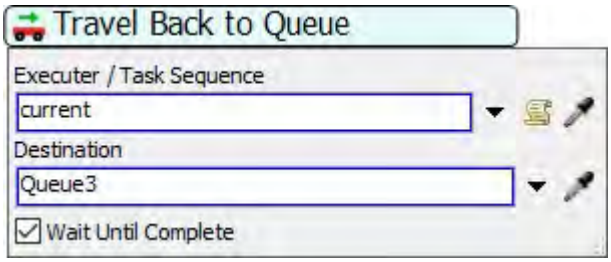


This will check if the token's *nextTS* label (the label that was assigned in the previous activity) points to a valid task sequence. If so, it will send the token out its "Yes" connector. If not, it will send the token out its "No" connector.

4. A Travel activity

   Add the activity and name it *Travel Back to Queue*.

   In its properties, under Executer / Task Sequence enter current, and under Destination press the button, then click on the queue in the model.



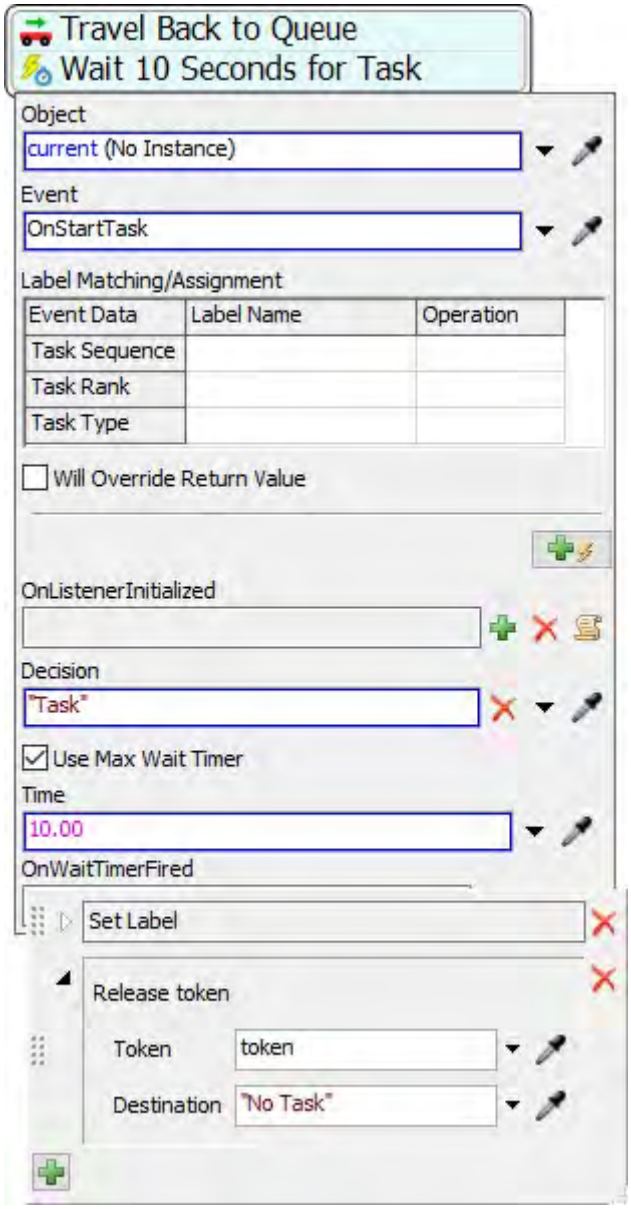
   This will tell the operator to travel back to the queue.

5. A Wait for Event activity

   Add the activity and name it *Wait 10 Seconds for Task*.

   In its properties, under Object press the button and then click on the operator in the model and select current > On Start Task.

   Next, check Use Max Wait Timer, and for its Time enter *10*. Then in the behavior settings for OnWaitTimerFired expand the Release Token behavior and for Destination enter "No Task"

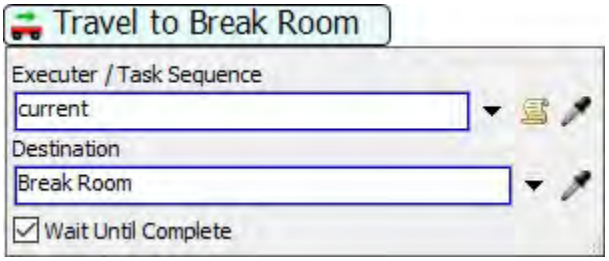


This will make the activity wait for the operator's *OnStartTask* event to fire (this event is fired every time the operator starts performing a task). If the event fires within 10 seconds, then it will send the token out its "Task" connector. If the event does not fire within 10 seconds, it will abort the activity and send the token out its "No Task" connector.

6. A second Travel activity

   Add the activity and name it *Travel to Break Room*.

   In its properties, under Executer / Task Sequence enter `current`, and under Destination press the button, then click on the break room visual plane in the model.

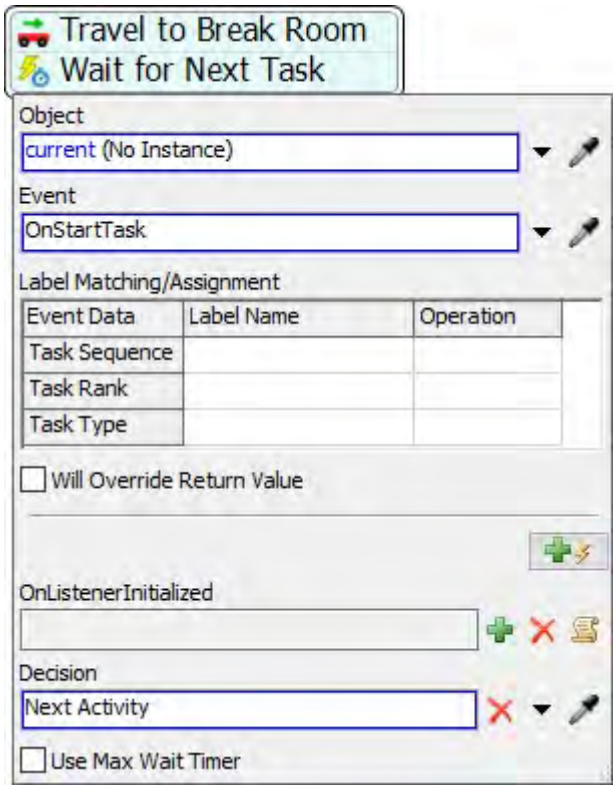


This will tell the operator to travel to the break room.

7. A final Wait for Event activity

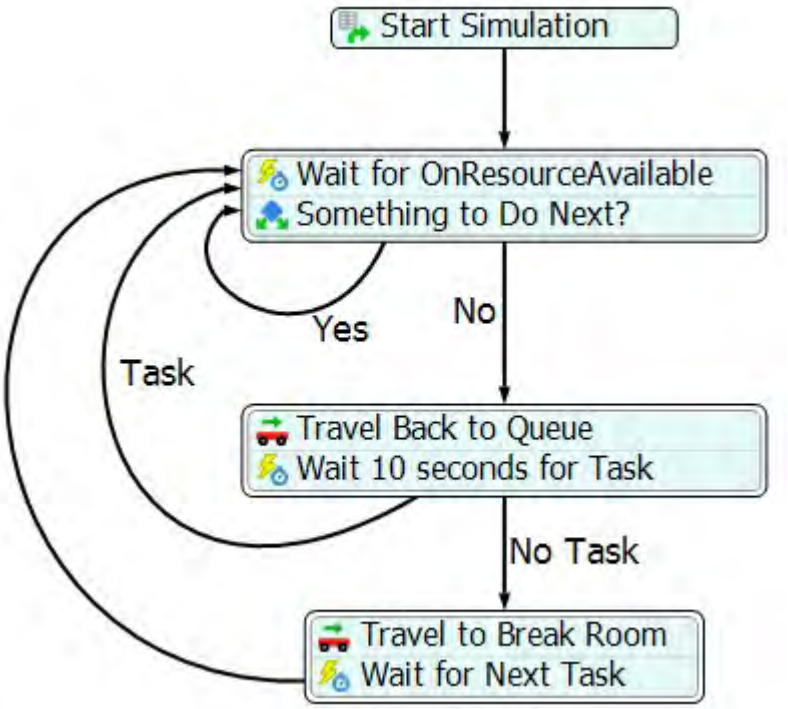
Add the activity and name it *Wait for Next Task*.

In its properties, under Object press the button and then click on the operator in the model and select current > On Start Task.



This will make the activity wait for the operator's *OnStartTask* event to fire (this event is fired every time the operator starts performing a task).

Once you've created all the activities, make sure they are connected as follows:



Since we are referencing connectors by their names, make sure especially that the names are correct.

Run the Simulation

At this point the model is ready. Reset and run. You should see the operator transporting items to the processors. When the operator finishes a transport operation and has nothing to do, you should see the token move through process flow and direct him to travel to the queue. Then if he is idle at the queue for 10 seconds, he will travel to the break room and wait there.

For More Information

Adding and Connecting ProcessFlow activities

Linking a Process Flow to a FlexSim model

AGV Lesson 1 Introduction

This lesson introduces FlexSim's AGV module, an add-on to FlexSim's object library.

What You Will Learn

- How to build AGV path geometry
- How to add Control Points to the AGV network

- How to assign TaskExecuters to travel using the AGV network
- How Control Point allocation/deallocation works.

New Objects

In this lesson you will be introduced to Straight and Curved Paths, and Control Points.

Time to Complete

This lesson should take approximately 15-20 minutes to complete.

AGV Model Overview

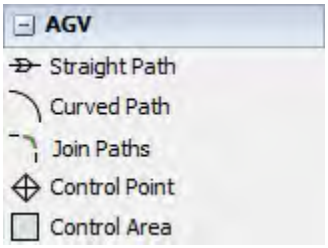
In our AGV model we will have two different AGVs travel to various spots on an AGV network, picking up and dropping off items.

Click here for the Step-By-Step Tutorial.

New Concepts

AGV Library Objects

When the AGV module is installed, you will see the AGV network building tools in library drag-drop window on the left.



Creating Paths

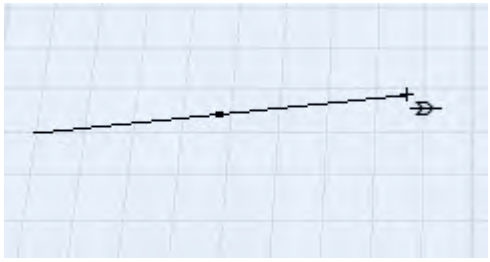
AGV library interaction works differently than with most other objects in the library. Here instead of dragdropping objects into the model, you just click on them in the library, and then make consecutive clicks in the model to define geometry.

1. Click 

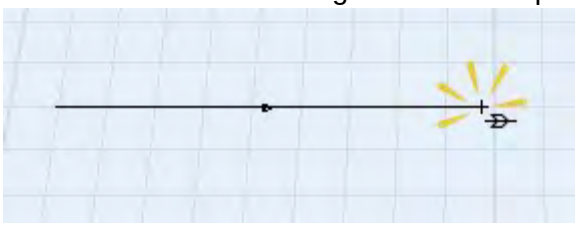
-release on Straight Path in the library.

2. In the 3D view, click-release at the position where you want the path to start.

3. Move the mouse to the position where you want the path to end.



4. Click-release the mouse again at the end position.

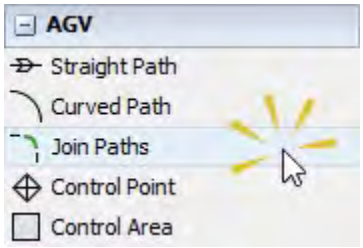


This method is the same for creating both Straight Paths and Curved Paths.

Joining Paths

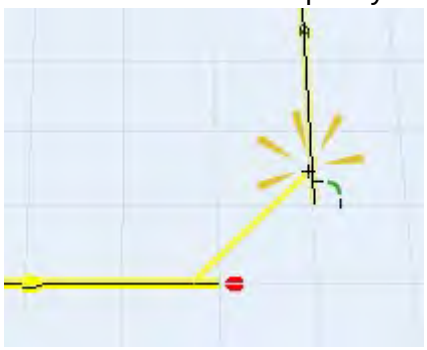
The AGV library also includes a Join Paths tool. This tool will join two paths together with a curved path.

1. Click-release on the Join Paths tool in the library.

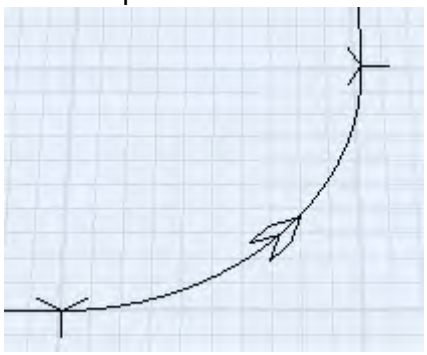


2. In the 3D view, click-release on the path you want to connect from.

3. Move the mouse to the path you want to connect to, and click on that path.

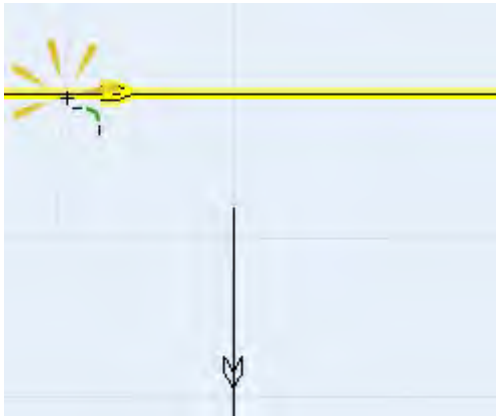


4. A curved path will be created that connects the two paths.

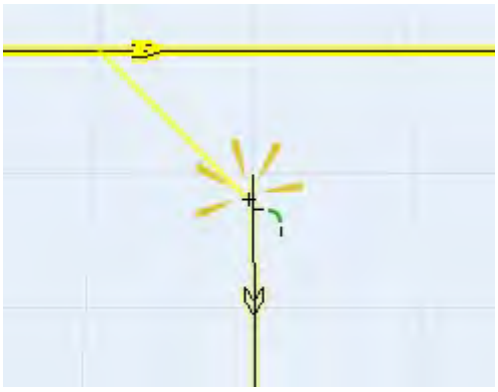


The Join Paths tool can also be used to join the end of one path with some point along another path.

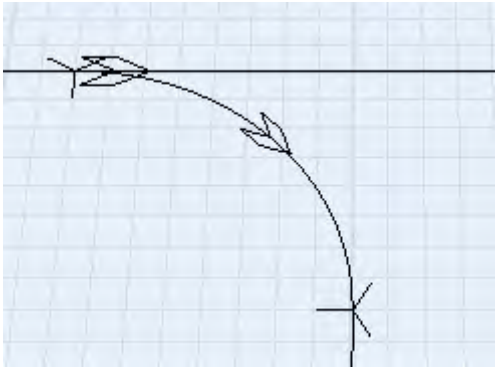
1. In the 3D view, click-release on the path you want to join from, on the side associated with where you want the curved path joint to be created.



2. Move the mouse to the path you want to connect to, and click on that path.



3. A curved path will be created that connects the two paths.

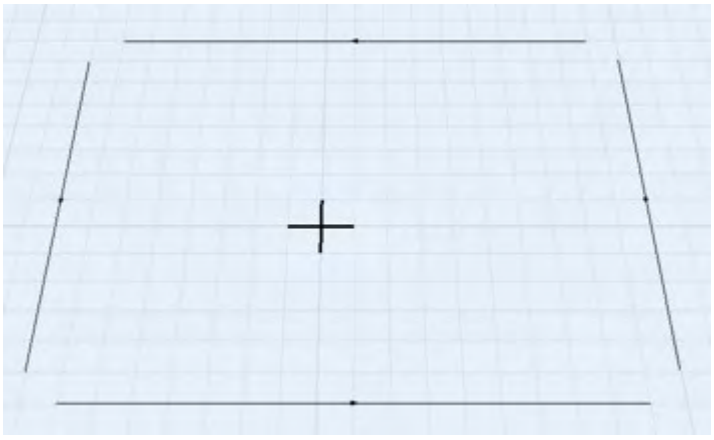


Now build the tutorial model with the step-by-step instructions.

AGV Lesson 1 Step-By-Step Model Construction

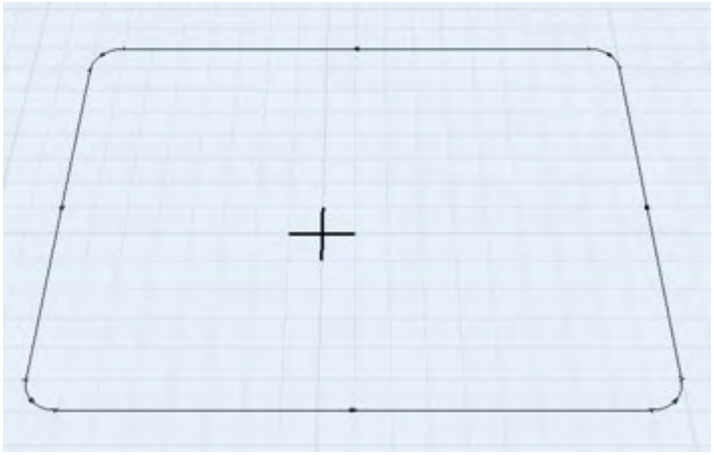
Build the Model

1. Start with a new model.
2. Using the Straight Path tool as explained in the introduction, create four straight paths that go around in a counter-clockwise loop. Make sure to leave gaps on the corners for the joining curved paths to be created.

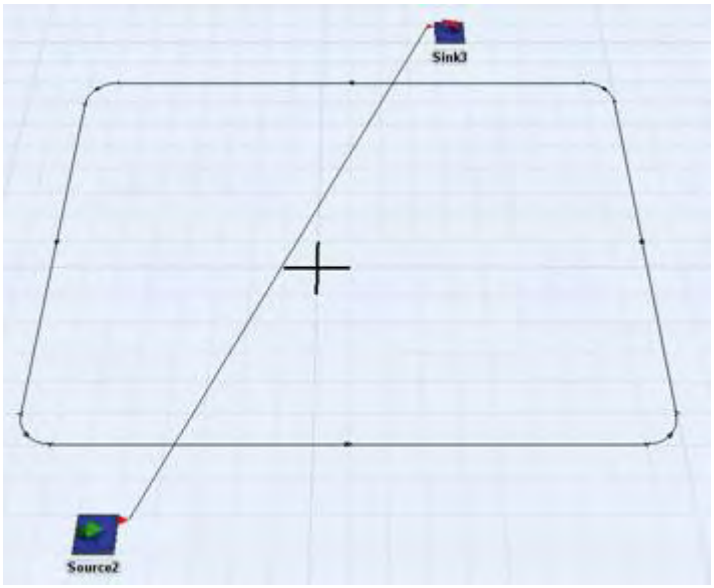


Note: Path Directionality is Important. By default, paths are created one-way. If your model is working, it may be because you didn't create the paths in the right direction. In this model we create a single counter-clockwise loop. Make sure you create the loop in a consistently counter-clockwise direction. Look at the arrow indicator on each path to see its direction.

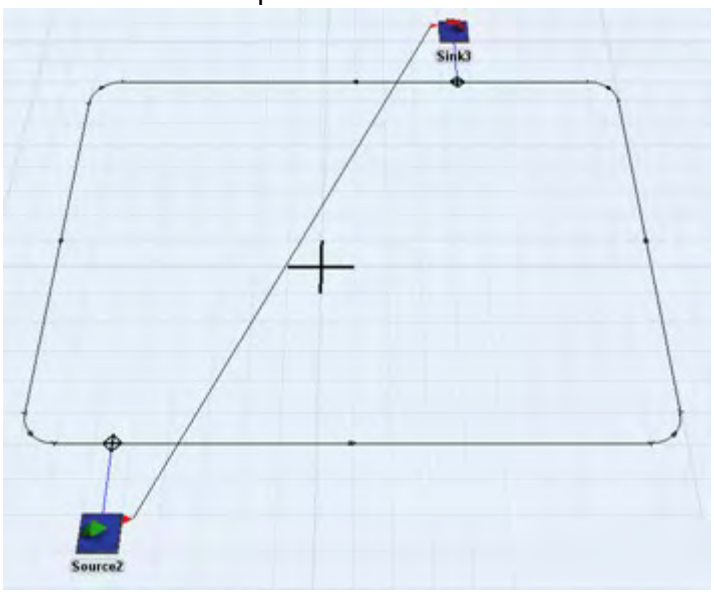
3. Using the Join Paths tool as explained in the introduction, join the four straight paths with curved paths. Again, make sure you connect them in a counter-clockwise direction.



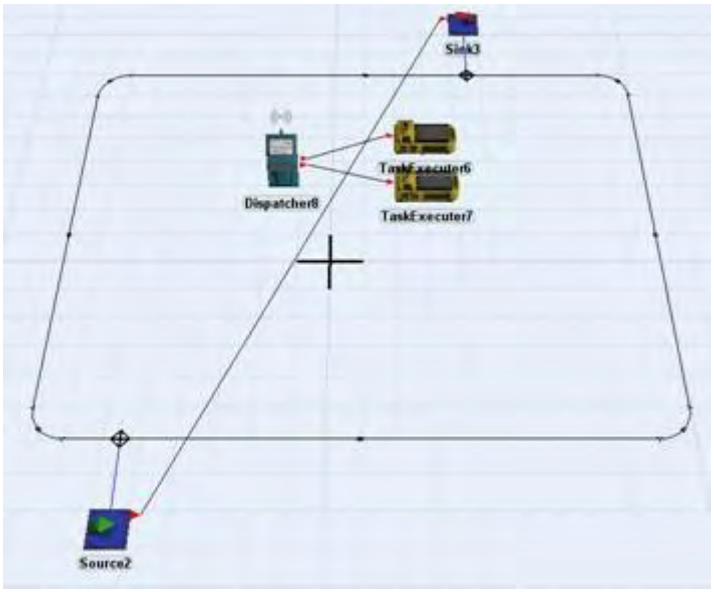
4. Add a Source at the bottom of the loop, a Sink at the top, and connect them with an A connect.



5. In the library, drag a Control Point into the model. Position it on the bottom path, close to the Source. Using an A connect, connect that control point to the Source.
6. Create a second Control Point and position it on the top path near the Sink. Again using an A connect, connect that control point to the Sink.

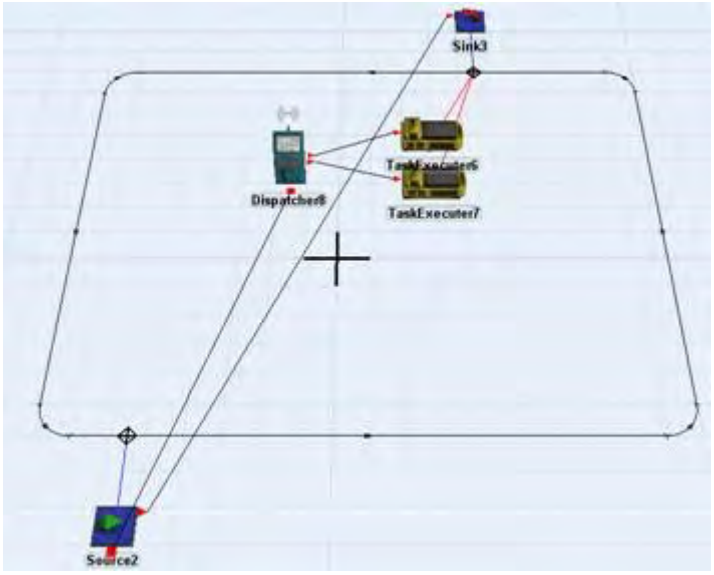


7. From the Library, drag two TaskExecuters into the model.
8. Drag a Dispatcher into the model, and connect it to the TaskExecuters using the A connect.



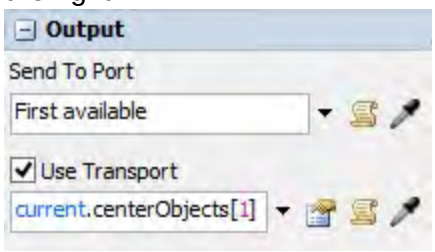
9. Shift-Select the two AGVs and connect them to the Control Point near the Sink with the A connect. From the popup menu, choose Traveler AGV. This tells the AGV that its travel operations should be done using the AGV network.

10. Tell the Source to use a transport. Use an S connect to add a center port between the Source and the Dispatcher.



Then click on the Source and check the Use Transport box in the Output panel of Quick Properties on

the right.



Run the Model

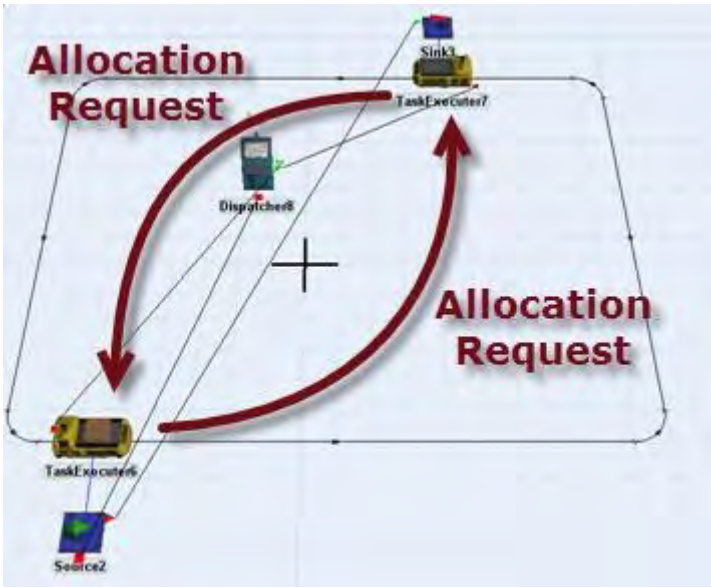
Reset and run the model.

You should see one of the TaskExecuters travel to the Source, pick up an item, and then the model will show a "deadlock" message.

Control Point Allocation

This first model demonstrates a significant difference between the AGV module's logic versus the standard travel network logic in FlexSim. The AGV system's look-ahead mechanism is more sophisticated. AGVs will look ahead all the way to the next Control Point on their path and attempt to allocate, or claim, that Control Point. If an AGV cannot allocate its next Control Point, it will stop at the previous Control Point.

This stop distance may span several path sections before the next Control Point's location.



Deadlock

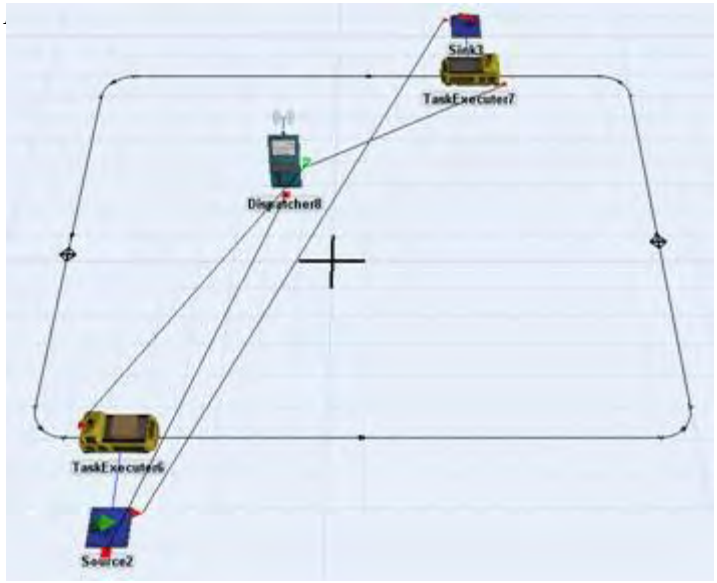
This model contains a deadlock situation. The first AGV (TaskExecuter6) is trying to allocate the Sink's Control Point, which is currently claimed by the second AGV (TaskExecuter7). However, TaskExecuter7 is currently waiting to claim the Source's Control Point which is claimed by TaskExecuter6. This type of deadlock is called circular wait, because the allocation/allocation request chain forms a circular loop of AGVs waiting on each other. Deadlock is a situation that is encountered quite often in the design of AGV systems, and its avoidance is an important reason why simulation is used during the design process.

FlexSim provides a feature where the model will automatically detect deadlock in your system. This setting is on by default. You can configure it by right-clicking on an AGV Path or Control Point, and in the context menu choose "AGV Network Properties". Go to the General tab of the AGV Network Properties window. Check or uncheck the "Check for Deadlock" box to define this setting, then press OK to close the properties window.

The Check for Deadlock feature can be very useful in the debugging phase, but since it does require additional calculations, you may want to turn it off once your model is functioning properly and you want to run faster scenarios.

Resolving the Deadlock

There are several strategies for resolving deadlock in an AGV system. In our case we can just add extra control points into our system. This adds more potential stopping points to the system, so that AGVs don't have to stop at a control point that is blocking another AGV.

1. 

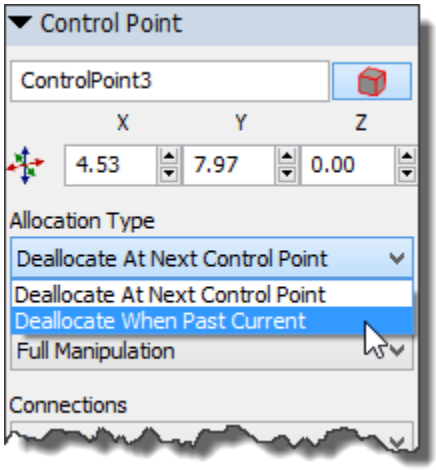
2. Reset and run the model.

Now the AGVs will no longer reach deadlock.

Control Point Deallocation

With the model's new configuration, AGVs no longer reach deadlock, but you may notice that they will still occasionally wait on each other. This again is caused by the Control Point Allocation/Deallocation mechanism, specifically by each Control Point's Deallocation Type. This setting defines when the Control Point will be deallocated. The default setting is the most conservative, namely to deallocate a Control Point when the AGV reaches its next Control Point. This will essentially cause a two-Control-Point gap to be preserved between AGVs when they are actively traveling. You can change this setting.

1. Shift-Select all the Control Points in your model. It's OK if you select more that just control points. Just lasso everything.
2. Click on one of the Control Points.
3. In Quick Properties on the right, under Deallocation Type, choose Deallocate When Past Current



4. Press the popup button Apply to All Selected. This will apply that change to all selected Control Points.
5. Reset and run the model.

Now AGV's will not wait on each other as much.

Deallocation Types

A basic AGV model has two Allocation Types by default. They are Deallocate At Next Control Point, and Deallocate When Past Current. You can reconfigure the settings for these Deallocation Types, or add your own, in the Deallocation Types tab of AGV Network Properties.

Finished

You're finished with this model. Move on the the next tutorial.

AGV Lesson 2 Introduction

This lesson extends the previous lesson to take you deeper into AGV simulation concepts.

What You Will Learn

- How AGV speeds are determined
- How to connect AGVs to a template process flow to perform AGV control.
- How to create and dispatch to spurs in the AGV network for drop-off
- What Control Point Connections are and how to use them to implement AGV dispatch logic
- How AGV Way Points work

Time to Complete

This lesson should take approximately 15-20 minutes to complete.

AGV Model Overview

Lesson 2 will start where Lesson 1 left off. First we will associate our AGVs with a template process flow. This process flow will do implement AGV control logic. Finally, we will define custom max speeds for the curved paths and spurs.

Click here for the Step-By-Step Tutorial.

New Concepts

AGV Process Flow

In lesson 1 we used the default Use Transport functionality, which creates a task sequence to transport the item, and dispatches it to a dispatcher object, which subsequently gives it to an AGV for transport. Often in real systems, however, the logic is more complex. Work is often more location-based, meaning transport requests are associated with the pickup location, and AGVs take up work at a location when they arrive there, instead of being dispatched immediately by a dispatcher when the work is first requested. Also, logic for when and where an AGV should park should be taken into account. In the lesson 1's model, when an AGV finishes a dropoff, if it's not immediately dispatched new work, it will simply sit where it is. This may not be the desired behavior.

Given these considerations, in lesson 2 we take a different approach to AGV control. Here we will use a process flow to define the AGV control logic. FlexSim includes a template AGV process flow that includes all of the logic we need for this lesson.

Note that detailed information on how process flow works is outside the scope of this lesson. See the process flow topics for more information on how process flow works.

Control Point Connections

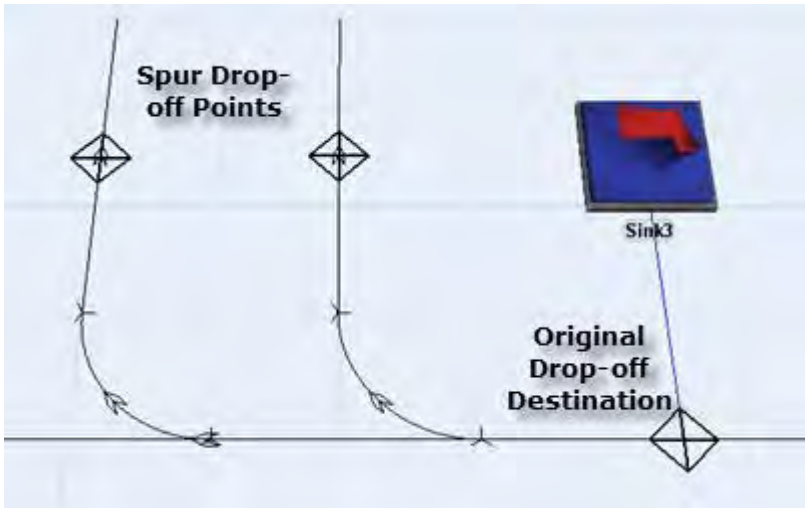
Control Point Connections are logical associations between Control Points in a model, or associations between Control Points and other model objects. While automatically you get a default set of Control Point Connection types, you can add, remove, or redefine these connections as needed.

Control Point Connections take on meaning as you implement logic that uses them in your model. For example, if an AGV is at a Control Point, and can't find any work to do at that Control Point, you can use a Control Point Connection to say "This is the next control point to go to to look for work." This is actually one of the default Control Point Connections, namely the NextLookForWork connection. Alternatively, if an AGV arrives at an "initial destination" Control Point to drop off a flow item, but you want to redirect it to an available "spur" location to make the drop-off, you can use a Control Point Connection to say "Here are the set of drop-off control points to search in dropping off the flow item". Again, this is another Control Point connection that you get by default, namely the DropoffPoints connection.

The process flow we will add has logic that uses several of the default Control Point connections. Once we add the process flow and associated with our AGVs, we can create various control point connections in the model, and the process flow's control logic will automatically recognize them and direct the AGVs accordingly.

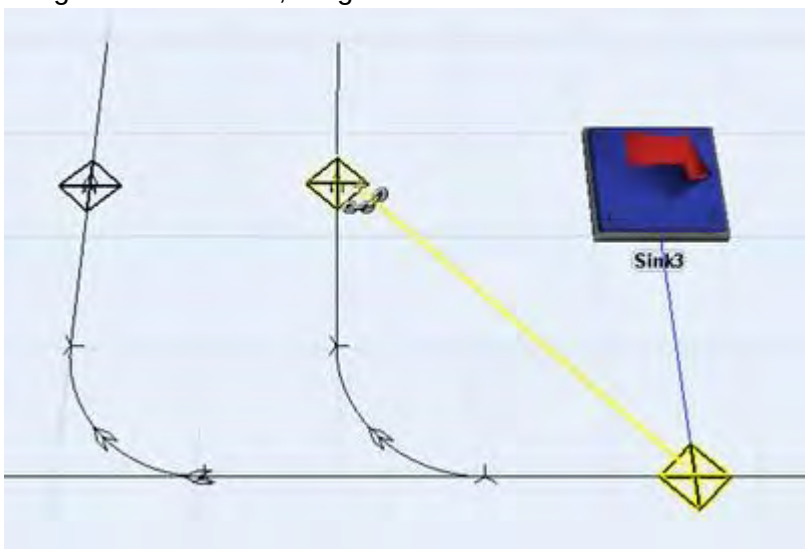
To view and configure Control Point connections, right-click on a Path or Control Point and choose AGV Network Properties. Go to the Control Point Connections tab.

Here you can add or remove control point connections, and define what color they will be drawn with and whether they are visible in the 3D view. The process flow queries these connections and dispatches the AGVs accordingly. For example, consider the drop-off dispatch situation mentioned previously. In this lesson we will change the model so that AGVs are dispatched to spur locations for drop-off.

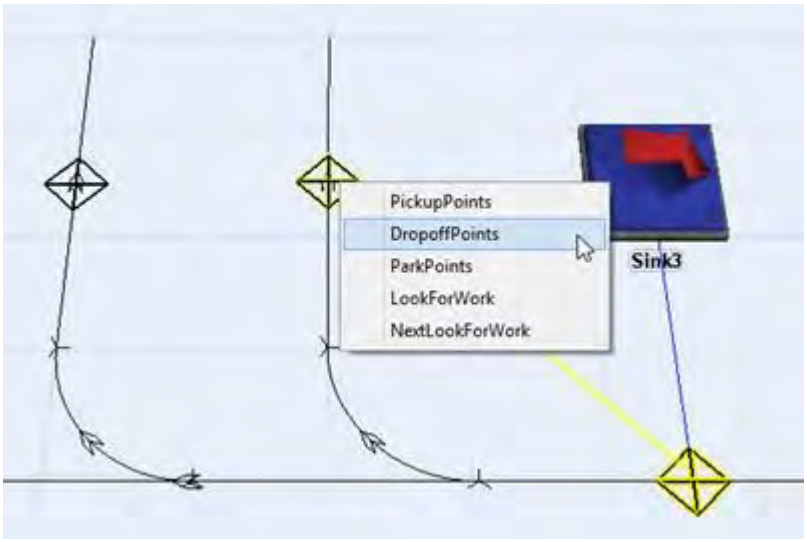


To do this, you add "DropoffPoints" connections from the original drop-off point at the Sink to the set of spurs you want to drop-off to. This is done using the standard "A" connect. Adding a Control Point Connection

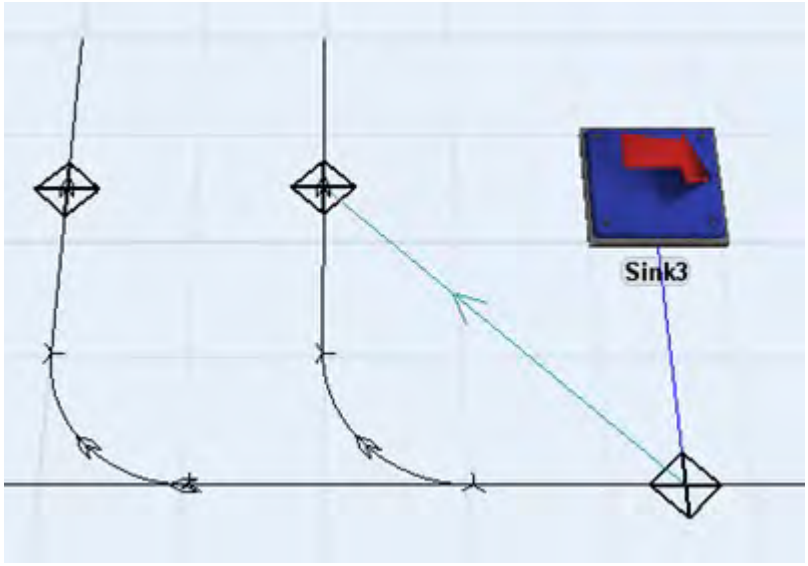
1. Using the "A" connect, drag-connect from one Control Point to another Control Point.



2. Choose from the drop-down menu the type of connection you would like to create.



This will create the associated connection between the Control Points



Once the connections are established, the process flow will direct the AGVs according to those connections.

Build the Model

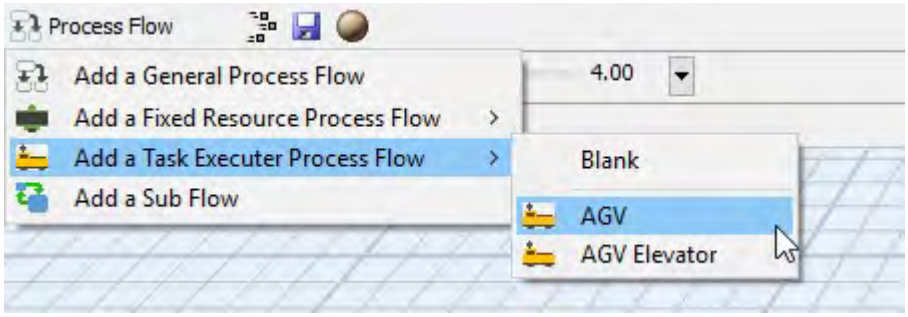
Now build the tutorial model with the step-by-step instructions.

AGV Lesson 2 Step-By-Step Model Construction

Start with the finished model from Lesson1.

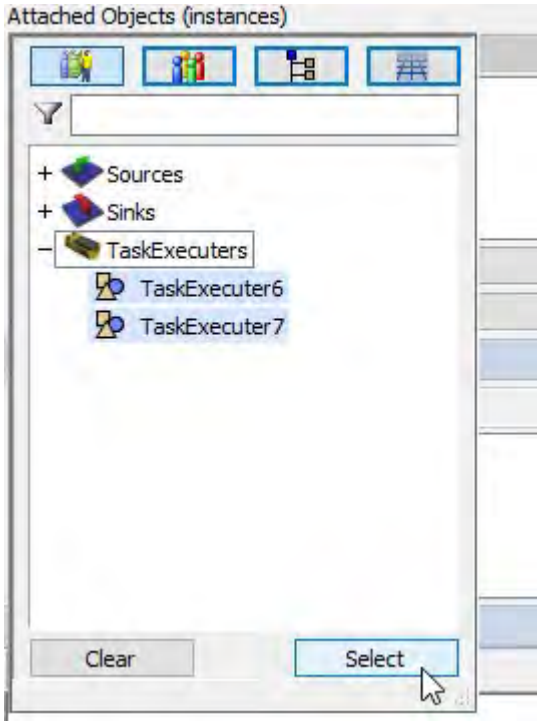
Add and Integrate the AGV Process Flow

1. From FlexSim's main toolbar, press the Process Flow button. Then choose Add a TaskExecuter Process Flow > AGV.



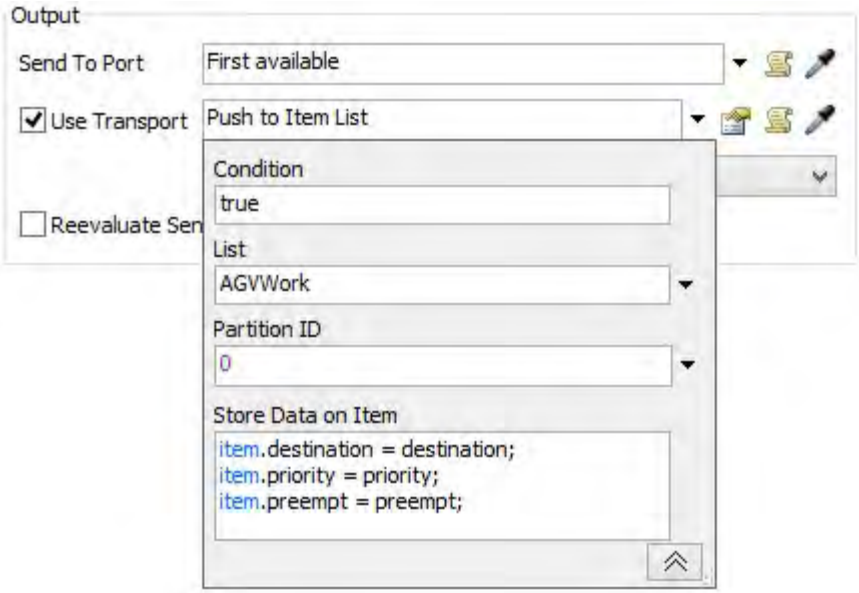
This will add an AGV process flow to the model. The process flow includes detailed instructions on how to use it. It's a good idea to read those instructions, but this tutorial also includes the primary tasks that need to be done to integrate the process flow with the model.

2. Associate the process flow with each AGV in the model. To do this, click in a blank area in the process flow view, and then in Quick Properties under Attached Objects, press the  button, and select both AGVs.



This will associate each AGV with this AGV process flow, which means that this process flow becomes like the "brains" of each AGV.

3. Open the source's properties window, go to the Flow tab, and in the Use Transport field, click the ▾ button, and choose the pick list option Use List > Push to Item List (No Task Sequence). Make sure that the List field says *AGVWork*.

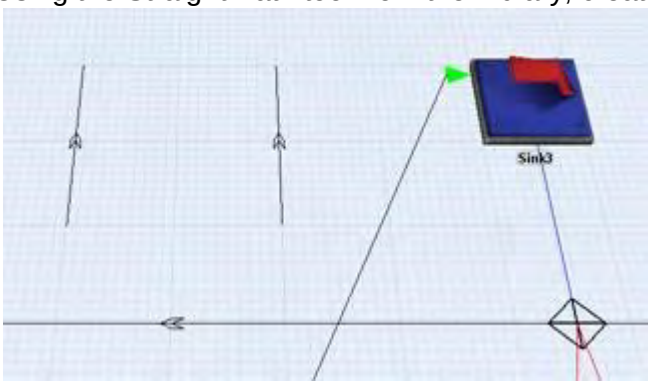


The process flow monitors this *AGVWork* list. When an item is placed on the list, it sorts it based on the item's location, and then an AGV will pick up the item when it gets to that location. Press OK to close the source properties window.

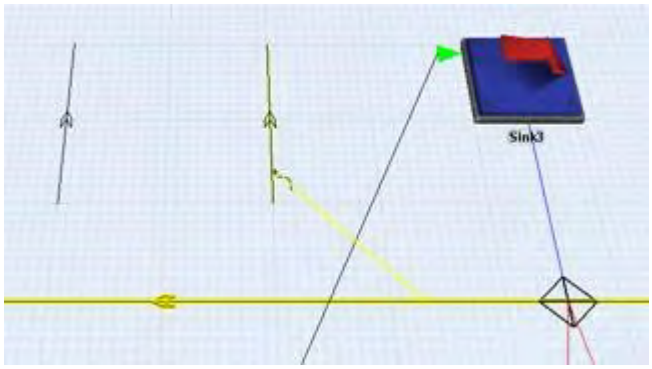
4. Delete the dispatcher. Since the AGV process flow handles work dispatching, the dispatcher object will no longer be need.

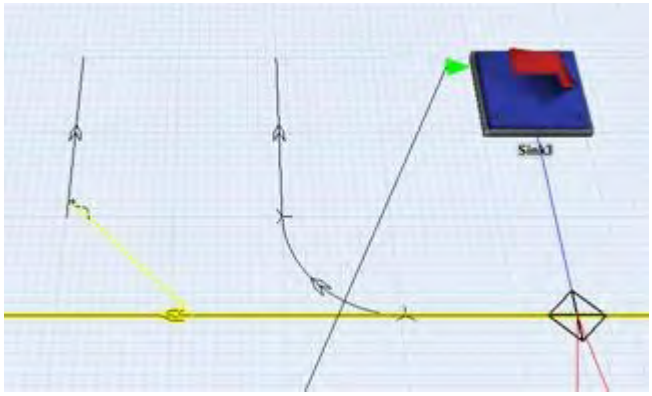
Add Drop-off Spurs

1. Using the Straight Path tool from the Library, create two straight spur paths near the Sink object.

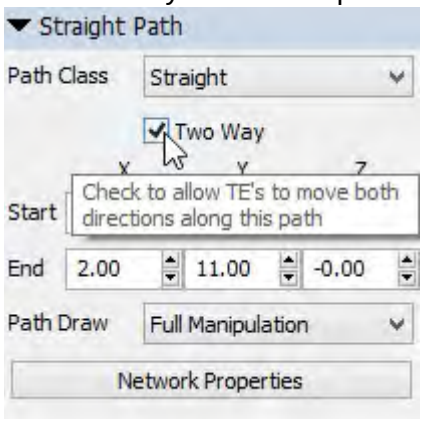


2. Using the Join Paths tool, make a click-drag connection from the horizontal path that goes past the Sink to each of the spur paths.

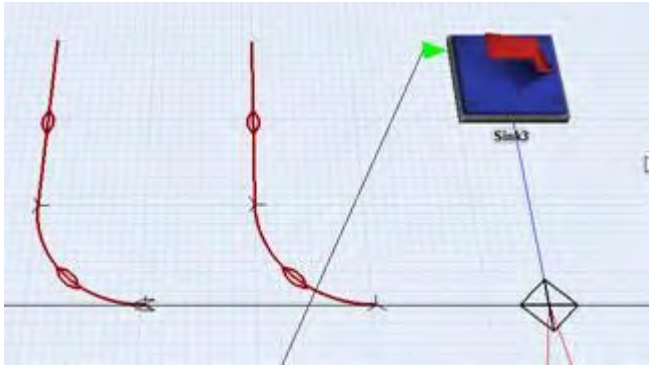




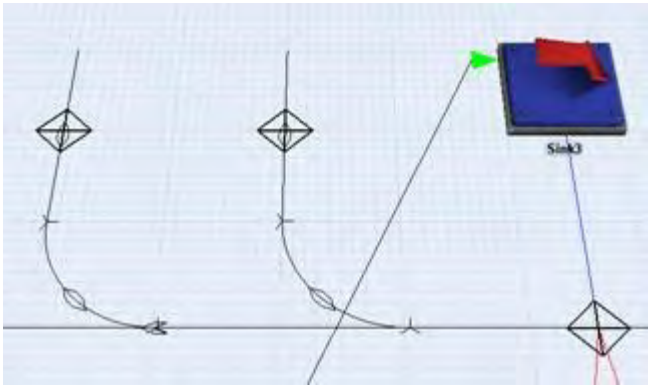
3. Shift-Select all four spur paths (2 straight and 2 curved).
4. Click on one of the straight spur paths, and in the Quick Properties Straight Path panel on the right, check the "Two Way" box. Then press the "Apply To All Selected" popup button.



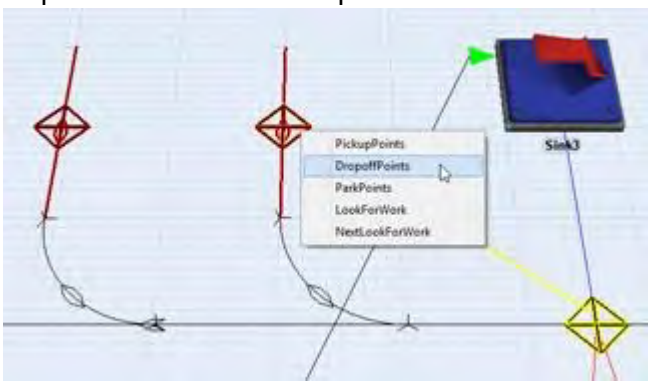
This will make the spur paths two-way, so that AGVs can both enter and exit the spurs. You can see that the paths are two way by their arrow indicators. They look kind of like a diamond shape, with the ends pointing in opposite directions.



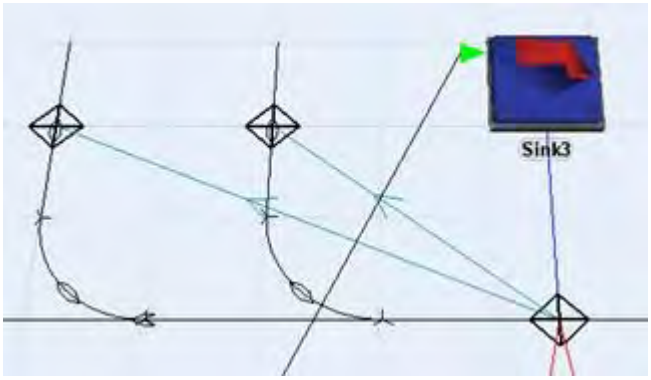
5. Add a Control Point at the center of each spur.



6. Shift-Select the two new Control Points. It's OK if you also select the Paths.
7. A-Connect from the Control Point on the main loop to one of the spur Control Points, and choose DropoffPoints from the drop-down.



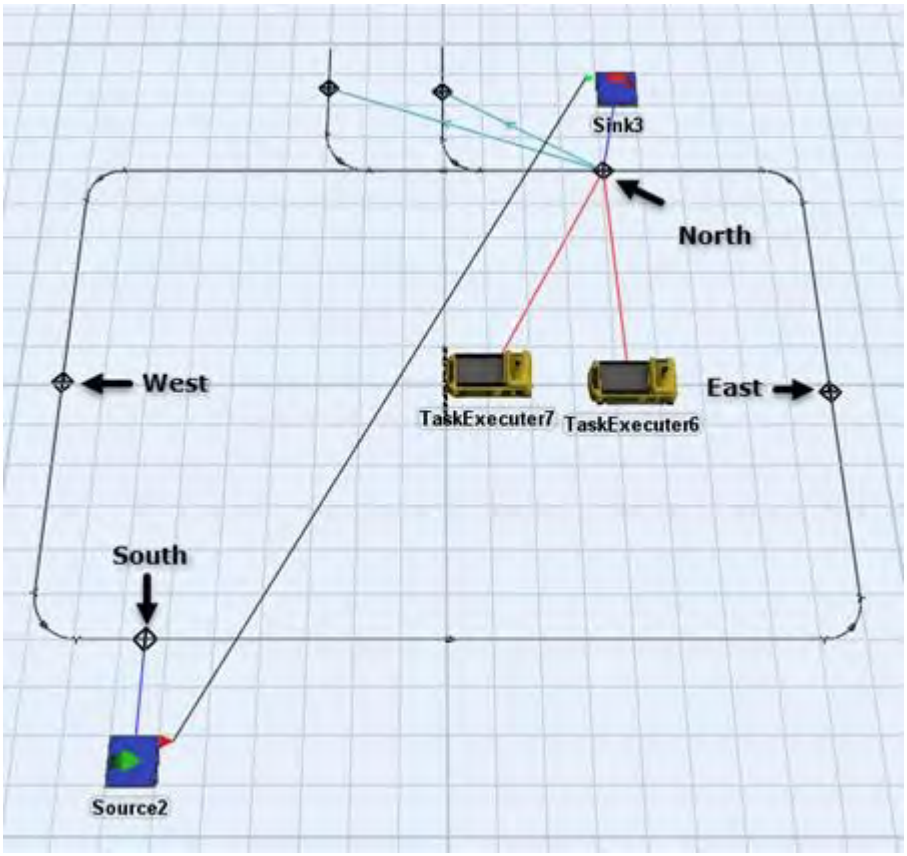
8. Shift-Click in a blank area to deselect the Control Points.



Name the Main Loop Control Points

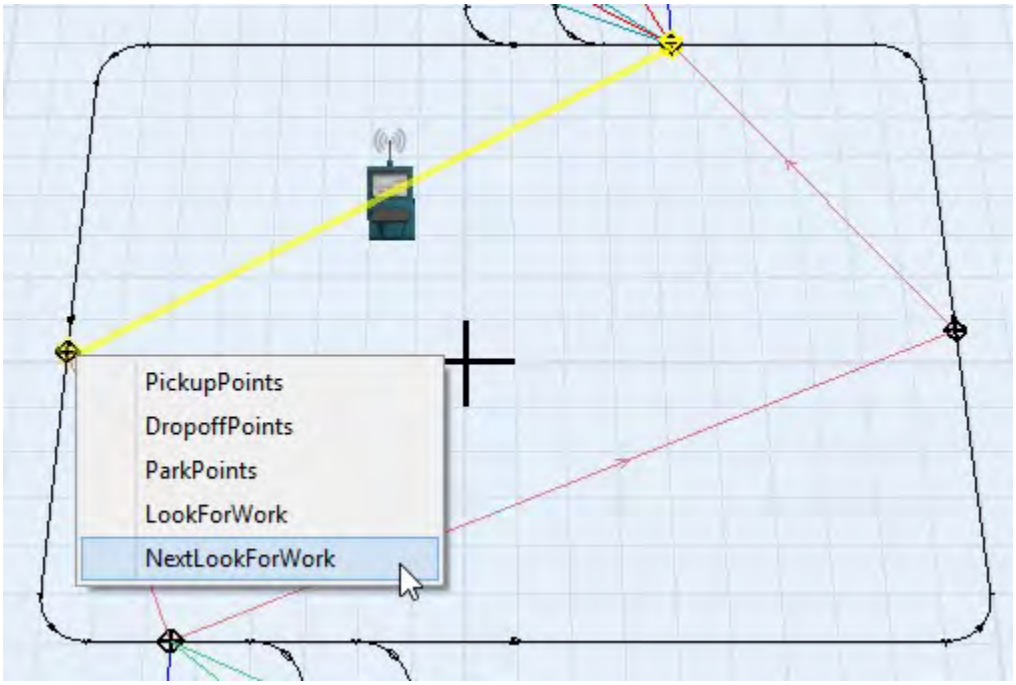
First, since we're going to be referencing the Control Points on the main loop, we should name them properly. Name the four Control Points on the main loop North, South, East and West respectively. For each Control Point on the main loop:

1. Click on the Control Point
2. In Quick Properties on the right, change the name to one of "North", "South", "East" or "West", corresponding to its location on the loop.



Create a Loop of NextLookForWork Connections

Using the "A" connect, create a loop of NextLookForWork connections. The loop should be:
North >> West
West >> South
South >> East
East >> North



Run the Model

Now run the model. The AGVs will now be dispatched to an available spur to drop off the items.

Finished

You're finished with this model.

AGV Lesson 3 Introduction

This lesson extends the previous lesson to take you deeper into AGV simulation concepts.

What You Will Learn

- How AGV speeds are determined
- How to park the AGVs
- How to dispatch to spurs in the AGV network for pick-up
- How to implement work polling
- How Control Areas Work

Time to Complete

This lesson should take approximately 15-20 minutes to complete.

AGV Model Overview

Lesson 3 will start where Lesson 2 left off. From that point we will add spurs to our network for parking AGVs, and for loading multiple items from the same location. Finally, we will define custom max speeds for the curved paths and spurs.

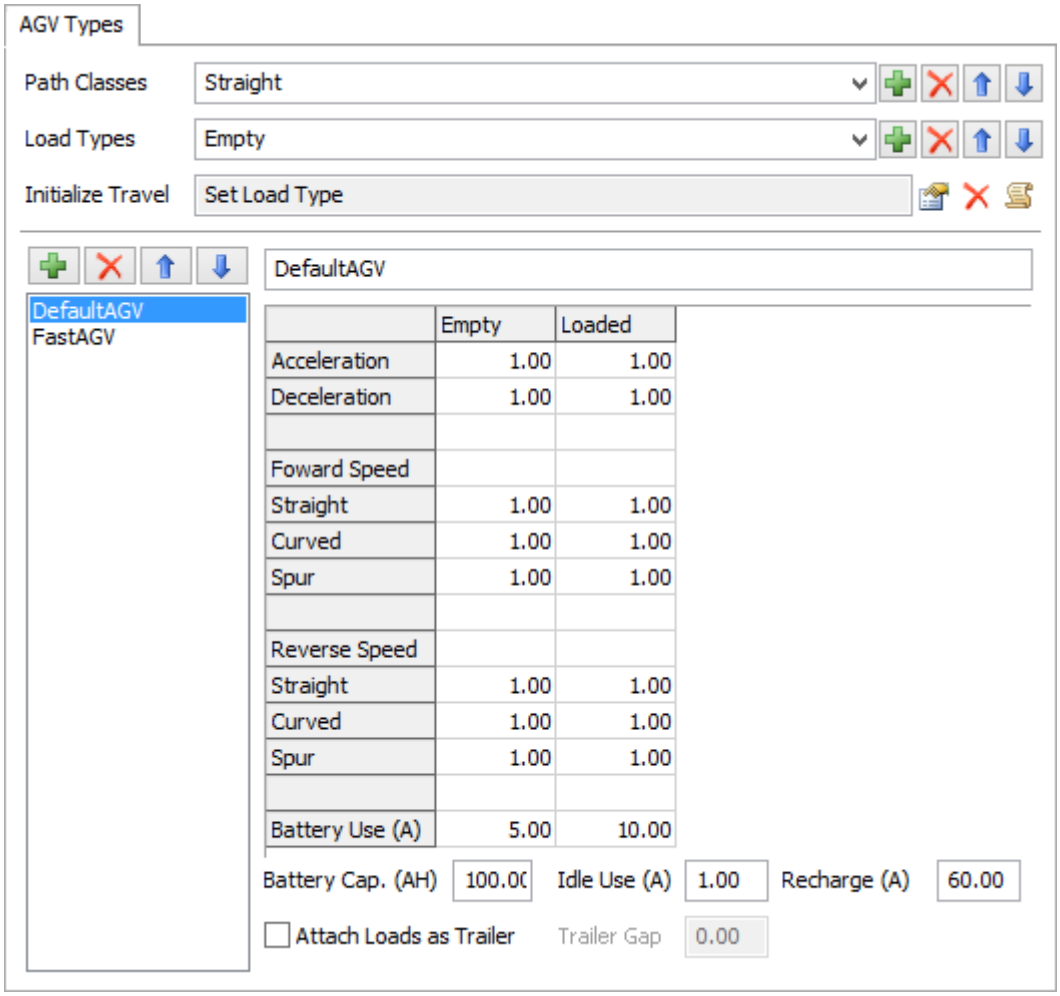
Click here for the Step-By-Step Tutorial.

New Concepts

AGV Types

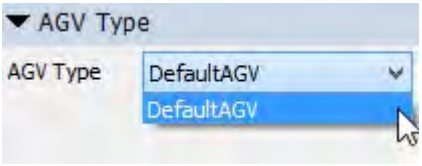
The AGV module includes a flexible method for determining AGV speeds and other AGV-specific attributes. First, the AGV network categorizes its member AGVs into AGV Types. An AGV Type correlates to a specific AGV's "make and model". An AGV vendor may sell several different lines of AGVs, some for transportation of heavy loads, some for medium loads and so on. With each of these models comes a spec sheet that defines the carrying capacity, max speeds, battery capacity, battery usage, etc. In FlexSim, an AGV Type would correspond to one of these vendor-specific AGVs.

By default, a FlexSim AGV model has only one AGV Type, called DefaultAGV. You can add, remove, and configure AGV Types by right-clicking on a Path or Control Point, and choosing AGV Network Properties. Then click on the AGV Types tab. Here you will see the list of AGV Types defined in your model, along with the specs for each AGV Type.



To define AGV Type for an AGV object in your model:

1. Make sure the AGV has been added to the AGV network with an "A" connect (there should be a red line drawn to its reset Control Point).
2. Click on the AGV.
3. In Quick Properties on the right, in the AGV Type panel, choose the desired AGV Type in the dropdown.



Note that since a default FlexSim AGV model only has one AGV Type, DefaultAGV will be the only option starting off. Again, you can add more AGV Types through the AGV Types tab.

You can also batch-define AGV Type for multiple AGVs by selecting them all, doing the above steps for one of them, and then pressing the popup button "Apply To All Selected."

AGV Type Specs

An AGV's acceleration and deceleration are determined by 2 factors:

1. Its AGV Type

2. What it is currently carrying (its Load Type)

An AGV's max speed is determined by 4 factors:

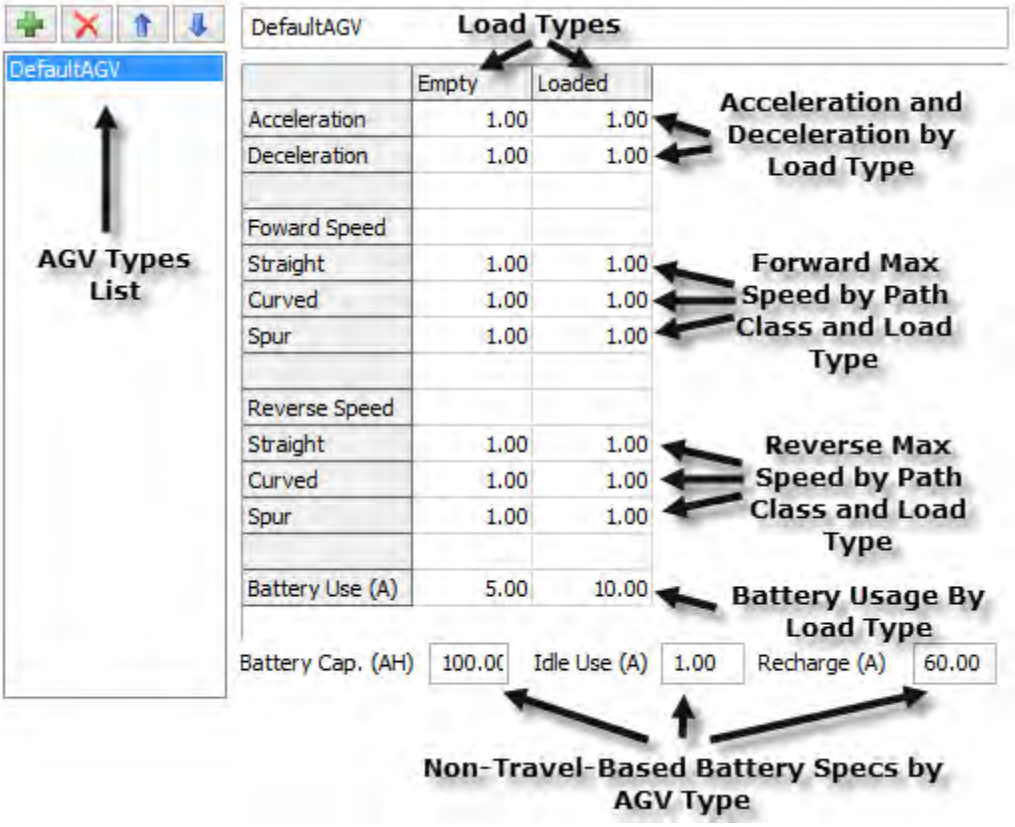
1. Its AGV Type
2. The type of path, or Path Class, it is currently traveling on
3. Whether it is traveling forward or backward
4. What it is currently carrying (its Load Type)

An AGV's non-idle battery usage, defined in Amps, is determined by 2 factors:

1. Its AGV Type
2. What it is currently carrying (its Load Type)

An AGV's battery capacity, idle battery usage, and recharge rate are determined solely by the AGV's AGV Type.

All of these factors are encapsulated in the AGV Type table. Each AGV Type has its own table.



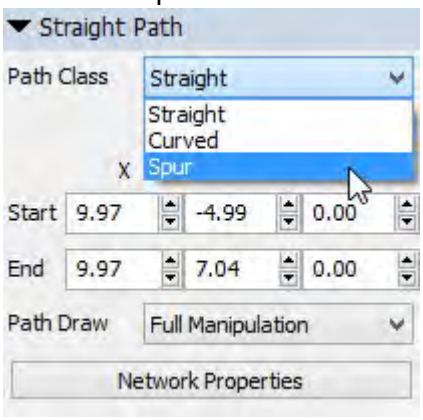
Load Types

Load Types are a user-defined list defining categories for what an AGV is carrying. This allows you to break out AGV speeds by the AGV's current load. Each time an AGV starts a travel operation, it will set its current Load Type. This is done in the Initialize Travel trigger. Here you can customize which Load Type the AGV should use for a given travel operation. You can add a Load Type, and then specify the acc/dec, speed and battery usage parameters for that Load Type in its corresponding column of the AGV Type table.

Path Classes

Path Classes are a user-defined list of path categories that allow you to break out AGV speeds based on the path the AGV is on. Each path in your model has an assigned Path Class. To change a Path's path class:

1. Click on the Path in the 3D view.
2. In Quick Properties on the right, in the Straight Path or Curved Path panel, choose the desired Path Class from the drop-down.



You can also batch-define the Path Class for multiple Paths by selecting them all, doing the above steps for one of them, and then pressing the popup button "Apply To All Selected."

Control Areas

Control Areas enable more fine-tuned control over who gets into and out of an area. Control Areas are especially useful in controlling traffic at intersection points.

Take for example an intersection where AGVs branch off to multiple spur locations.

When an AGV travels along a path, it looks ahead to its next Control Point and claims that Control Point before traveling to it. If the paths of two AGVs cross but do not need to claim the same Control Point, as in the picture above, then they may run into each other. This is where Control Areas can be used.

As an AGV looks ahead to claim its next Control Point, if the AGV's path enters a Control Area before reaching the next Control Point, then the AGV must additionally claim the Control Area before proceeding. If it cannot claim both the Control Point and the Control Area, then it will leave both unclaimed and wait until both can be claimed before continuing to the Control Point.

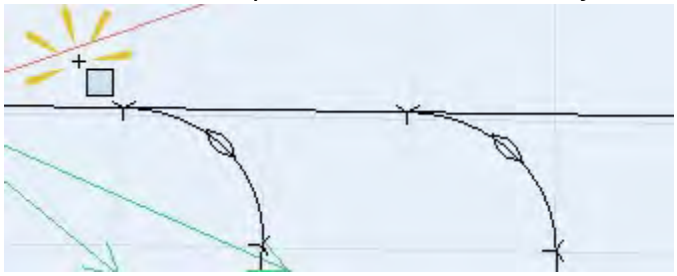
You can have any number of Control Areas between two Control Points. The AGV must claim all Control Areas between it and its next Control Point, plus the Control Point itself, before continuing to the next Control Point.

Creating Control Areas

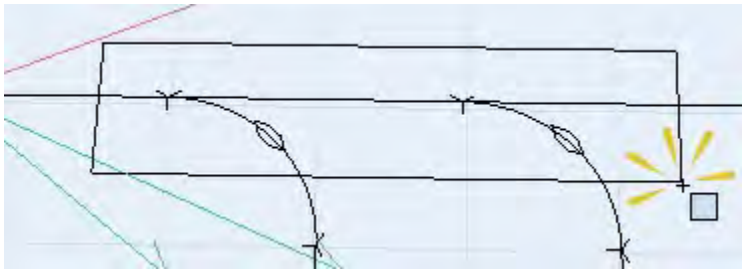
To create a Control Area:

1. Click on the Control Area tool in the library.

2. Click-release at the point in the model where you want to place a corner of the Control Area.



3. Move the mouse to the location in the model where you want the other corner of the Control Area
4. Click-release at the other corner's position.



Build the Model

Now build the tutorial model with the step-by-step instructions.

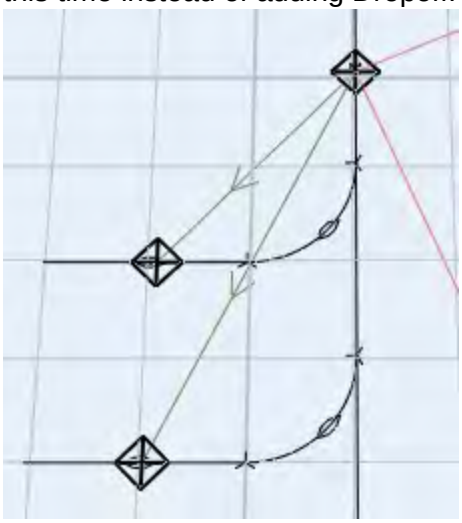
AGV Lesson 3 Step-By-Step Model Construction

Start with the finished model from Lesson 2. You will build this model in small incremental steps, running the model after each step is finished, so that you can understand the effects each of the changes you make has on the model.

Add Parking Spurs

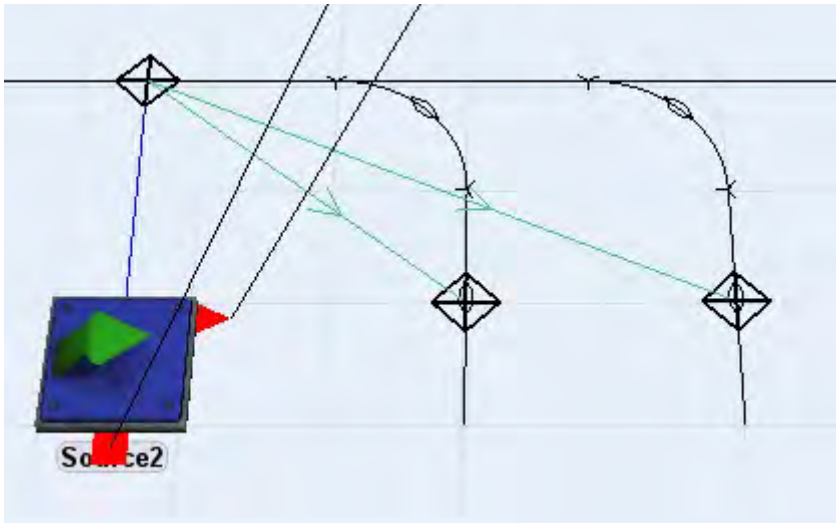
The AGV process flow that we added in lesson 2 includes logic for sending the AGVs to park. However, we must add ParkPoints connections to a control point in the NextLookForWork loop to define where AGVs can park. Otherwise, the AGVs will keep traveling around the NextLookForWork loop, never parking.

Using the same method as in Lesson 2, add two parking spur points beside the West control point. However, this time instead of adding DropoffPoints connections, add ParkPoints connections.



Add Pick-Up Spurs

Using the same method as in Lesson 2, add two pick-up spur points beside the Source object. However, this time instead of adding DropoffPoints connections, add PickupPoints connections.

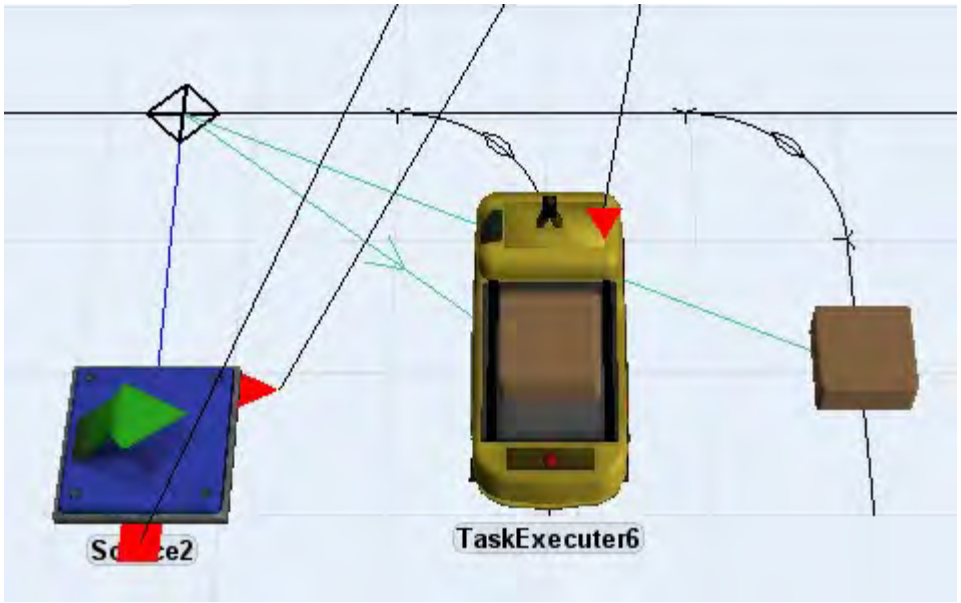


Change Source Inter-Arrival Time

In order to actually see the AGVs park, we'll need to change the source's inter-arrival time so they can take a break occasionally. Change the source's inter-arrival time to exponential(0, 100, 0).

Run the Model

Reset and run the model.
Now items will be dispatched to the individual spurs and AGV's will pick those items up from the spurs.



When there is no work to do, the AGVs will park in their designated parking spots.

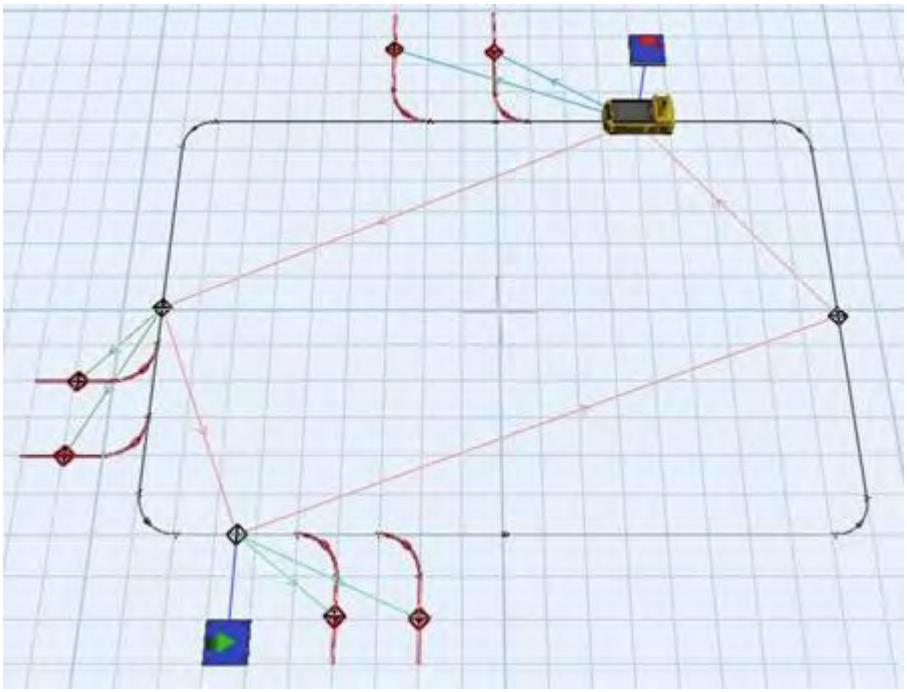
AGV Speed

Now we will define custom speeds for Straight vs Curved vs Spur Paths.

1. Right-click on a Path or Control Point and choose AGV Network Properties.
2. In the AGV Types tab, on the DefaultAGV type (the only one available), define speeds as follows for both forward and reverse:
3.

	Empty	Loaded
Acceleration	2	1
Deceleration	2	1
Straight	6	2
Curved	3	1
Spur	1.5	0.5

4. Click OK to close the AGV Network Properties window.
5. Shift-select the two pick-up spurs.
6. Control-select the two drop-off spurs, and then the parking spurs. This will add these spurs to the selection so that all spurs are selected.



7. Click on one of the selected paths.
8. In Quick Properties on the right, for Path Class, choose Spur.
9. Press Apply to All Selected.

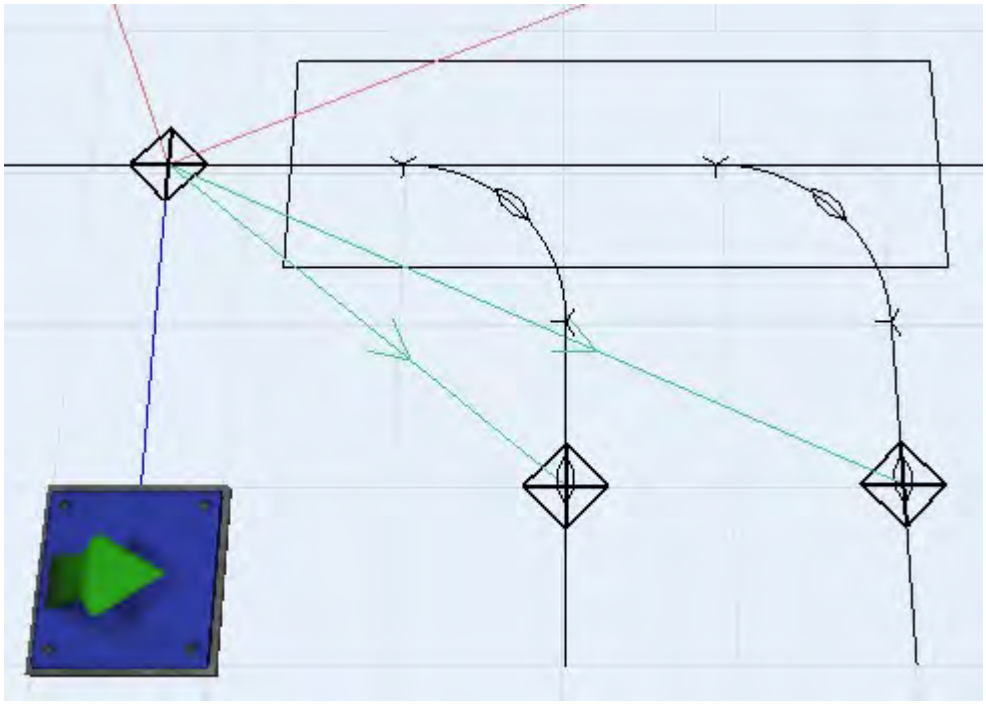
Run the Model

Now you should see the AGVs define their speeds based on the type of path they are on as well as whether they have a flow item or not.

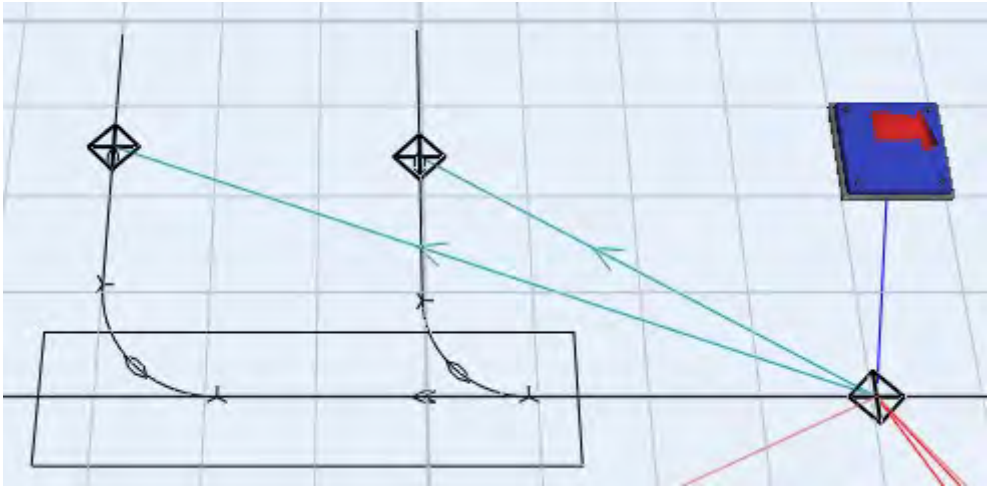
Avoiding Collisions

If you didn't already noticed before, now the model will make prominent the fact that the AGVs will occasionally collide with each other, primarily at the pick-up and drop-off points. This is because in this situation, the regular Control Point allocation mechanism isn't sufficient to prevent the collisions. Instead we need Control Areas.

1. Using the method described in the introduction, add a Control Area that covers all intersections at the



2. Add a Control Area that covers all intersections at the drop-off location.



Run the Model

Run the model again. Now AGVs will properly wait at the intersections so as not to run into each other. Note that you may need to move the Source and Sink Control Points back to allow enough room for the AGVs exiting the spurs.

Finished

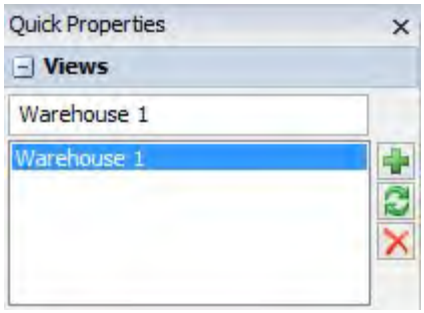
You're finished with this lesson. At this point it is a good idea to review the AGV process flow. This process flow is self-documented, so it will guide you through how it works, and give you ideas for customizations. Also, as a template process flow, its logic can be completely customized according to your needs.

Orthographic/Perspective View

Topics

- Saving Views
- View Settings
- Moving Around in the View
- Moving Objects
- Sizing Objects
- Connecting Objects
- Creating and Editing Selection Sets
- Editing an Object's Properties
- Right-Click Menu

Saving Views



The Views utility is found in the Quick Properties window when there are no objects selected in the 3D view. It allows you to capture the view's current camera position and add it to a list of views.

View Name - Rename the selected view by typing a new name in this field.

View List - Select a view from the list to rotate/move to that view.

- Captures the current view.

- Updates the currently selected view to the camera's current position/rotation.

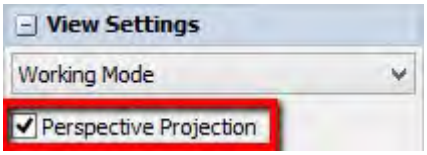
- Removes the selected view from the list.

Views can also be access through the right-click menu.

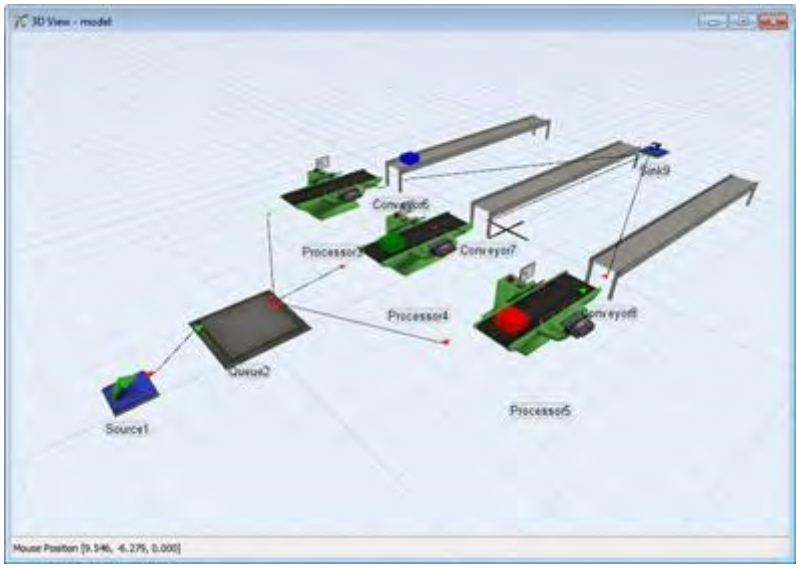
View Settings

For more available view settings, see the View Settings.

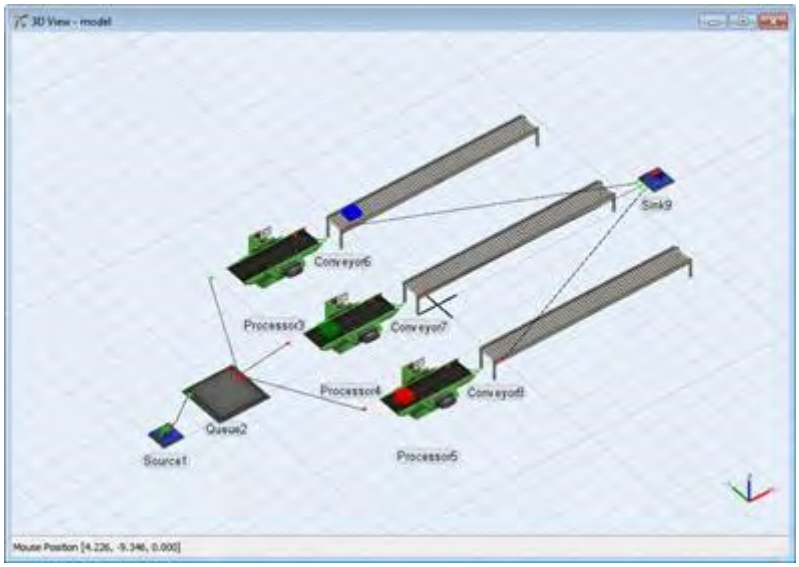
The 3D view may be toggled between orthographic and perspective modes by deselecting all objects in the 3D view and checking/unchecking the Perspective Projection option in the Quick Properties window.



Perspective projection gives the model a more 3D world feel.



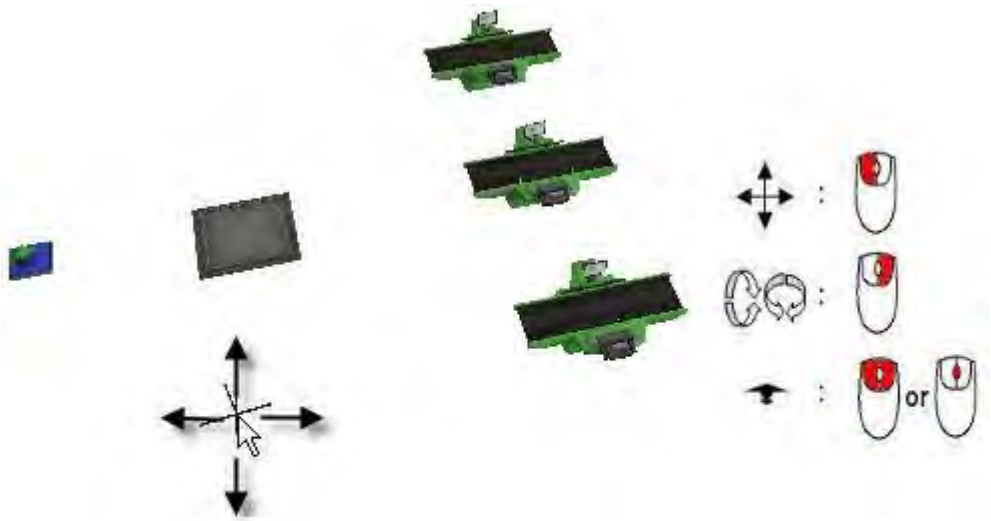
Where as the orthographic view has no depth.



For more information about changing the View Settings for the 3D View, see the View Settings page.

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.

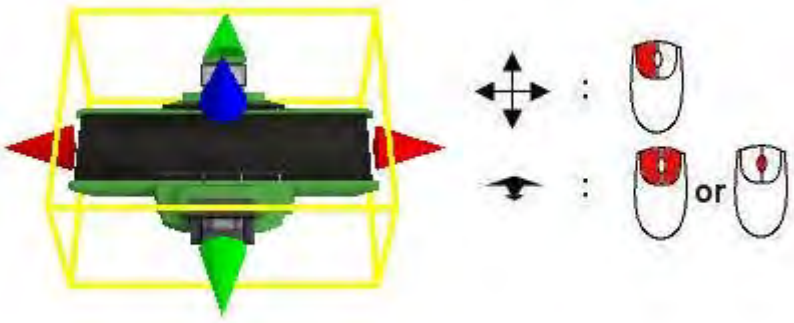


Holding down the alt key while clicking and dragging will ignore any objects clicked in the model. This makes it easier to move in the view without accidentally moving objects in the model.

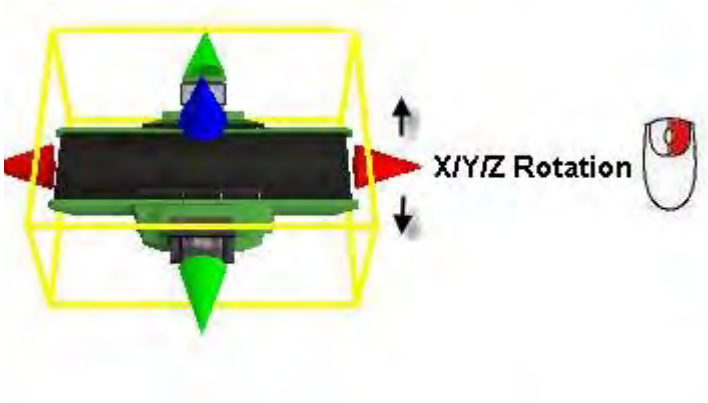
If you are working in a perspective view, you can also do a mouse guided fly through using the F8 key. Make sure the mouse is in the center of the window. Then press the F8 key. Move the mouse up and down to fly forward and backward. Move the mouse left and right to turn left and right. Once you are finished, click on the F8 button again to exit fly-through mode. It is sometimes easier to navigate if the view is configured as first person (from the view 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, click on one of the three axis arrows with the right mouse button and drag the mouse up and down.



Sizing Objects

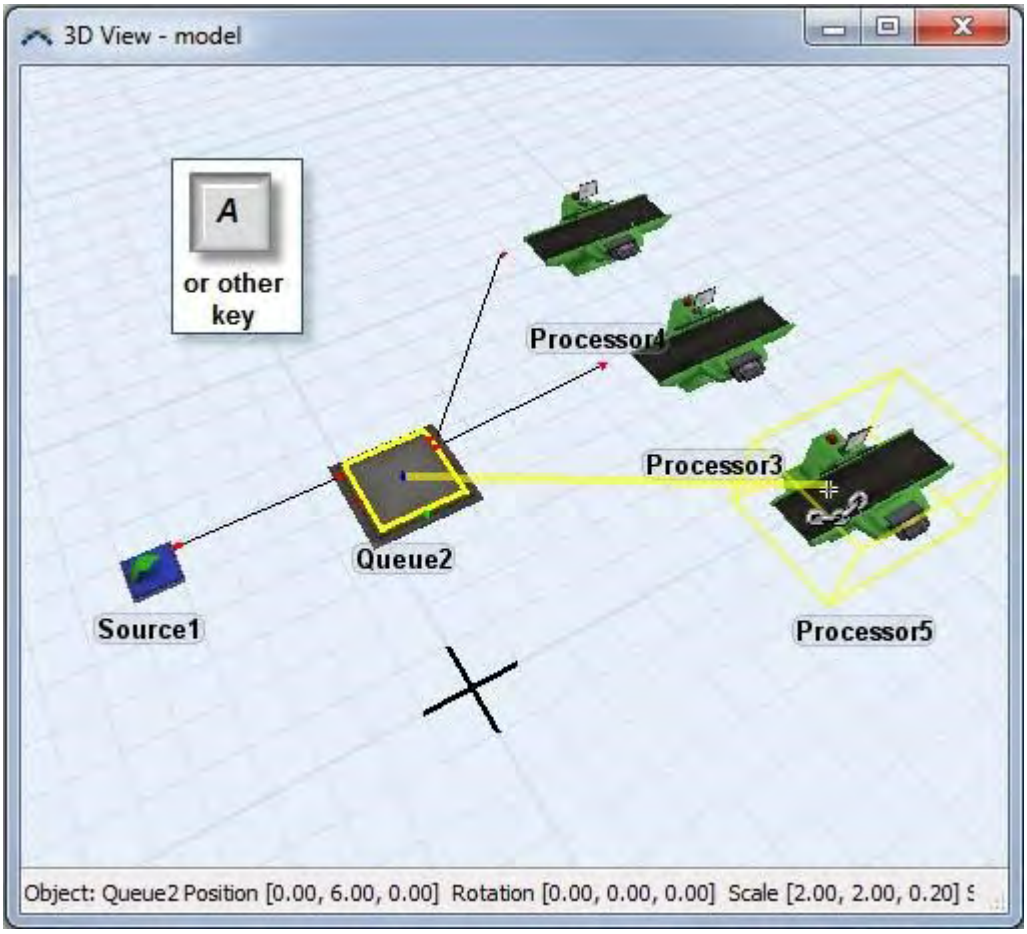
To change the object's size, click on one of the three axis arrows with the left mouse button and drag the mouse in the direction of the arrow. You can size the object in all three axes by 5% by pressing Ctrl and "L" to size up or "K" to size down. You can also size the objects in all three axes at once by holding both the left and right mouse buttons down on the object and clicking on one of the three axes and moving the mouse in the direction of the axis. If the three axis arrows are not showing, go to the Edit menu and select Resize and Rotate Objects.

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. For instance, the 'Q' key is used for disconnecting. These and others are described in detail in the keyboard interaction section.

Another way to connect objects is to use the Connect Objects mode on the main toolbar. Having this mode selected allows you to create connectionsa without holding the 'A' key.

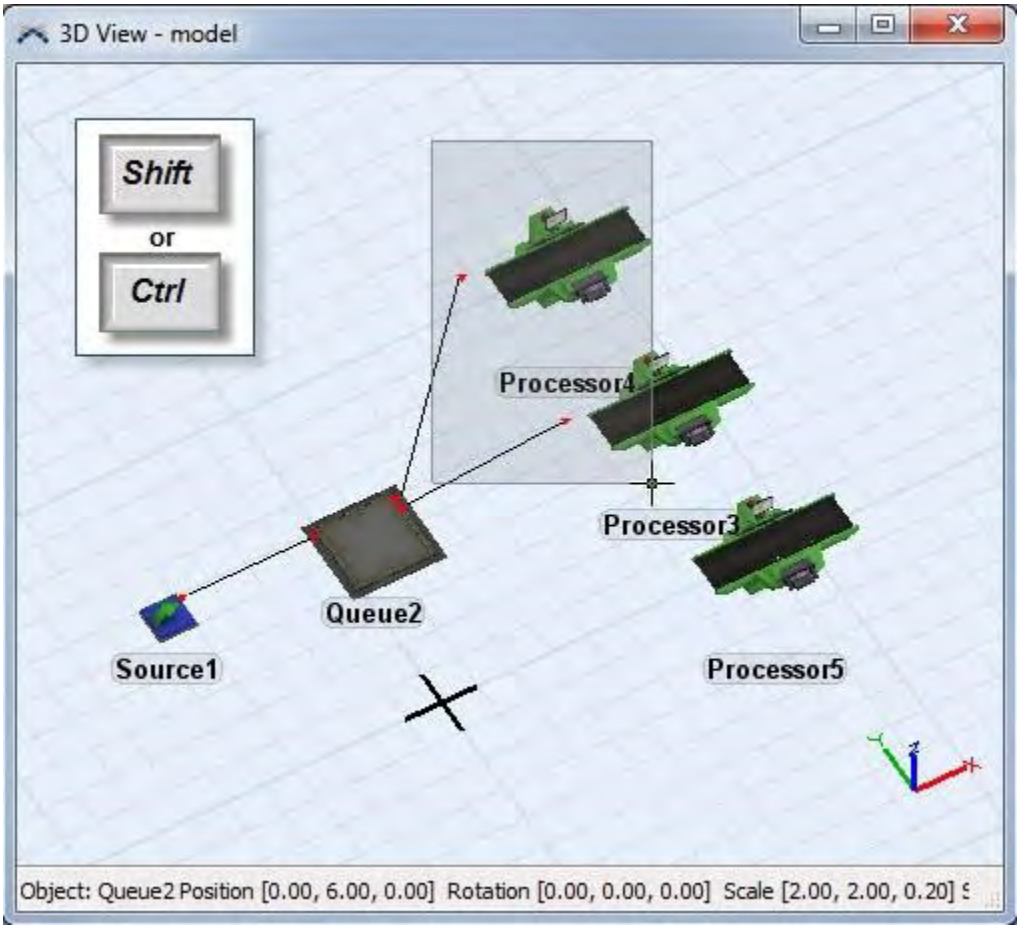
You can also create multiple connections in series. To do this, you hold 'A' or use Connect Objects mode  , click an object, then click another object, then click another object, and so on.



Creating and Editing Selection Sets

You can create selection sets to have operations apply to a whole set of objects. To add objects 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 or removes 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.

Another way to create selection sets is to use the New Selection mode on the main toolbar. This mode works the same as holding the Shift key down. The Toggle Selection mode works the same as holding the Ctrl key down.



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

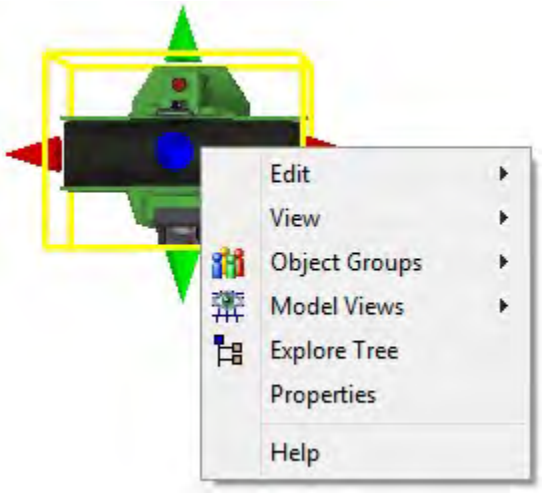
Once you have created a selection set, moving, rotating and scaling one of the objects will cause the other objects in the selection set to be moved, rotated or scaled as well. You can also perform several operations on the selection set from the Edit Selected Objects menu.

Editing an Object's Properties

To edit the properties of an object, refer to the Quick Properties window on the right side of your screen, or double click on the object or right click on it and select "Properties" from the popup menu to bring up more properties.

Right-Click Menu

Right click on the Perspective View or on an object to display this popup menu:

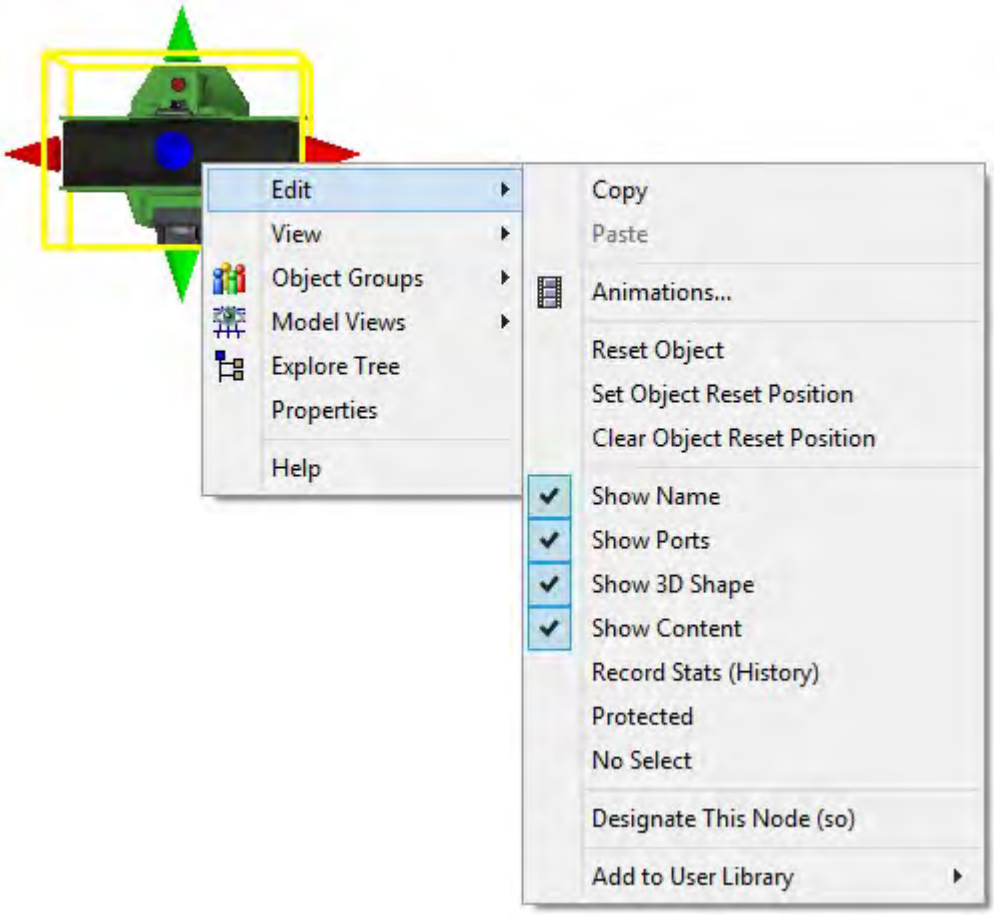


Explore Tree - Opens a Tree Window and displays the object in the Tree.

Properties - Opens the object's properties window.

Help - Displays this help page.

Edit



Copy - This copies the current item(s) to the clipboard.

Paste - This pastes from the clipboard into the current item (usually the model).

Animations... - Opens the Animation Creator to edit the object's animations.

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

Set Object Reset Position - This saves the object's current position so that whenever you press the reset button, the object will go back to that position (particularly useful for mobile objects such as transporters and operators).

Clear Object Reset Position - Clears the object's Reset Position.

Show Name - When this option is selected, the object's display name is always visible.

Show Ports - This toggles visibility of the ports and connections of the object(s).

Show 3D Shape - This toggles object visiblity in 3D views.

Show Content - This toggles flowitem visibililty within the object (i.e., flowitems in a Queue).

Record Stats (History) - This is another way to turn on statistics recording for the object.

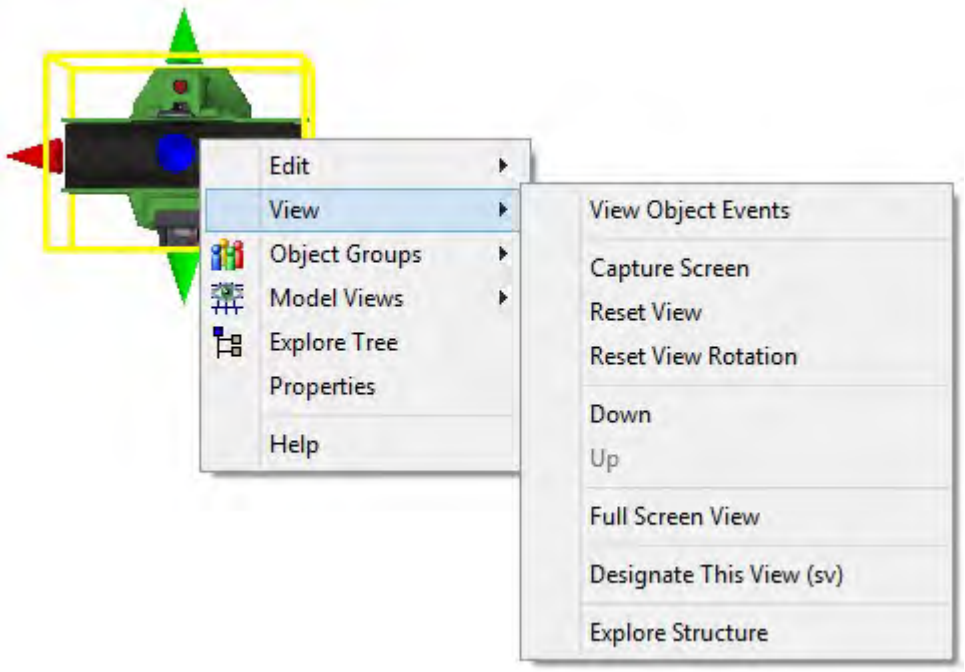
Protected - This locks the objects' location, size and rotation.

No Select - This toggles the object's No Select, making the objects unclickable in the 3D View (still accessible in the Tree Window).

Designate This Node (so) - This designates the object as the "selected object", which can then be referenced in code as so(). You will usually use this option for writing code in the script console. There can only be one so() at any time.

Add to User Library - This option adds the selected object to the a selected user library.

View



View Object Events - Opens the Event List and displays all the current events for that object.

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, which is focused at 0,0,0.
Reset View Rotation - This returns the camera rotation of the view to its initial position, which is looking straight down.

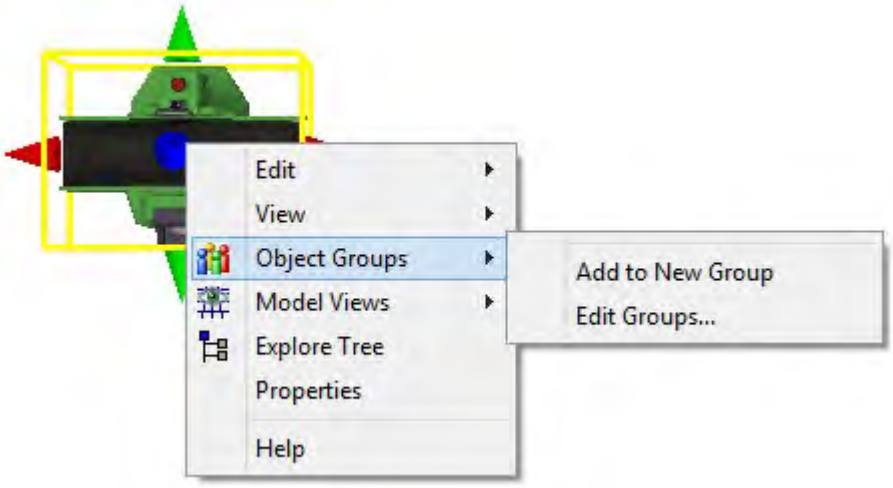
Down - This option causes the view to drill down into the selected object, allowing you to see what is going on in the "contents" of that object. The title bar of the window will show the name of the object that you are observing. This feature is particularly useful for Visual Tools that hold other objects.

Up - This option causes the view to return up out of a selected object. If you have drilled down multiple levels you will have to return up the same number of levels to reach the main view.

Full Screen View - This sizes the window to take up the full screen. The window's title bar and frame will not be shown, and any other Flexsim windows on that monitor, including the main window, will be hidden behind this view. To exit Full Screen View, right click in the view and select View > Out of Full Screen View.
Designate This View (sv) - This designates the window as the "selected view", which can then be referenced in code as sv(). You will usually use this option for writing code in the script console. There can only be one sv() at any time.

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

Object Groups

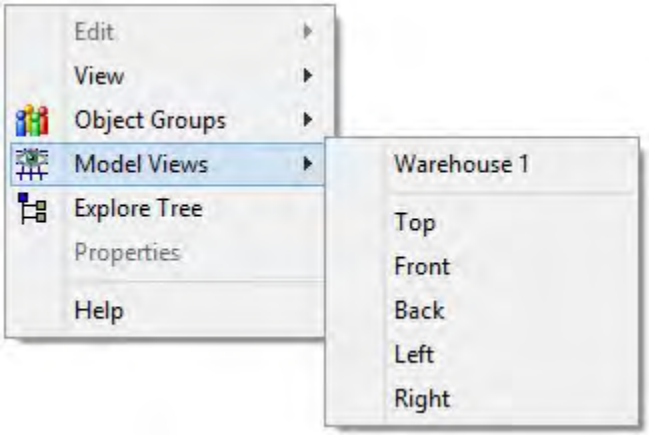


This list will dynamically update as you add groups. Each group name will be listed, and will become checkable, so you can easily add objects to any number of groups.

Add to New Group - This adds the object to a new group.

Edit Groups... - This opens the Object Groups utility. For more information see the Groups page.

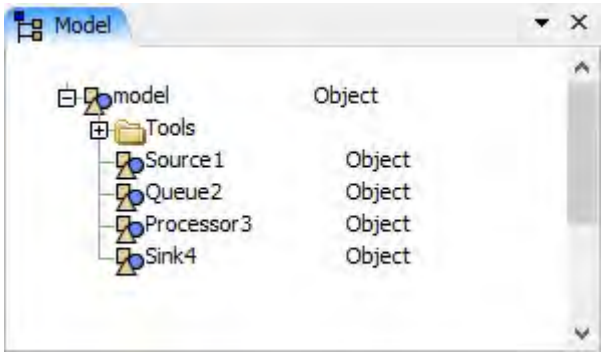
Model Views



This list dynamically updates to show all available views. The defaults are Top, Front, Back, Left, and Right. Selecting an available view will rotate the model to that view.

Custom views, the ones created through the Views utility, will appear at the top of the menu.

Tree Window



For information about the structure of the tree, see the Tree Structure page.

The Tree Window can be accessed from the Toolbar by clicking the Tree or from the View menu.

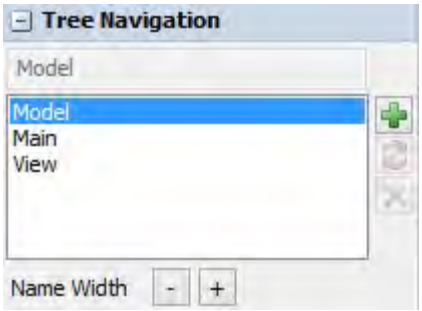
Navigating the Tree

To move around in the tree 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. You can also use the Tree Navigation panel from the Quick Properties, as described below, to jump around the Tree.

Quick Properties

When the tree window is active, the Quick Properties will change to display the Tree Navigation panel and the Search panel. If a node is selected in the tree, the Node Properties panel will also be displayed in the Quick Properties. Clicking on nodes with Object data will display similar results as clicking on them in the 3D view with the addition of tree panels.

Tree Navigation



This panel allows you to navigate between sections in the FlexSim tree.

- Model - A subset of the Main Tree, this contains all of the objects used in the currently open model.
- Main - The Main Tree contains many of the higher level functions in FlexSim.

- View - The View Tree contains all of the GUIs in the FlexSim interface.

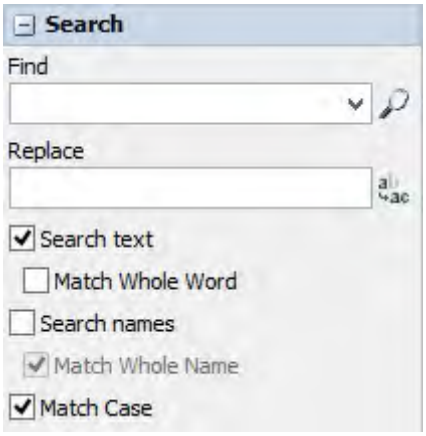
You also have the option of saving views from the Tree. By highlighting a node in the Tree and pressing the that node will be added to the list of preset views. This allows you to quickly jump back and forth between different sections of the Tree. Select a view and click the to remove it from the list. will update the selected preset view to the highlighted node.

Views that are saved are persistent even after FlexSim closes. Views that are added from the Model get saved into the Tools/TreeNavigation folder and wil be available to anyone who loads that model. Views that are added from anywhere else in the Main Tree or from the View Tree are saved in the User Prefs and will be available whenever FlexSim is open under your user.

Rename the preset view by entering a new name in the name field. Name

Width - This controls the name width of nodes in the Tree.

Search



The Search allows you to search through text and node names in the Tree. The Search will begin at the highlighted node in the active Tree Window. If no node is highlighted, the search will begin at the top of the active Tree Window. The search recursively searches through all subnodes and object attribute nodes. Replace allows you to replace all occurrences of the found text. This can be applied to text and/or node names.

- Finds the specified text (or press the Enter key in the field).

- Finds the above text and replaces it with the specified text (or press the Enter key in the field).

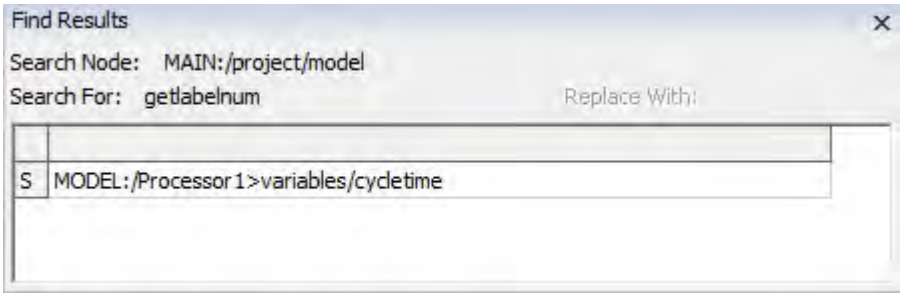
Search text - Searches through all text under the selected node.

Match Whole Word - If checked, only finds whole word matches of the text (meaning characters immediately before and after the search text must be non-alphabetic characters).

Search names - Searches through all node names under the selected node.

Match Whole Name - If unchecked, the search will return any node name that contains the searched text.
Match Case - If unchecked, the search will find all text/names containing the search text, regardless of capitalization. For example, if you search for "myvariablename", the results will still return all nodes containing "MyVariableName".

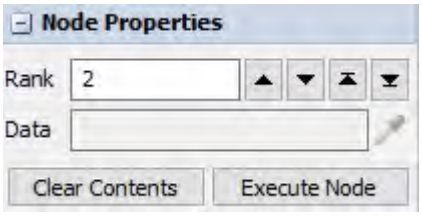
Once a search is begun, the Find Results window will appear:



Double-click on a result to view the text in a Code Editor window. You may also right click the result to Edit Code or Explore Tree.

Nodes toggled as FlexScript will have an S to the left of the node path. Nodes toggled as C++ will display a C and nodes toggled as DLL linked will display a D.

Node Properties



When a node is highlighted in the Tree, the Node Properties panel will appear, displaying rank of the node.
Rank - This specifies the rank of the highlighted node in its parent's tree. Enter a new rank number or use the up and down arrows to rerank the node.

Data - If the node has coupling data, the Data field will display the coupling's associated node. Use the to select the node to create a coupling with.

Clear Contents - Destroys all subnodes of the highlighted node.

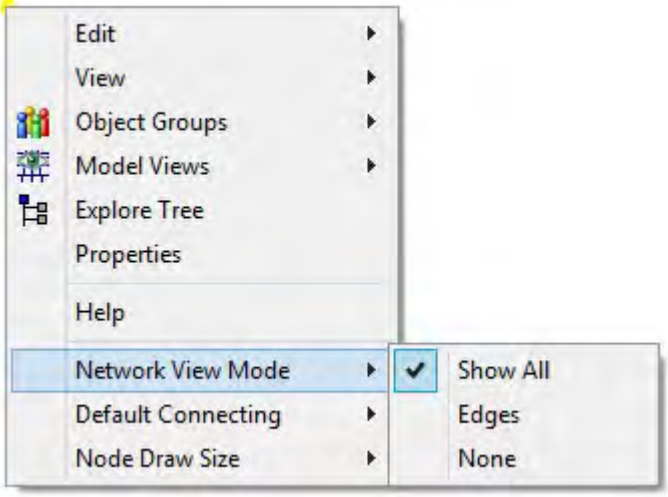
Execute Node - Calls nodefunction or executefsnode on the node, executing its associated FlexScript, C++ or DLL code.

Travel Networks Utility

The Travel Networks Utility has been moved since Version 6.0.2 and can now be found by dragging a Network Node into your model and right clicking on it.

Network View Mode

To change the way NetworkNodes are viewed in the 3D view, right-click on a NetworkNode and choose Network View Mode:



Alternatively, you can hold the X key down and left click on a NetworkNode to toggle between these three modes.

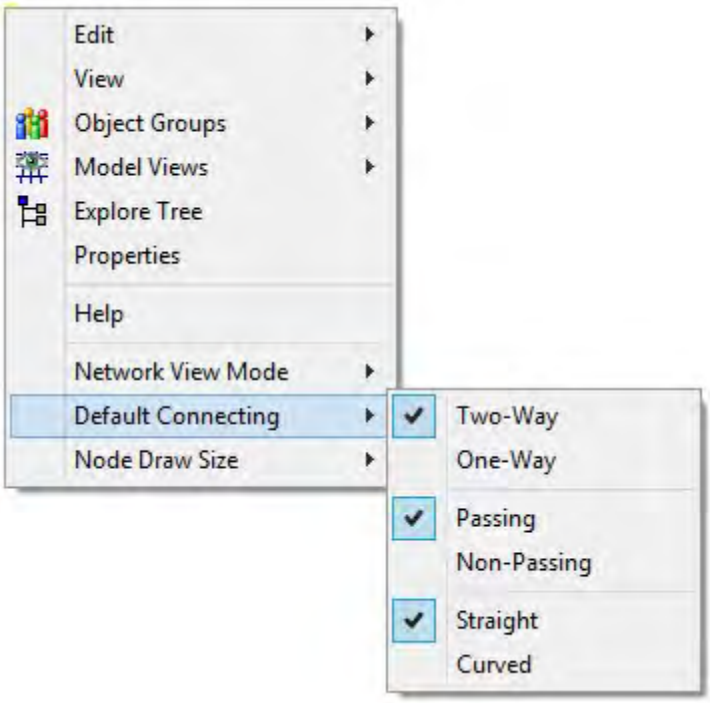
Show All - Displays the NetworkNodes, the edges between them, and the direction arrows.

Edges - Displays the edges between NetworkNodes only.

None - Hides all NetworkNodes and edges except for the NetworkNode that was right-clicked on.

Default Connecting

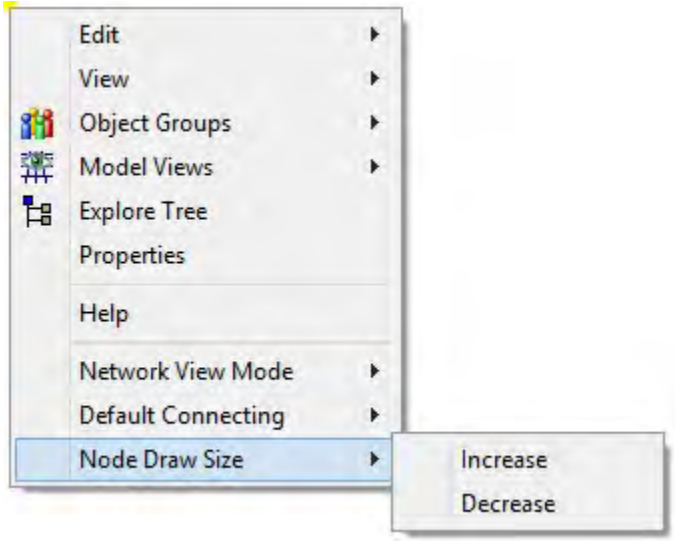
Default Connecting specifies how new connections will be made between NetworkNodes when you do an 'A' drag. These parameters may be changed on each individual connection after it has been created.



You can select default connecting as two way or one way, passing or nonpassing, straight or curved.

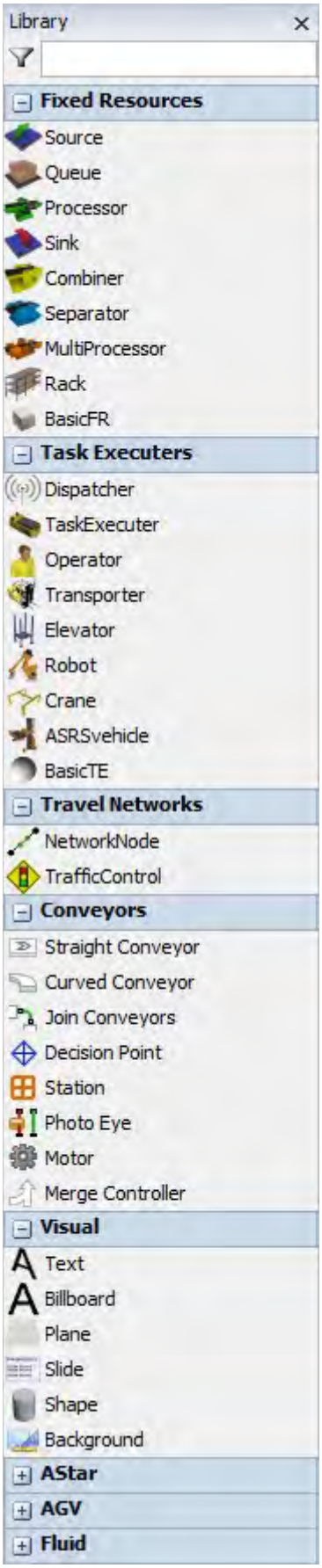
Node Draw Size

You can also change the size that network nodes are drawn by right clicking on the node and choosing Node Draw Size Network nodes are drawn a specific pixel size. They are unaffected by the zoom of the camera. That is why they appear to get larger as you zoom out and smaller as you zoom in relative to the other objects in your model.



Alternatively, you can left-click a NetworkNode and press the Ctrl+L or Ctrl+K key to increase and decrease NetworkNode sizes respectively.

Library Icon Grid


Library
Fixed Resources
Source
Queue
Processor
Sink
Combiner
Separator
MultiProcessor
Rack
BasicFR
Task Executers
Dispatcher
TaskExecuter
Operator
Transporter
Elevator
Robot
Crane
ASRSvehicle
BasicTE
Travel Networks
NetworkNode
TrafficControl
Conveyors
Straight Conveyor
Curved Conveyor
Join Conveyors
Decision Point
Station
Photo Eye
Motor
Merge Controller
Visual
Text
Billboard
Plane
Slide
Shape
Background
AStar
AGV
Fluid

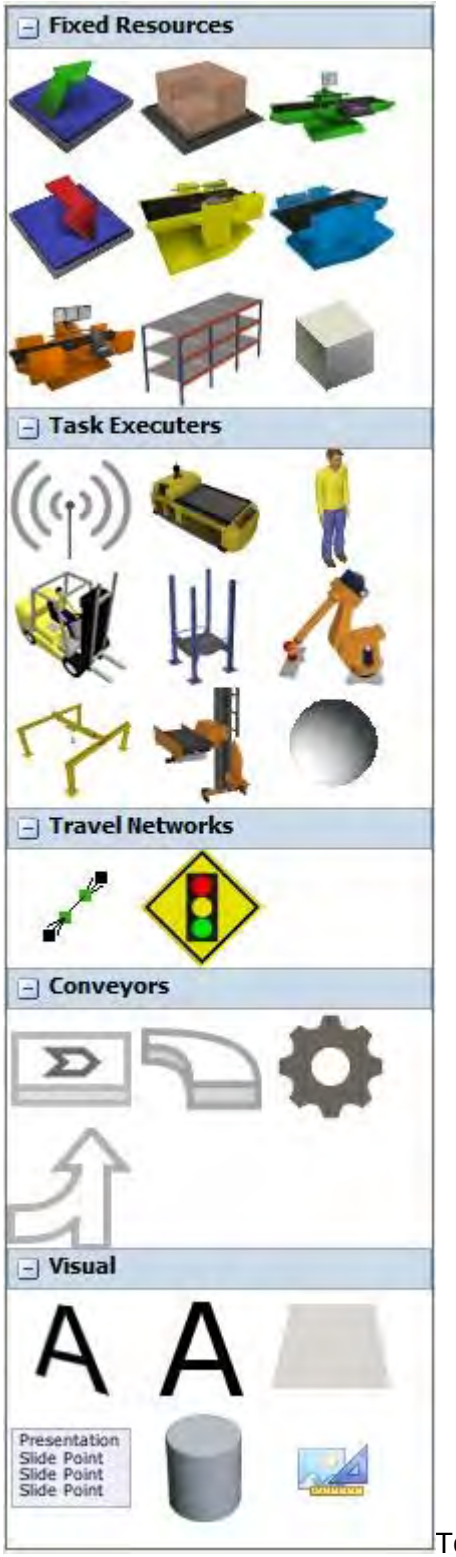
The library icon grid lets you drag objects into your model from one of FlexSim's standard libraries, or from custom-made libraries. If the grid is not already visible, you can access it through the View menu > DragDrop Library.

The objects are sorted into groups that can be expanded or collapsed by pressing the plus or minus buttons next to the group of objects. FlexSim's standard groups are Fixed Resources, Task Executers, Travel Networks, Visual, and Fluid. If you have loaded additional libraries, these libraries will be displayed at the top of the list in their own groups.

Creating Objects

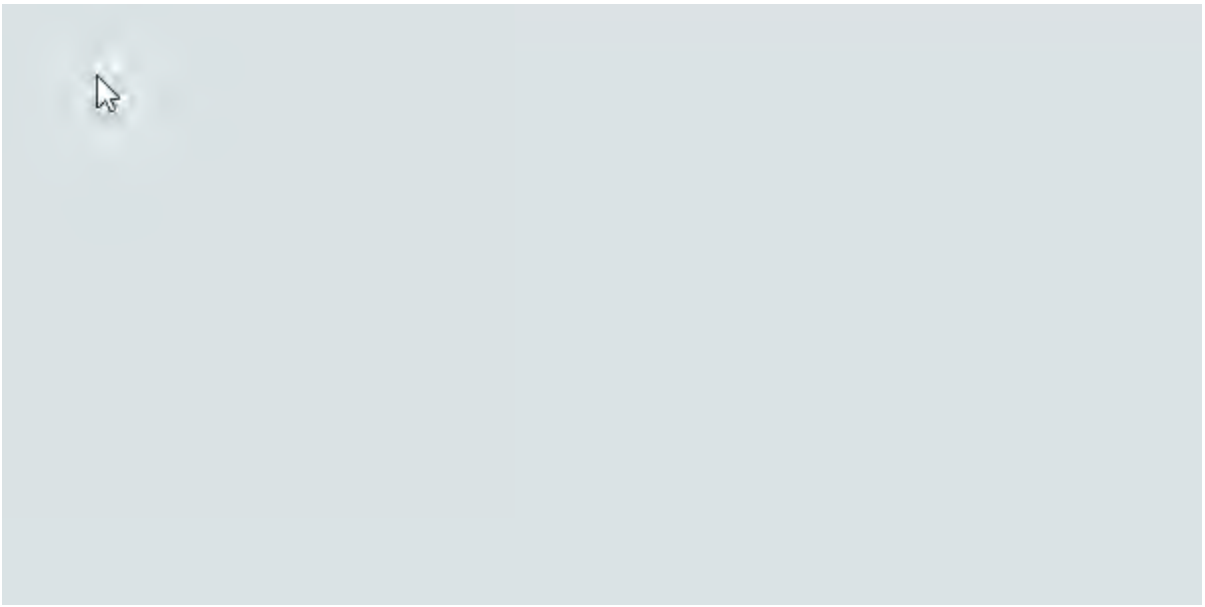
There are three methods for creating objects from the Library:

- Click and hold on the object you would like to add to your model, drag it over a 3D view and release the mouse button at the location you would like to drop the object into the model.
- Click and release on an object in the Library to enter Create Mode. Then click anywhere in the 3D view to create new objects. To exit create mode, right click or press the Escape key.
- Use the Quick Library in the 3D View.



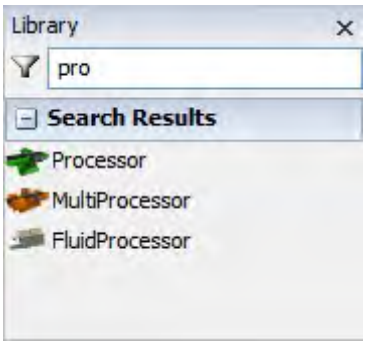
To display the Quick Library you can:

- Double click in empty space in the 3D view (not on an object).
- A-Connect or S-Connect from an object already in the 3D view to empty space. This will create the selected object and connect it to the previous object.



Filtering

The Library Icon Grid can be filtered by name by entering text in the field.



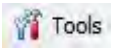
Filtering works for all Context Sensitive displays.

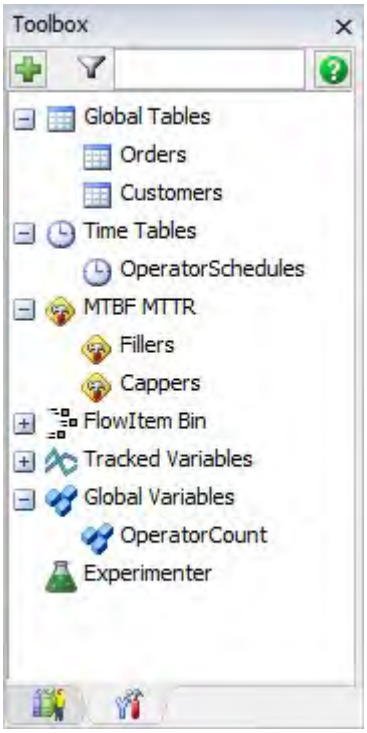
Context Sensitive

The Library Icon Grid is context sensitive. It will changed based on what the modeler is currently working on. It will display 3D shapes when in the FlowItem Bin and Animation Creator, code builder blocks while in the Code Editor and dashboard widgets while editing Dashboards.

FlexSim Toolbox

FlexSim's Toolbox window is the central location where you manage your model's tools, such as Global

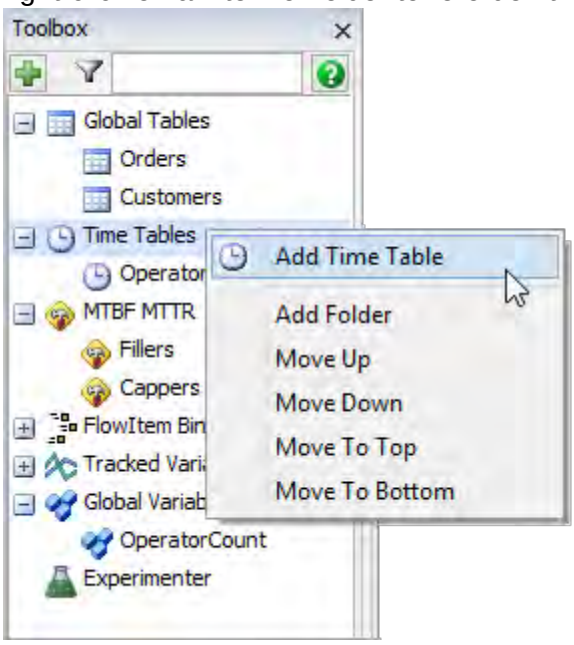
Tables, Time Tables, Dashboards, etc. You access the Toolbox by choosing View > Toolbox from the main menu, or by choos ing the 



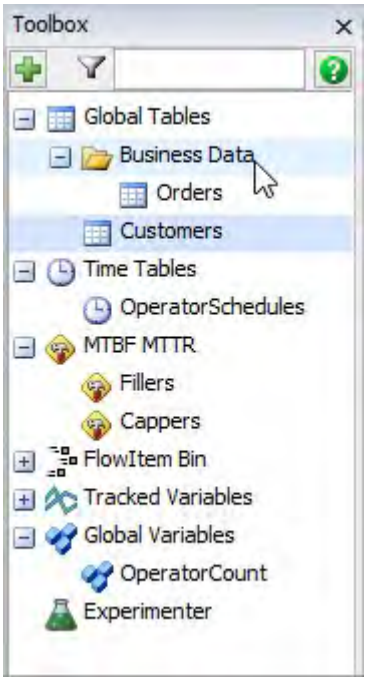
button from the main toolbar.

In the Toolbox you can:

- Click on the ✚ button at the top of the window to add a tool to the model.
- Double-click on an item to go to its properties.
- Right-click on an item or folder to re-order it in the list, add a new folder, or add a new tool.



- Drag items up and down to reorder them, or drag them into folders to organize them hierarchically.



- Select an item and hit Delete to delete it from the model.
- Click in the filter edit box at the top of the window and type in the name of a tool you are looking for to filter by that name.
- Slow-lick twice on the name of an item to rename.

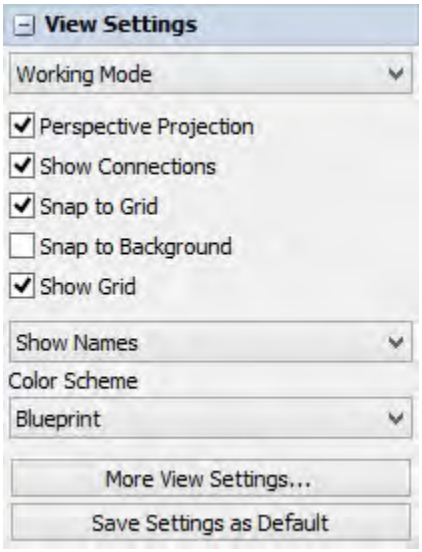
Although add-on modules may include additional tools, by default you can manage the following tools from the Toolbox:

- Dashboards
- Excel Import/Export
- Experimenter
- FlowItem Bin
- Global Tables
- Global Variables
- Graphical User Interfaces
- Model Background
- Model Triggers
- MTBF/MTTR
- Presentation Builder
- Reports and Statistics
- Time Tables
- Tracked Variables
- User Events
- User Commands
- Video Recorder
- Visio Import

View Settings

This dialog box can be opened by selecting View > View Settings from the main menu or clicking More View Settings... from the Quick Properties window. View Settings lets you configure the view's visual display settings. These settings apply only to currently active window and will be lost if that window is closed. If a window other than an 3D view is active, this window will be grayed out.

The Quick Properties window will display a few of the most common View Settings whenever a 3D view is active and no objects are selected in the view.



Mode - This option gives you two view setting presets. Switching to Presentation mode turns of Connections, the Grid, and Object Names.

Perspective Projection - This option toggles the 3D view from being a Perspective View, to displaying as an Orthographic View. The Perspective View looks more real-world where as the Orthographic view has no depth.

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.

Snap to Background - If this box is checked, objects will automatically move to the nearest background snap point when they are moved or resized in the model. For more information, see Model Background.

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

Color Scheme - These presets modify the 3D view colors.

Save Settings as Default - As described above, 3D View Settings are not saved and will be lost when the 3D view is closed. This will save your current View Settings as default for any new 3D views that are opened.

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.

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.

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

Grid

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.

Snap to Background - If this box is checked, objects will automatically move to the nearest background snap point when they are moved or resized in the model. For more information, see Model Background.

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

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 Color - This value sets the color of the grid lines.

Names

Show Names - This pull-down list allows you to choose whether names and stats are shown or hidden in the 3D view.

Name Style - This pull-down list allows you to choose where the name of the object will be drawn. Choose either below the object or above the object.

Names Color - This value sets the color of object names displayed in the 3D view.

Other Settings

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.

1st 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.

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.

Sync Views - 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.

VR Mode - Checking this box will cause the view to be rendered on a connected virtual reality headset. FlexSim works with Oculus Rift and HTC Vive VR headsets. The view will also render and handle inputs from VR motion controllers (Oculus Touch or HTC Vive controllers) if this box is checked. If you are using an Oculus Rift, ensure that Unknown Sources is checked in your Oculus Settings.

Floor Z - This specifies the height (in model units) of the floor. This is useful when creating multi-floored models. Setting the Floor Z to the height of your model's 2nd floor will allow you to quickly and easily drag objects into the 3D view and have their Z position set to the Floor Z value.

Light Sources

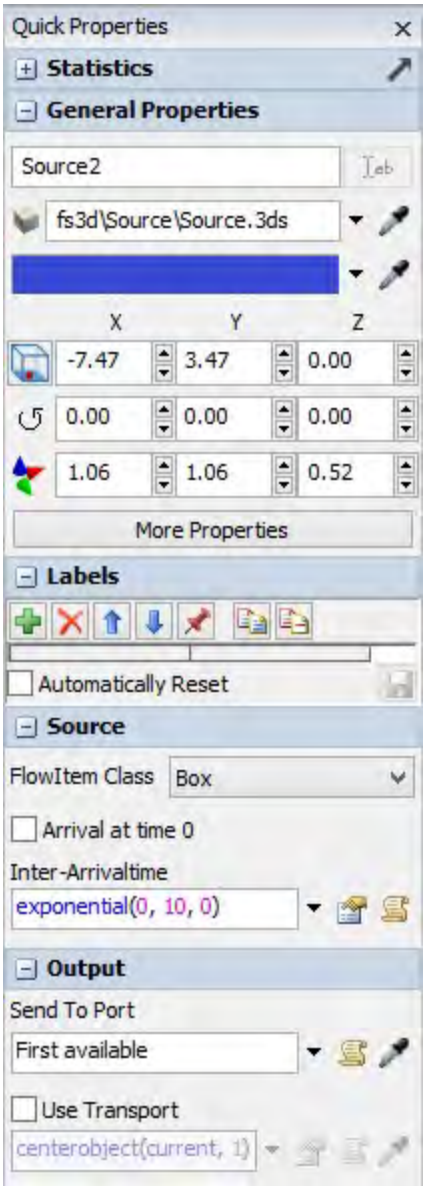
These controls allow a user to add, edit and create light sources for the view window

Ambient Color - This value sets the color of light that is applied to all faces equally regardless of location.
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 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.



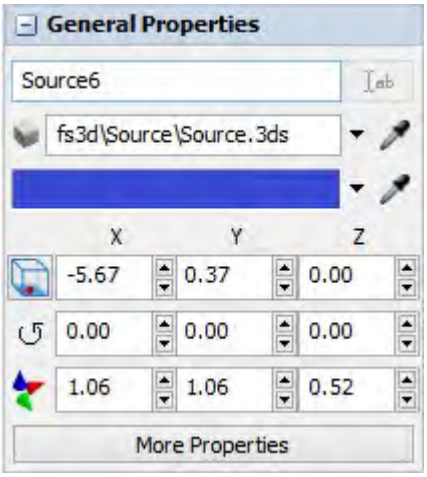
The Quick Properties window is a context sensitive window. Depending on what window you currently have active and what object is selected, the panels in the Quick Properties will change. We will not cover these context sensitive panels in this topic. For more information on these panels, see one of the following pages:

- Animation Creator
- Dashboard
- FlowItem Bin
- Orthographic/Perspective View
- Tree Window
- Edit Selected Objects

If the Quick Properties have been closed, you can open them through the View menu.

The above image shows the Quick Properties when *Source1* is clicked on in the 3D view. The panels display the most commonly used properties for the highlighted object. The Statistics, General Properties, and Labels panels will display for most objects. For more detailed information on other panels, see the object's Property Window.

Note on selected objects: Whenever you make a change to a highlighted (yellow) object's properties using the Quick Properties, that change will be automatically applied, if possible, to all selected (red) objects.



The General Properties panel is where you can quickly edit the visuals and position of an object.

Name

Name - This value is the name of the selected object. The button to the right will allow you to apply the name to all selected objects.

3D Shape

3D Shape - This option lets you change the 3D shape displayed by the object.

Color

Color - This option lets you change the color of the object.

Position

Position Reference Point - This option lets you change the point on the object from where the object's position will be calculated by the position fields in this panel. For more information see Position Reference Point

Position - These options lets you change x, y, and z positions of the object.

Rotation - These options lets you change x, y, and z rotations of the object. Size

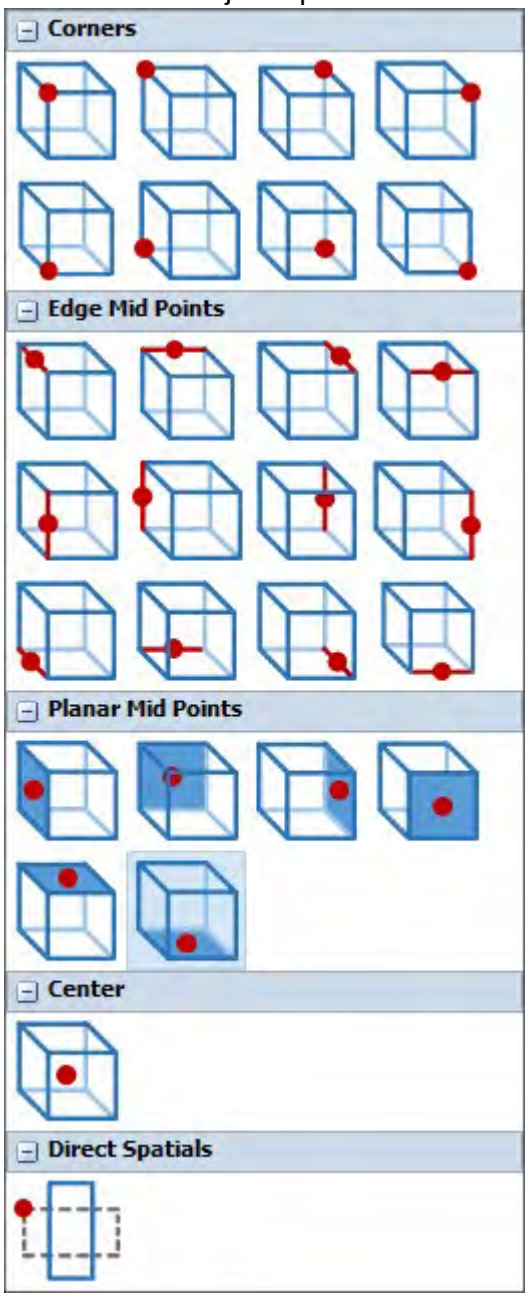
- These options lets you change x, y, and z sizes of the object.

More Properties

More Properties - This button opens the full properties window of the object.

Position Reference Point

The Position Reference Point button  will open a popup letting you specify the point on the object from where the object's position will be calculated by the position fields in this panel.

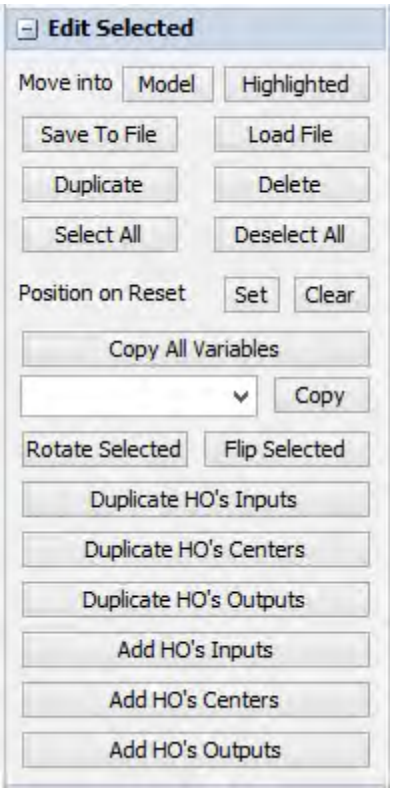


By default the position reference is set to the mid point at the bottom of the object. The position values displayed by Quick Properties are calculated from the bottom center of the object. If you set the position of the object to (0, 0, 0) the object will be centered at the origin of the model. If, however, you set the position reference to be one of the bottom corners and then set the position to (0, 0, 0) that corner of the object would be set at the origin.

Direct Spatials - The Direct Spatials option works differently than all the above options. Selecting this option will display the position of the object as the spatial values of the object stored by the software. These spatial values are calculated off the top left corner (when viewed from above) on the bottom of the object before any

rotations are applied. Because this value is not very useful after an object has been rotated, this mode is not normally recommended.

This mode is useful when you want direct access to the values stored by the software, for example, when using the position of the object as an experimenter variable.



This window can be accessed through the Quick Properties menu . It appears when there are selected objects in the view and the user highlights one of those selected objects.

It offers several options that are performed on the currently selected set of objects in that view window. To select a group of objects, hold the shift or control key down and drag a selection box around a group of objects. Objects in the selection set will have a red box drawn around them. The view's currently highlighted object (the last object you clicked on) will have a yellow box drawn around it.

Edit Selected

Move into Highlighted - 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.

Move into Model - This option moves the selected objects (the ones with the red box) into the model directory.

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. This can also be done by highlighting one of the selected objects in a view and pressing the Delete or Backspace key.

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.

Set Reset Position - This option sets the reset position of each object in the selection set to its current position. When the model is reset, all objects with recorded reset positions will be moved, rotated, and sized to that position.

Clear Reset Position - This option removes the reset position of each object in the selection set. Copy All Variables - This option will copy the value of all variables the highlighted object has onto each object in the selection set. If a selected object does not have a particular variable, the variable is not created and the object is simply skipped for that variable.

Copy - This option will copy the value of the selected variable of the highlighted object has onto each object in the selection set. If a selected object does not have that particular variable, the variable is not created and the object is simply skipped.

Rotate Selected - This option will rotate the selection set around the z-axis at a user-defined axis point. Upon clicking this button, the cursor will change and the user will be prompted to pick a point in the model view.

Flip Selected - This option will flip the selection set across a line in the x-y plane defined by the user. Upon clicking this button, the cursor will change and the user will be prompted to pick two points in the model view, which become the endpoints of the flip line.

Duplicate HO's Inputs - This option destroys all the selected objects' input ports and creates copies of the highlighted object's input port connections to all objects in the selection set.

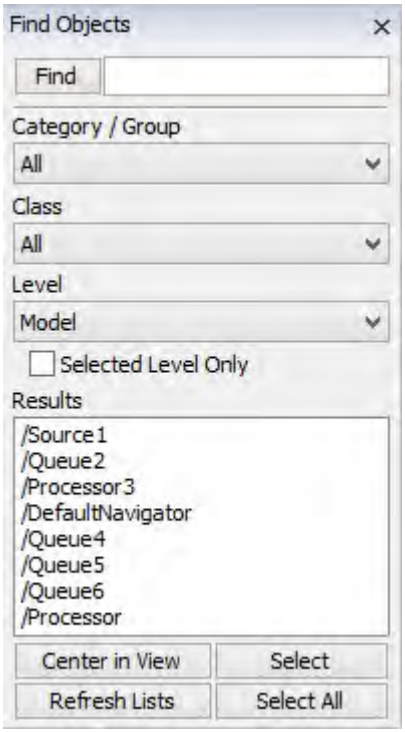
Duplicate HO's Centers - This option destroys all the selected objects' center ports and creates copies of the highlighted object's center port connections to all objects in the selection set.

Duplicate HO's Outputs - This option destroys all the selected objects' output ports and creates copies of the highlighted object's output port connections to all objects in the selection set.

Add HO's Inputs - This option adds copies of the highlighted object's input port connections to all objects in the selection set.

Add HO's Centers - This option adds copies of the highlighted object's center port connections to all objects in the selection set.

Add HO's Outputs - This option adds copies of the highlighted object's output port connections to all objects in the selection set.



This window can be accessed through the View menu > Find Objects.

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

Find By Name

Type all or part of a name into the edit at the top and press Find to find an object in the 3D View or Tree view. This field is case-sensitive. "Queue" is different than "queue". Find will search through all of the objects in your model to find any object name that contains the specified text. ie searching for "Processor" would find both "Processor3" and "FluidProcessor4".

Find By Category/Group/Class

You can also select the library category or group (see Groups), and/or the class type, and/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 in the list and press Center in View to go to that object in the 3D/Tree view.

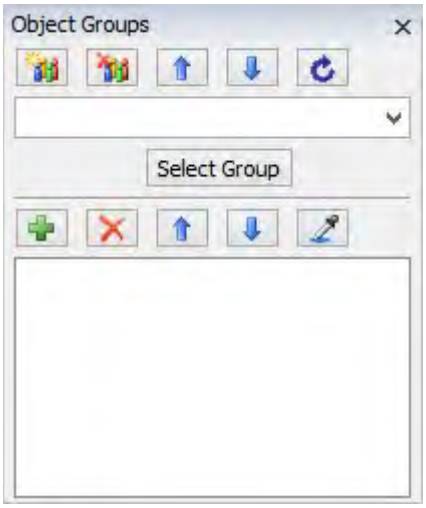
Selected Level Only - Check this box to display only objects in the selected level. For example, if you have a Visual Tool in the model that contains three processors and you select the Level as Model, those three Processors will not be displayed in the Results list.

Center in View - Centers the object selected in the Results list in the 3D or Tree View.

Refresh Lists - Refreshes all lists including the Results List (useful when you've added an object to your model).

Select - Selects (red box) the selected object from the Results list.

Select All - Selects (red box) all of the objects in the Results list.



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

Group information is stored both in the /Tools/Groups folder and on each of the individual objects. A node is added to the object attributes called Groups with a coupling to the associated group. Objects may be included in multiple groups.

- Creates a new empty group.

- Removes the currently selected group.

- reorder the group.

- Reloads the list of groups.

Group Dropdown Menu - All groups are listed in this drop down menu. The name of the currently selected group is editable.

Select Group - Selects all the objects in the group (red box).

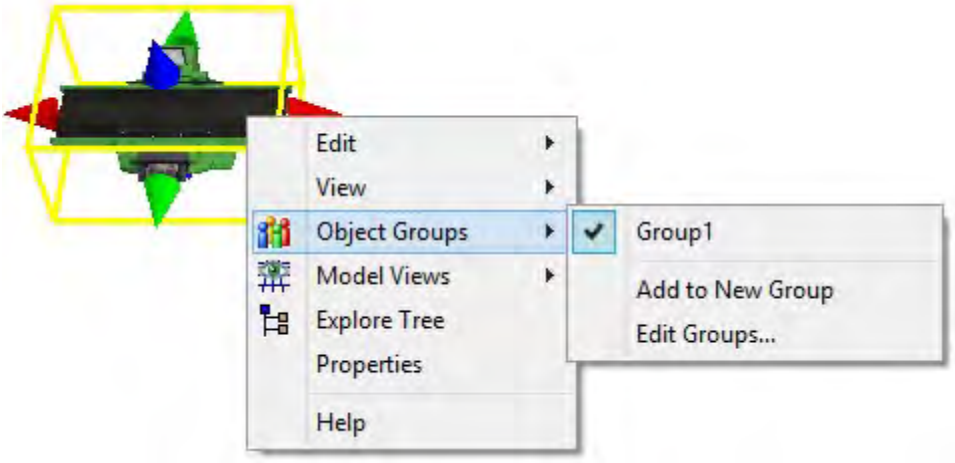
- Opens a menu allowing you to add currently selected items to the group or set the group to match the current model selection.

- Removes the selected(s) object from the group.

- reorder the members in the group.

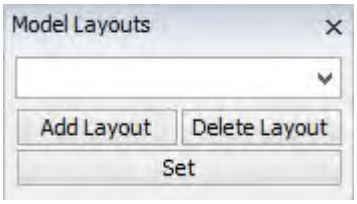
- Click to enter "Sample" mode then select an object in the 3D View to add it to the group.

You may also add objects to groups through the right-click menu of the 3D View.



The current list of groups will appear at the top of the menu. Each group name will be checkable, so you can easily add objects to any number of groups

Add to New Group - Creates a new group and add's the object to the group. If multiple objects are selected (red box) this will add those objects to the group as well. Edit Groups... - Opens the Groups window.



This window can be accessed through the View menu > Model Layouts.

This utility lets you create and edit certain layouts for your model. Objects may be moved around in the view in order to create different layouts for different scenarios. 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 window lets you do that. It is also useful for testing different potential layouts of your facility.

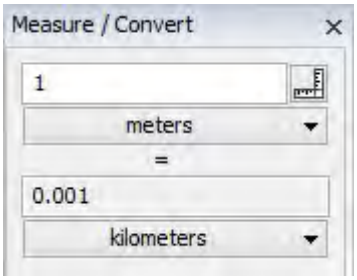
Layout information is stored on each of individual objects. A node is added to the object attributes called Layouts. For each created layout, a copy of the object's current spatial information (position, size, rotation) is stored.

Layout List - Displays the list of current Model Layouts. Select a layout to immediately update your model that layout. You may rename a layout by entering a new name in this field.

Add Layout - Adds a new layout. When you add a layout, it will save the current layout of the model.

Delete Layout - Removes the selected layout.

Set - This will update the selected Model Layout to match the current layout of the model.



This window can be accessed through the View menu > Measure Convert.

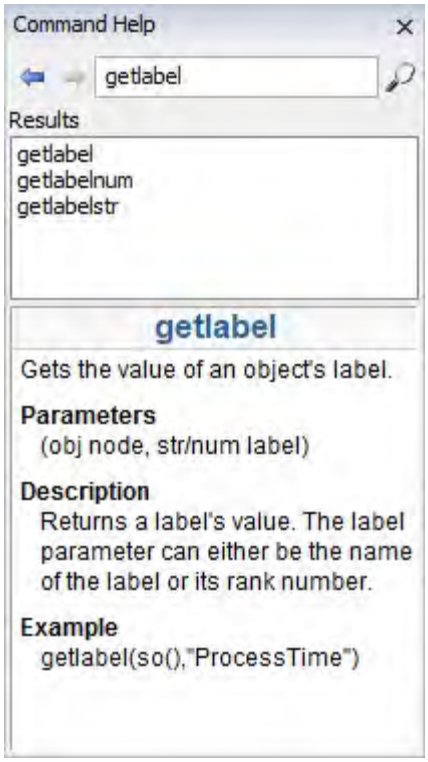
The Measure/Convert tool allows you to take measurements of distances in your model as well as to convert length/time units. By default, the Measured units will be the model units as defined in the Model Settings window will be selected.

- Click this button to enter "Measure" mode. You can then click anywhere in the 3D view to define your start point, and click again to define the end point. The resulting distance will be displayed to the left.

In the first drop down (shown above as meters), you can select the units to "Measure" with or select a length, time or length per time unit to convert. You may type in your own value to the top field to make a conversion. The bottom field will be the result. The bottom drop down can be changed to convert to different units.

The Command Helper is a quick reference for flexscript commands. It can be accessed by two methods:

1. From the Help menu.
2. By pressing F1 while hovering the cursor over a flexscript command.



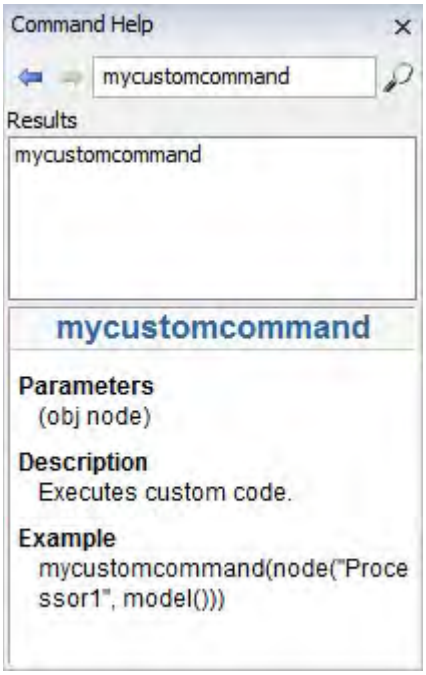
- Go to the previous command topic.

- Go to the next command topic.

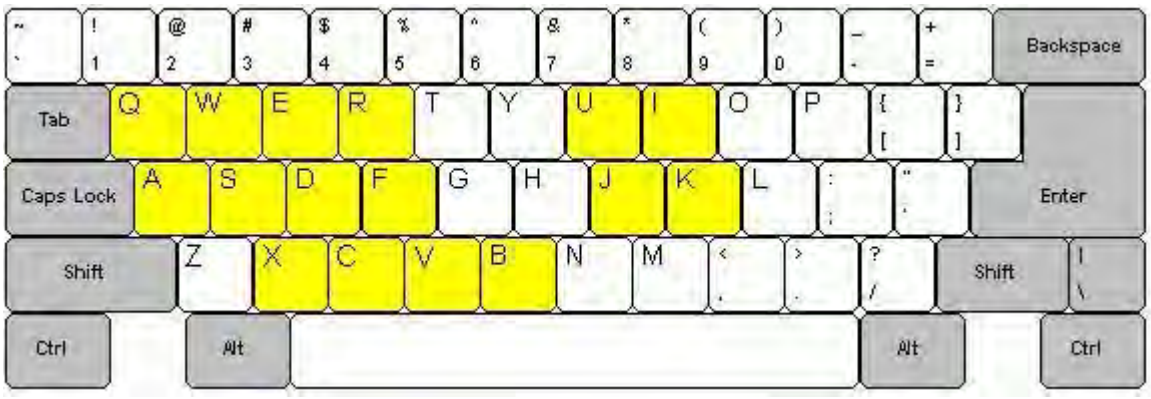
- Search the command li st.

Results - Displays search results.

User Commands that the modeler creates will also appear in the Command Helper and in the Command Reference.


Command Help
mycustomcommand
Results
mycustomcommand
mycustomcommand
Parameters
(obj node)
Description
Executes custom code.
Example
mycustomcommand(node("Proce
ssor1", model()))

When you are working in the 3D 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.



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 travellers, 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, click on an empty area in the 3D view and make sure the Show Connections button is checked in the Quick Properties window. 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 travellers, 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.

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. 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.

Ignore Objects

Hold the ALT key down to cause all objects in the 3D view to ignore any mouse clicks. This can be useful for moving around in a view with a lot of objects. You can also click and hold the middle mouse button and drag to move around the view, ignoring all objects.

Hot Keys / Accelerators

Ctrl + Space - Start and stop the model run.

Ctrl + Left Arrow - Reset the model.

Ctrl + Right Arrow - Step to the next model event.

Ctrl + Up Arrow - Increase the simulation run speed.

Ctrl + Down Arrow - Decrease the simulation run speed.

Ctrl + F - Find text in the open view.

Ctrl + H - Find and replace text in the open view.

Ctrl + C - Copy the selected object(s) to the clipboard.

Ctrl + X - Cut the selected object(s) to the clipboard.

Ctrl + V - Paste the object(s) in the clipboard.

Ctrl + T - Open a new tabbed document window (based on the currently active document window, 3D or Tree only)

Ctrl + Tab - Switch to the next window tab.

Ctrl + Shift + Tab - Switch to the previous window tab.

Ctrl + L - Scale the selected object(s) up by 5%.

Ctrl + K - Scale the selected object(s) down by 5%.

Ctrl + Shift + D - Add a keyframe to the presentation builder.

Ctrl + W - Close the active document window or floating window.

Ctrl + Alt + E - Switch Environments. If an environment is currently active, exit the environment, otherwise, enter the Minimal environment.

F1 - Open the Command Helper and search for the selected text.

F11 - Switch in and out of Full Screen Mode (3D View).

P - Takes a screen shot of the active view and saves it to projects folder/screenshots (adds pointer data if highlighted a node in the tree)

Tree Window Shortcuts

The following are available in the Tree Window:

Spacebar - Insert a new node after.

Enter - Insert a new node into.

Del - Delete selected node(s).

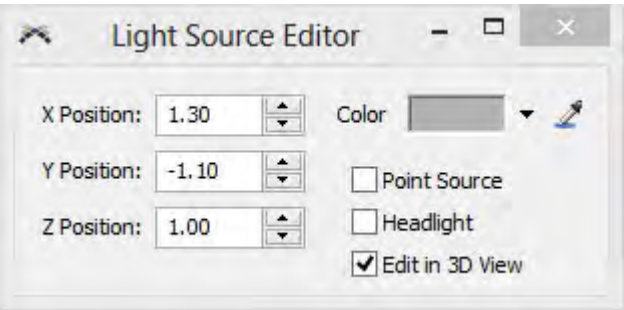
N - Add number data to the highlighted node.

T - Add string (text) data to the highlighted node.

O - Add object data to the highlighted node.

P - Add pointer data to the highlighted node. Shift + Del - Delete node data.

Light Source Editor



This dialog can be accessed by opening the View Settings and clicking the Edit button in the Light Sources section

X Position - This number is the position along the X axis of the model where this light source is located.

Y Position - This number is the position along the Y axis of the model where this light source is located.

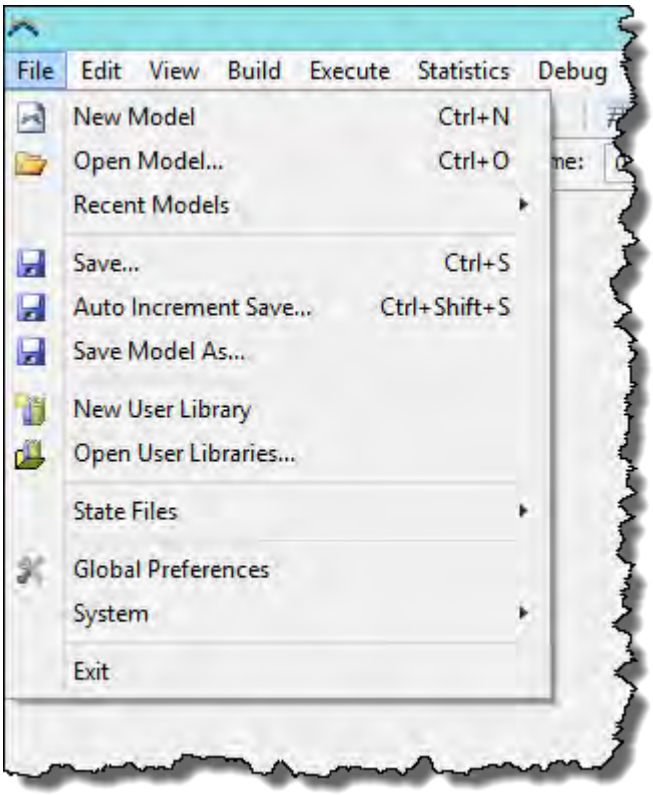
Z Position - This number is the position along the Z axis of the model where this light source is located.

Color - This allows the user to choose the color of the light from this light source.

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.

Headlight - If this box is checked, the light light source will move with the camera acting as a headlight. Edit in 3D View - If this box is checked, a lightbulb object will be displayed in the 3D view and allow you to move the light source to a desired position.

File Menu



New Model - This option clears the current model so that a new one can be created. (A warning will be displayed asking if you would like to save your work.) The result is a blank FlexSim model, except that all opened user libraries will remain intact.

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.

Recent Models - Shows a list of the latest FlexSim models to have been opened on this computer, with the most recently opened model at the top of the list.

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

Auto Increment Save... - This option saves the current model file (.fsm extension) and appends an underscore and a number to the file name. The number increments each save so that the file name is unique.

Save Model As... - This option allows the user to save the model to a file. The most common file that is created is a FlexSim Model file (.fsm); the entire contents of the model tree will be saved. The two other file saving options are for a FlexSim XML file (.fsx – for more information see FlexSim XML) and a Tree file (.t – effectively identical to a .fsm file).

New User Library - This option creates a new user library and adds it to the library pane. For more information on custom libraries, see Creating Custom Libraries.

Open User Libraries... - This option loads one or more saved libraries, adding them to the library pane. 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. For more information on custom libraries, see Creating Custom Libraries.

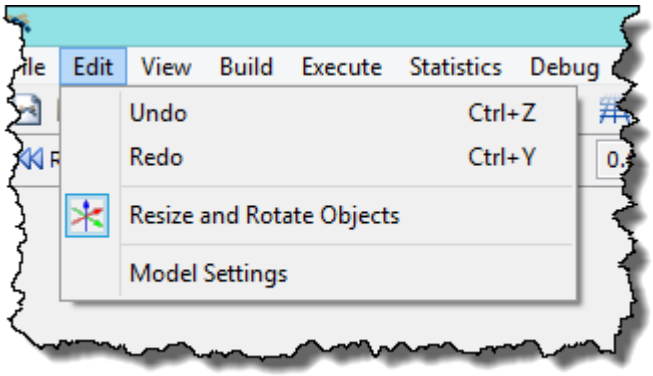
State Files - This option allows the user to save the state (current model run) of the model, or load a saved state model to continue the run. Saving a state is helpful when your model is in the middle of a simulation run and you want to save it in its current state (all flowitems remain where they are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

Global Preferences - Opens the Global Preferences window, which contains settings and options that affect the entire FlexSim application and are independent of any specific model file. For more information, see Global Preferences.

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.

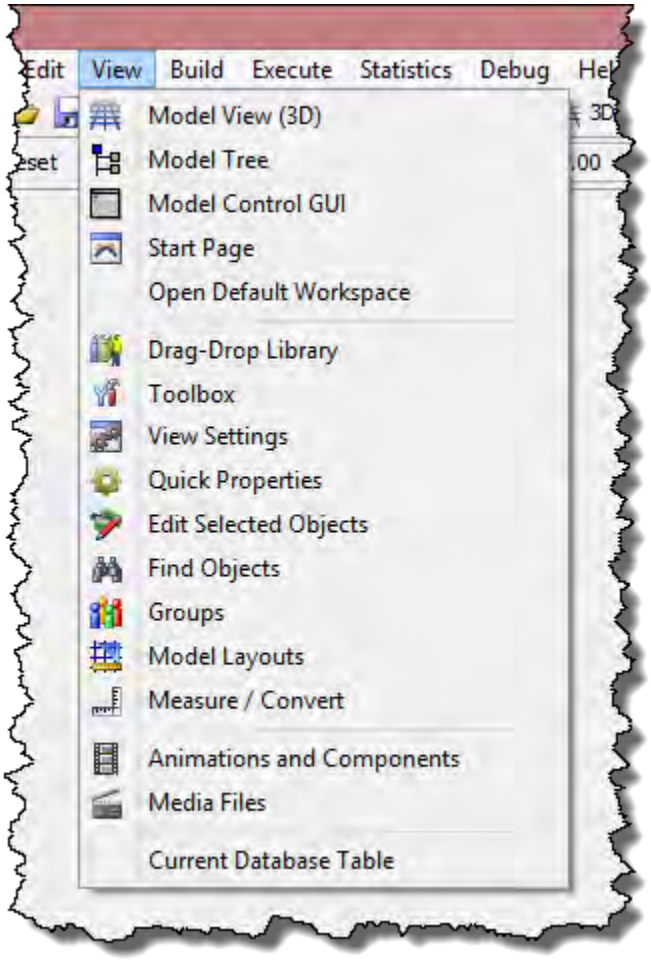
Edit Menu



Undo - Erases the last change made to the model, reverting it to the state immediately prior to making the change.

Redo - Reverses the Undo effect.

Resize and Rotate Objects - Clicking this option toggles whether or not object axes are shown in the Model View, which allows you to size and rotate model entities. By default, this option is enabled. Model Settings - Opens the Model Settings window, with settings unique to the current FlexSim model. For more information, see Model Settings.



Model View (3D) - opens a new perspective view of the model.

Model Tree - opens a tree view 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 Control GUI - opens the model's control GUI. This is a window that you can create yourself to do operations that are specific to your model. See Graphical User Interfaces for more information.

Start Page - opens FlexSim's start page, which is seen when you first open the FlexSim software. The start page provides options for creating a new model or opening an existing model, accessing the Getting Started guide and this users manual, adjusting global preferences, and licensing. It also shows the most recently edited FlexSim models, information on the currently installed license and FlexSim version, and some Internet-enabled active content.

Open Default Workspace - closes all of the active windows in FlexSim and then opens any windows saved as the default workspace from the Environment tab of the Global Preferences window.

Modeling Utilities

The following options will open various modeling utilities in one of FlexSim's panes. By default, the DragDrop Library and Quick Properties utilities are already open.

Drag-Drop Library - opens the Library utility. In most cases, this Library acts as an object library, with an icon grid of the available objects that can be dragged into the model.

Toolbox - opens the Toolbox utility.

View Settings - opens the View Settings utility.

Quick Properties - opens the Quick Properties utility. This utility provides quick access to object and flowitem properties that previously were only available in a properties window.

Find Objects - opens the Find Objects utility.

Groups - opens the Object Groups utility.

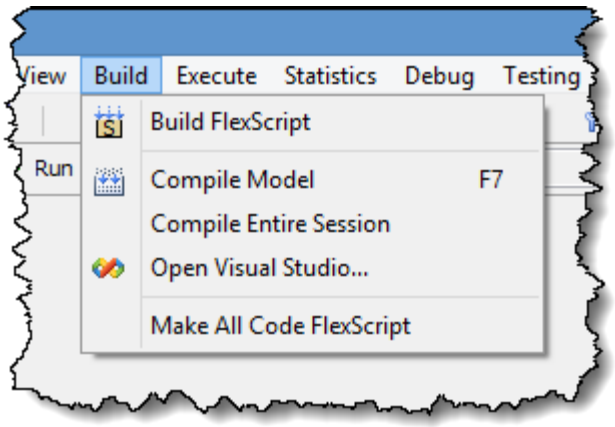
Model Layouts - opens the Model Layouts utility.

Measure/Convert - opens the Measure/Convert utility.

Animations and Components - opens the Animations and Components utility. This is the bottom window of the Animation Creator. Right click on an object and select Edit > Animations to open the Animation Creator.

Database

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.



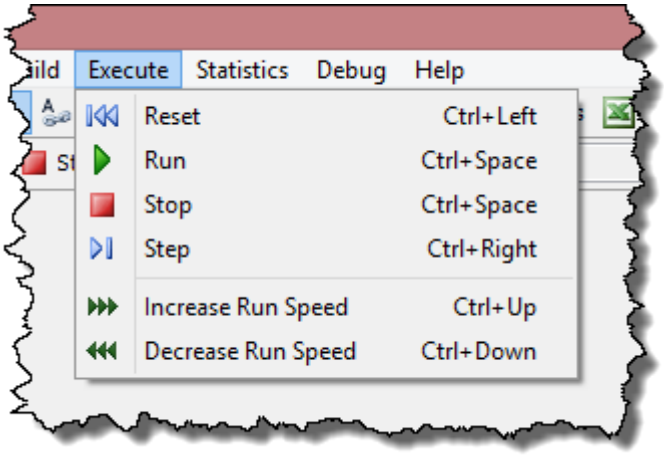
Build FlexScript - Builds all FlexScript code.

Compile Model - Compiles all C++ code in the model. This feature can also be accessed any time by pressing F7.

Compile Entire Session - Compiles all C++ code in the main tree. You usually 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 FlexScript - Toggles all C++ nodes in the model to be Flexscript.

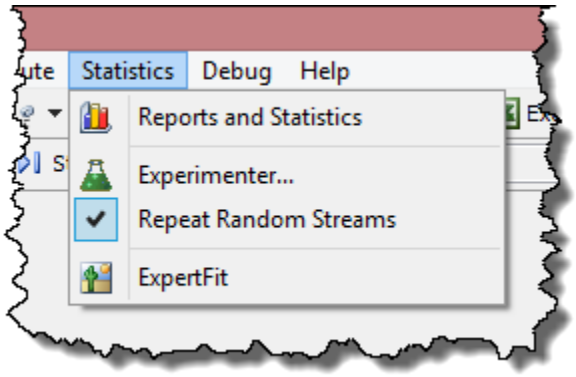


Reset - This option resets the current model file and prepares it to run. Same as selecting the Reset button on the simulation run panel. (Also accessible by holding down *Ctrl* and pressing the *left arrow*.) Run - This option runs the current model file. Same as selecting the Run button on the simulation run panel. (Also accessible by holding down *Ctrl* and pressing the *Spacebar* to toggle between run and pause.) Stop - This option stops the current model. Same as selecting the Stop button on the simulation run panel.

Step - This option pushes the model forward to the next event on the event list. Same as selecting the Step button on the simulation run panel. (Also available by holding down *Ctrl* and pressing the *right arrow*.)

Increase Run Speed - This option increases the run speed of the model by increasingly greater degrees. Same as moving the slider bar to the right on the simulation run panel. (Also available by holding down *Ctrl* and pressing the *up arrow*.)

Decrease Run Speed - This option decreases the run speed of the model by increasingly greater degrees. Same as moving the slider bar to the left on the simulation run panel. (Also available by holding down *Ctrl* and pressing the *down arrow*.)

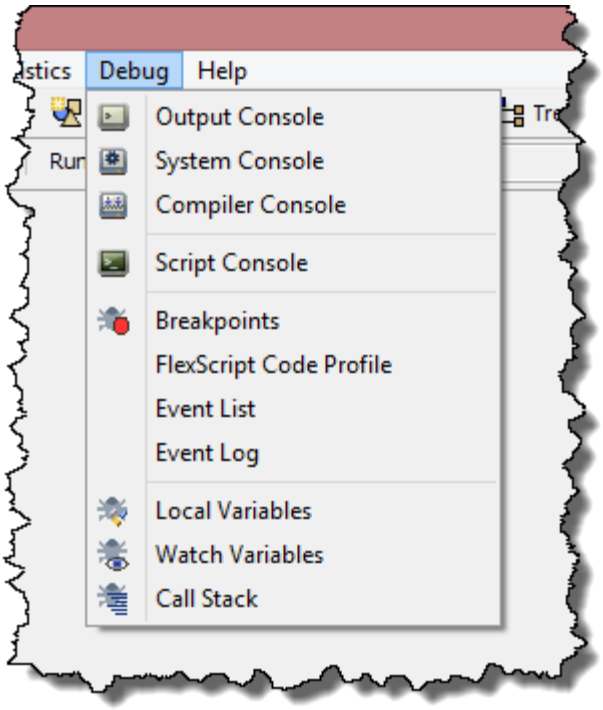


Reports and Statistics - This option opens the Reports and Statistics window. It allows the modeler to create various reports based on statistics gathered during a model run. These reports can include information about flowitem throughput, staytime, state history and other data that the modeler can select or customize.

Experimenter - This option brings up the Simulation Experiment Control input window. The Experimenter is used to perform multi-run multi-scenario analyses of your model. This is also how you access FlexSim's Optimizer, which 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 Optimizer, using OptTek's OptQuest engine, requires a special license sold as an add-on to the basic FlexSim software license.

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.

ExpertFit - This option opens the curve fitting software ExpertFit.



Output Console - This option opens a pane 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 pane where information about the status of the program engine is printed. Unhandled exceptions and other errors will be printed to this console.

Compiler Console - This option opens a pane where information is printed while the model is compiled. This shows the status of each step of the compilation process. This console also shows any FlexScript errors when you Build FlexScript.

Script Console - This option opens the scripting console. This pane is an interactive window to execute FlexScript commands manually. See Scripting Console for more information.

Breakpoints - This option opens the Breakpoints pane. See Breakpoints for more information.

FlexScript Code Profile - This option opens the FlexScript Code Profile pane, which lists all the FlexScript functions defined in the model, ordered by how much time the model spent executing them. This is an

excellent way to find out how the computer is spending its time and might lead to ideas on how to make the model faster. See Code Profiler for more information.

Event List - This option opens the Event List window, a sorted list of all pending events. See Event Lists for more information.

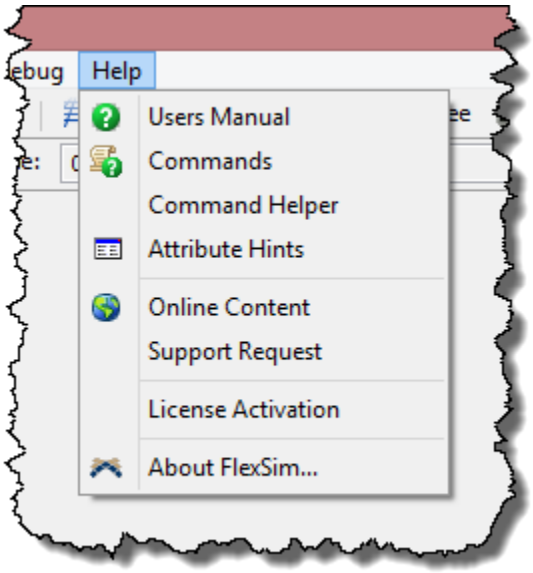
Event Log - This option opens the Event Log window, a sorted list of all events that have taken place. See Event Log for more information.

Local Variables - This option opens the Local Variables pane, which shows you the current values of any locally defined variables. See Local Variables for more information.

Watch Variables - This option opens the Watch Variables pane, which allows you to specify other variables or expressions that you want to see, such as global variables. See Watch Variables for more information.

Call Stack - This option opens the Call Stack pane, which shows the current call stack, a function call history. See Call Stack for more information.

Help Menu



Users Manual - opens this FlexSim user manual.

Commands - opens the User Manual to the Commands Reference.

Command Helper - opens the Command Helper pane that allows you to reference any FlexSim command. Provides details, parameters, and examples of each command.

Attribute Hints - opens the Attribute Hints window. This list shows all of the FlexSim attributes and their meanings.

Online Content - uses an internal browser to view and download online content through FlexSim including modules, 3D shapes, images, and sample models.

Support Request - opens the FlexSim Support Request page in a web browser, allowing you to submit a support ticket.

License Activation - opens the License Activation window.

About FlexSim... - opens a splash screen giving you information about FlexSim. It displays the license status, the version of FlexSim currently running, graphics card info, and copyright information.

FlexSim Toolbar

New: closes the current model and allows you to begin building a new one. A dialog box will appear asking if you'd like to save before creating a new model.

Open: allows you to open a previously saved model (.fsm file). A dialog box will appear asking if you'd like to save before creating a new model.

Save: allows you to save the current model. If the model has never been saved before, you will be asked to specify where the file will be saved.

Mode Toolbar

The mode toolbar lets you toggle between different edit modes in your 3D view. You enter a mode by pressing the appropriate mode button, and then exit the mode either by pressing the Esc key, right-clicking in the view, or by pressing a different mode button. Alternatively, you can hold down keys as defined in the keyboard interaction page.

Standard Mode - This mode is the standard mode for your view. Here you can move objects around, size them, etc. You can return to this mode at any time by pressing the Esc key.


Connect Objects - To connect two objects in this mode, click-and-hold on one object, drag the mouse to the other object, and release the mouse button on that object. You can also create multiple connections in series in this mode. To do this, click an object, then click another object, then click another object, and so on. This works the same way as holding down the A key.


Connect Center Ports - This mode works the same way as the Connect Objects mode, except that it connects center ports instead of input/output ports. This works the same way as holding down the S key.


Extended Connect - This mode works the same way as the Connect Objects mode, except that it does an extended connect that varies depending on the objects clicked. This works the same way as holding down the D key. For more information on extended connections (also known as "context sensitive" connect), see the keyboard interaction section.


Disconnect Objects - To disconnect two objects in this mode, click-and-hold on one object, drag the mouse to the other object, and release the mouse button on that object. You can also delete multiple connections in series in this mode. To do this, click an object, then click another object, then click another object, and so on. This works the same way as holding down the Q key.


Disconnect Center Ports - This mode works the same way as the Disconnect Objects mode, except that it disconnects center ports instead of input/output ports. This works the same way as holding down the W key.


Extended Disconnect - This mode works the same way as the Disconnect Objects mode, except that it does an extended disconnect that varies depending on the objects clicked. This works the same way as holding down the E key. For more information on extended disconnections (also known as "context sensitive" disconnect), see the keyboard interaction section.


New Selection - You can create selection sets to have operations apply to a whole set of objects. To select objects in this mode, drag a box around the objects that you want selected. To select none, click the background of the view in this mode. This works the same way as holding down the Shift key.


Toggle Selection - You can create selection sets to have operations apply to a whole set of objects. To select objects in this mode, drag a box around the objects that you want selected. Any

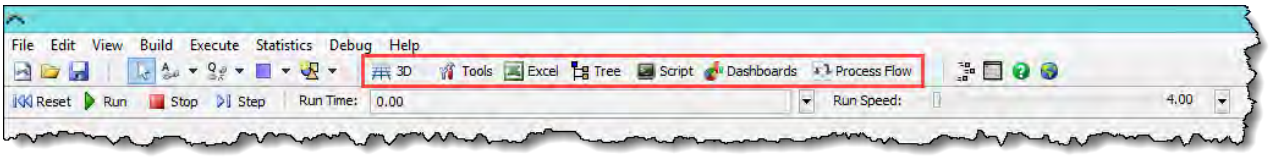
selected objects in the box will become unselected. Any unselected objects in the box will become selected. This works the same way as holding down the Ctrl key.

 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. This works the same way as holding down 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'). This works the same way as holding down the R key.

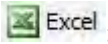
Modeling Tools/Views Toolbar

A set of the most common modeling views and tools are available through this toolbar.

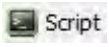


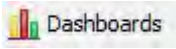
 : opens a new perspective view of the model.

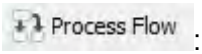
 : opens the Toolbox window.

 : opens the Excel interface dialog box.

 :
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 Model View. Objects' attributes can also be edited here, however, it is recommended that object properties be changed through the object's Properties dialog window or Quick Properties, and not through the tree.

 : opens a script console. Here you can execute FlexScript commands to change or get information from your model.

 : here you can create and open dashboards. Here you can gather statistics from your model.

 : here you can create and open process flows.

Custom Toolbar

This section of the toolbar can be customized with any options from the main menu. To customize this section, go to File > Global Preferences and select the Customize Toolbar tab.

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 Simulation Run panel is found at the top of the main window in FlexSim, and we describe it here in three sections:

Buttons

Reset: resets the model. The OnReset function is called for each object in the model. This should always be selected before running a model.

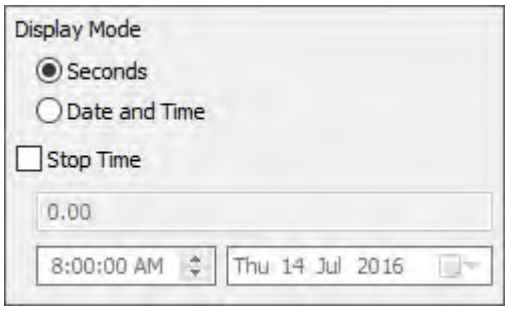
Run: starts the model running. The model clock will continuously advance until the model is stopped or the event list is empty.

Stop: 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.

Step: sets the model clock to the time of the next event that needs to occur. That event then happens. This enables users to step through the execution of a model one event at a time. 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.

Run Time: Displays the model's current run time.

Time/Stop Time



Display Mode

- Seconds (or current model time unit) - The Run Time field will display in the model's time units.
- Date and Time - The Run Time field will display the model's current date and time. The model start date and time is defined in your Model Settings. This display mode is nice for Time Tables that are using the Graphical Time Table.

Stop Time

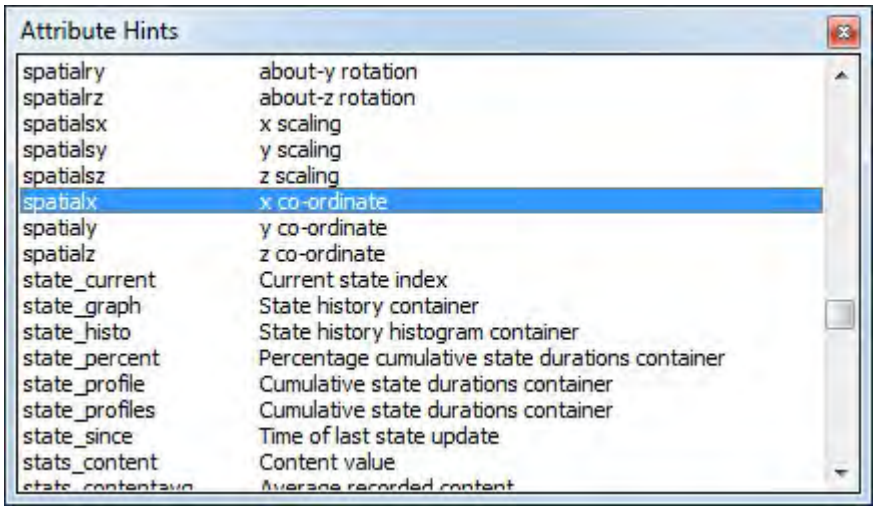
If not selected, the model will run indefinitely. The model can still be stopped if the "Stop" button is clicked or the event list is empty.

If selected, the model will run the specified number of time units (or to the specified date and time) before stopping. The model may also be stopped at a given time using a picklist option on any relevant object in the model. In the "OnEntry" or "OnExit" trigger, the "Stop the Model Run" option can be used to dynamically stop the model run.

Speed

Speed slider: 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. This value can also be changed by clicking the black arrow to the right and entering a number manually.

Attribute Hints

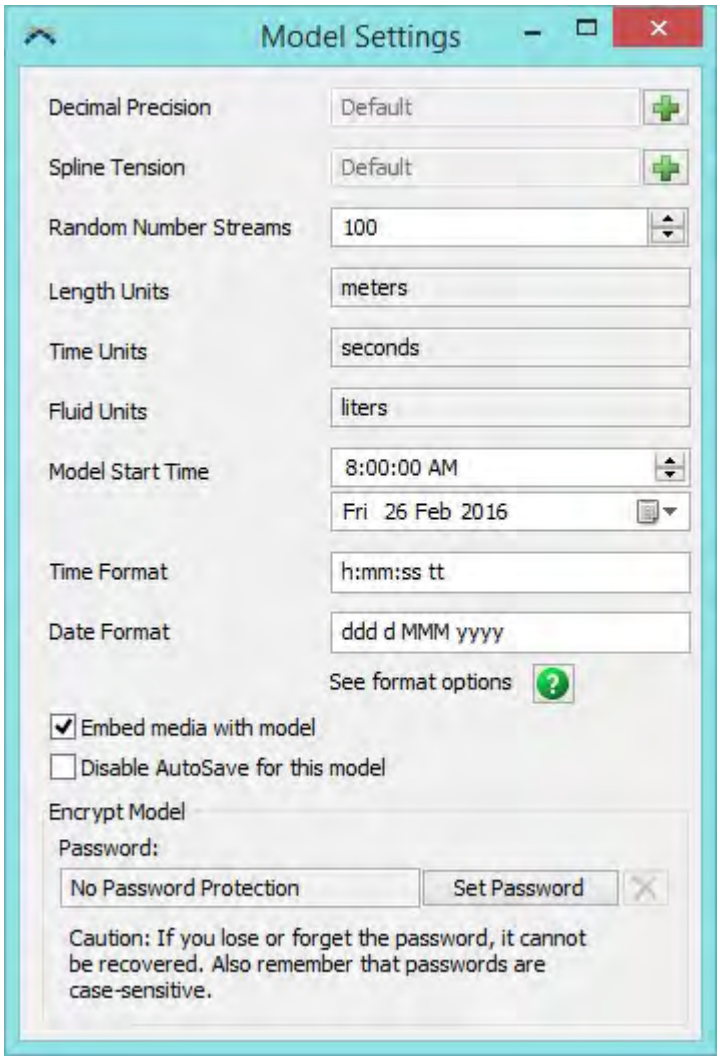


This table view is accessed from the Help menu. The attribute hints window shows the list of special FlexSim attributes and their meaning. Click inside the window and start typing the name of an attribute you would like to know about to have the window will automatically scroll to that attribute.

For more information about attributes, see the View Attributes Reference and GUI Reference pages.

Model Settings

This table view is accessed from the Edit menu.



Decimal Precision- This option controls the precision of editable values seen in the interface.

Spline Tension- This option controls how straight the lines in a curved network node path are. A value close to 0 will produce straight lines. A value close to 1 will produce curved lines.

Random Number Streams- This option controls the range (between 0 and the specified value) of random number streams that will be initialized by Flexsim when the model is reset.

Length Units- This displays the length units of the model.

Time Units- This displays the time units of the model. Fluid

Units- This displays the fluid units of the model.

Model Start Time- This displays the global model start time and date that is used in various objects like the TimeTable.

Time Format- This specifies what format the Model Start Time should be displayed in. See format options.
Date Format- This specifies what format the Model Start Date should be displayed in. See format options.
Embed media with model- This option allows you to embed all of the 3D and image files associated with your model into the model file.

Disable AutoSave for this model- This option allows you to disable the AutoSave feature as defined in Global Preferences.

Encrypt Model- This area allows you to set a password for your model, so that only those with the password can open it. There is no way to recover lost passwords. To set a password, click the Set Password button. Then enter the desired password and click the Enter Password button. Enter the password again and click the Confirm Password button. To remove a password, click the  button.

Date/Time Format Strings- The following is a list of characters or elements that may be used in a date/time format.

Body text can be added to the format string by enclosing the text in single quotes. Note:

Time formats and date formats are exclusive and may not be mixed. Time Format

Element	Description
h	The one- or two-digit hour in 12-hour format.
hh	The two-digit hour in 12-hour format. Single-digit values are preceded by a zero.
H	The one- or two-digit hour in 24-hour format.
HH	The two-digit hour in 24-hour format. Single-digit values are preceded by a zero.
m	The one- or two-digit minute.
mm	

	The two-digit minute. Single-digit values are preceded by a zero.
s	The one- or two-digit second.
ss	The two-digit second. Single-digit values are preceded by a zero.
t	The one-letter AM/PM abbreviation (that is, AM is displayed as "A").
tt	The two-letter AM/PM abbreviation (that is, AM is displayed as "AM").

Date Format

Element	Description
d	The one- or two-digit day.
dd	The two-digit day. Single-digit day values are preceded by a zero.
ddd	The three-character weekday abbreviation.
dddd	

	The full weekday name.
M	The one- or two-digit month number
MM	The two-digit month number. Single-digit values are preceded by a zero.
MMM	The three-character month abbreviation.
MMMM	The full month name.
yy	The last two digits of the year (that is, 1996 would be displayed as "96").
yyyy	The full year (that is, 1996 would be displayed as "1996").

Global Preferences Window

This table view is accessed from the File menu. The global preferences dialog window specifies default user preferences for FlexSim. These preferences are saved across several models and remembered when FlexSim is closed and reopened. The dialog window is split into seven tabs.

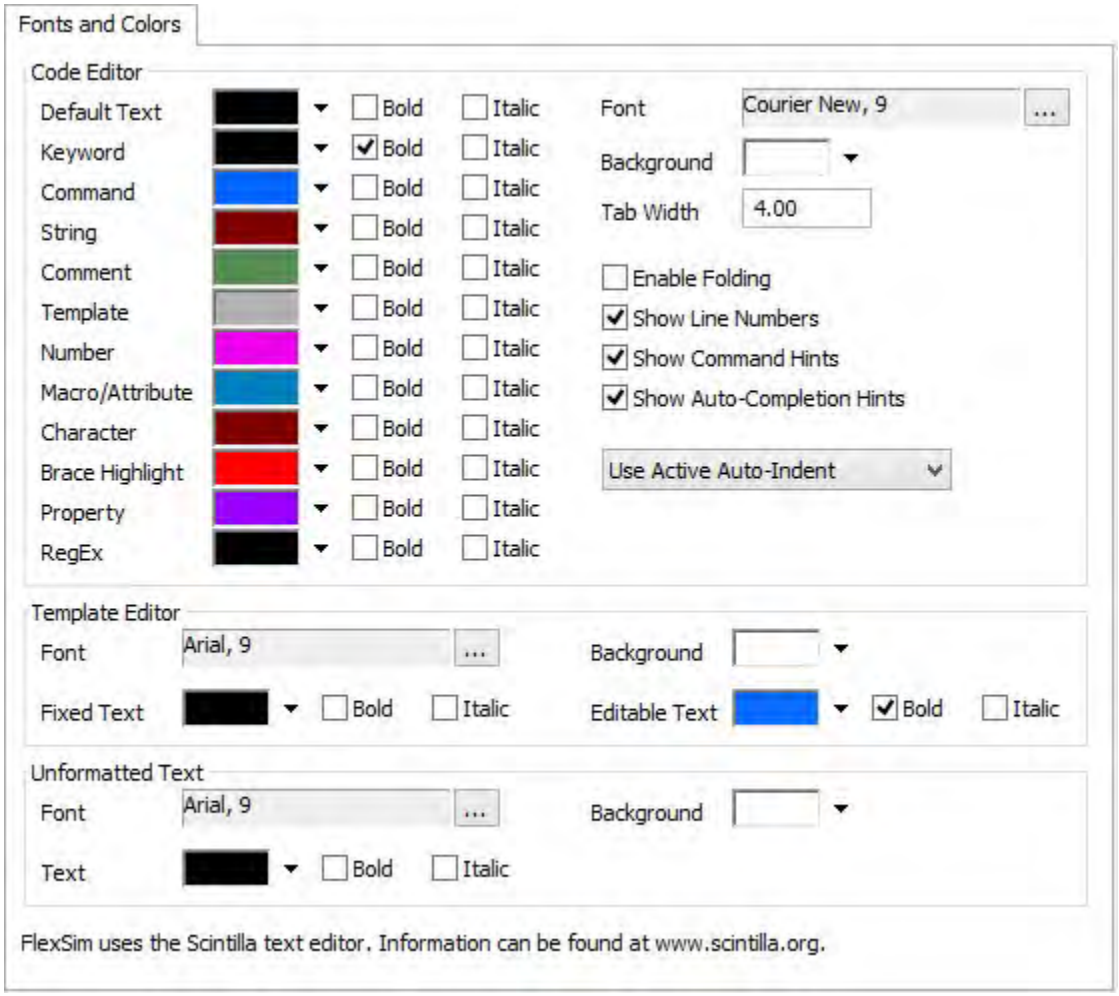
Tabs

- Fonts and Colors
- Environment
- Libraries
- Dynamic Content
- Customize Toolbar

- Graphics
- Compiler

Fonts and Colors

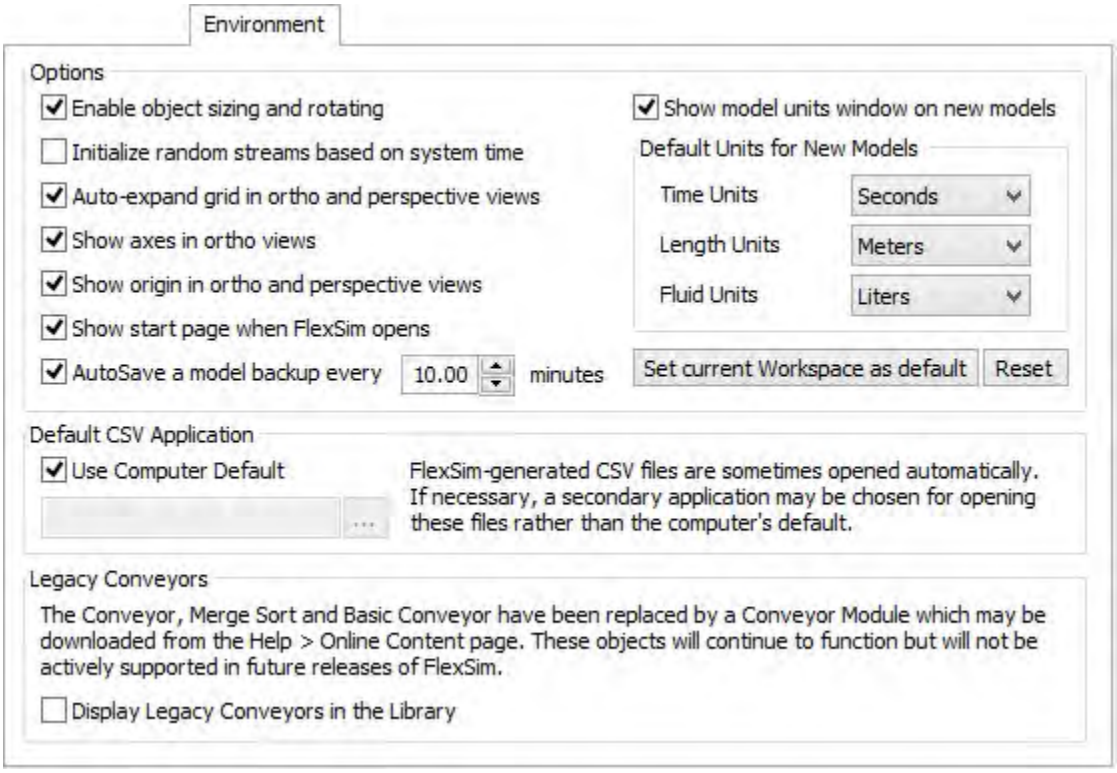
The Fonts and Colors tab specifies syntax highlighting and other settings used in FlexSim's implementation of the Scintilla code editor. You can also specify settings and colors for the template editor, as well as for unformatted text (which is used in multi-line unformatted text controls like the user description portion of the Properties window).



For more information on the Scintilla code editor, go to www.scintilla.org.

The Scintilla text editor is under copyright 1998-2003 by Neil Hodgson <neilh@scintilla.org>. All Rights Reserved.

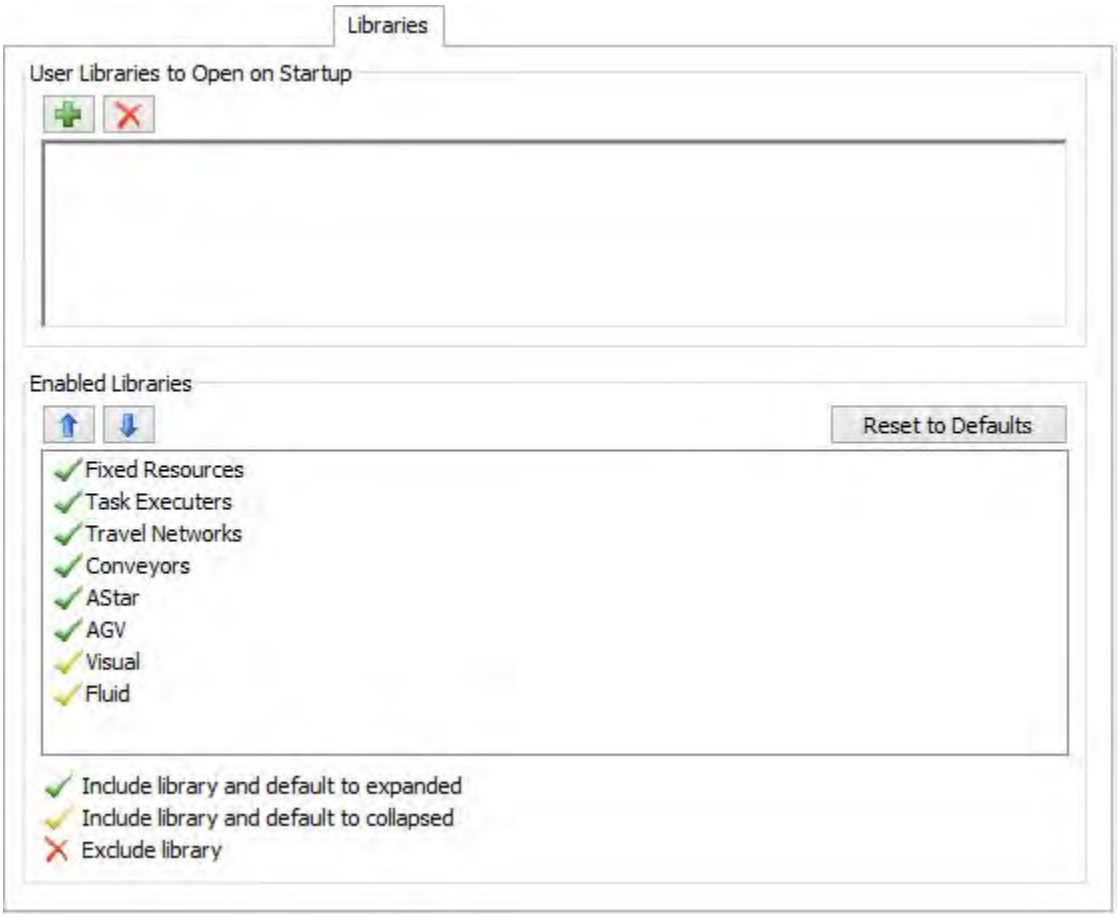
Environment



In this tab page you can specify various settings such as whether you want code to be C++ or FlexScript by default, various grid settings in the ortho and perspective view, excel DDE settings, etc.

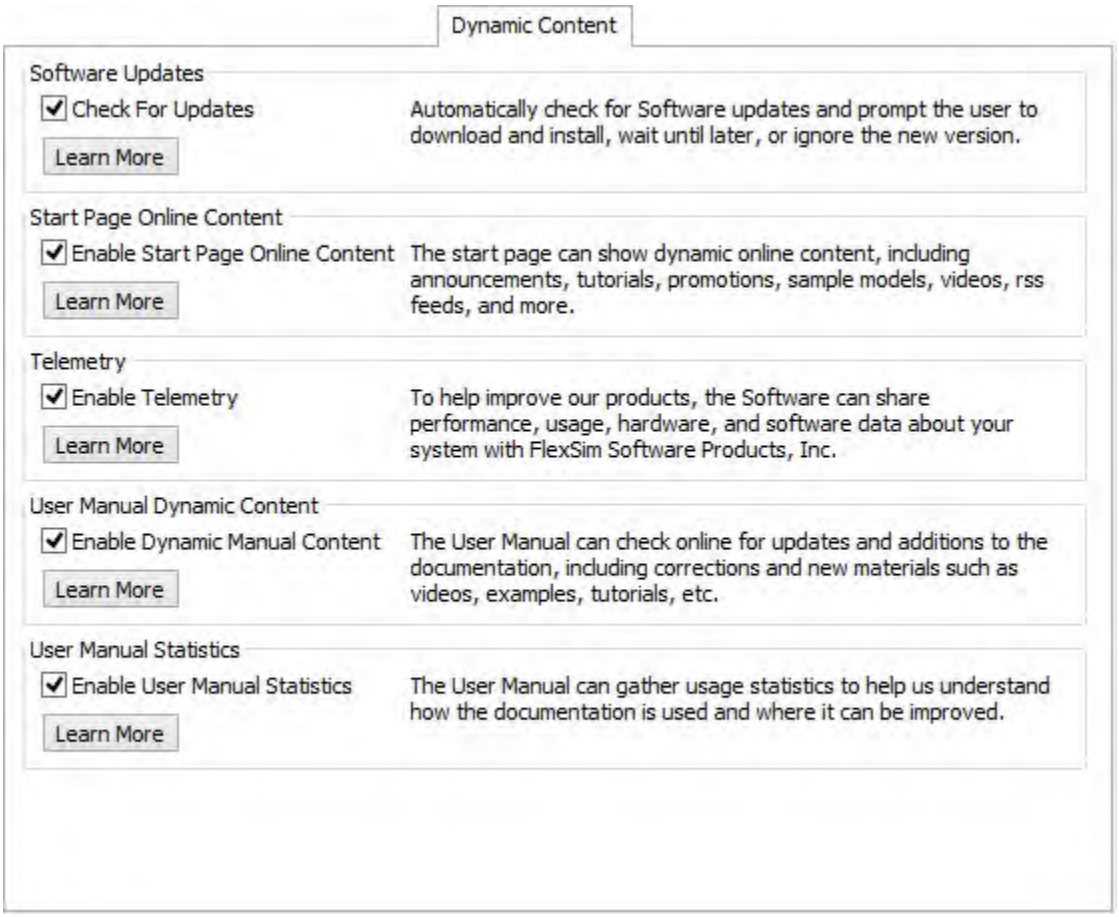
AutoSave will automatically save a backup model of your currently open model. This backup model will be named [modelname]_autosave.fsm and will be saved in the same directory as your model. AutoSave will only save your model if it is reset and not running. It will not save your model if the model is running. You can disable AutoSave for a specific model in the Model Settings window.

Libraries



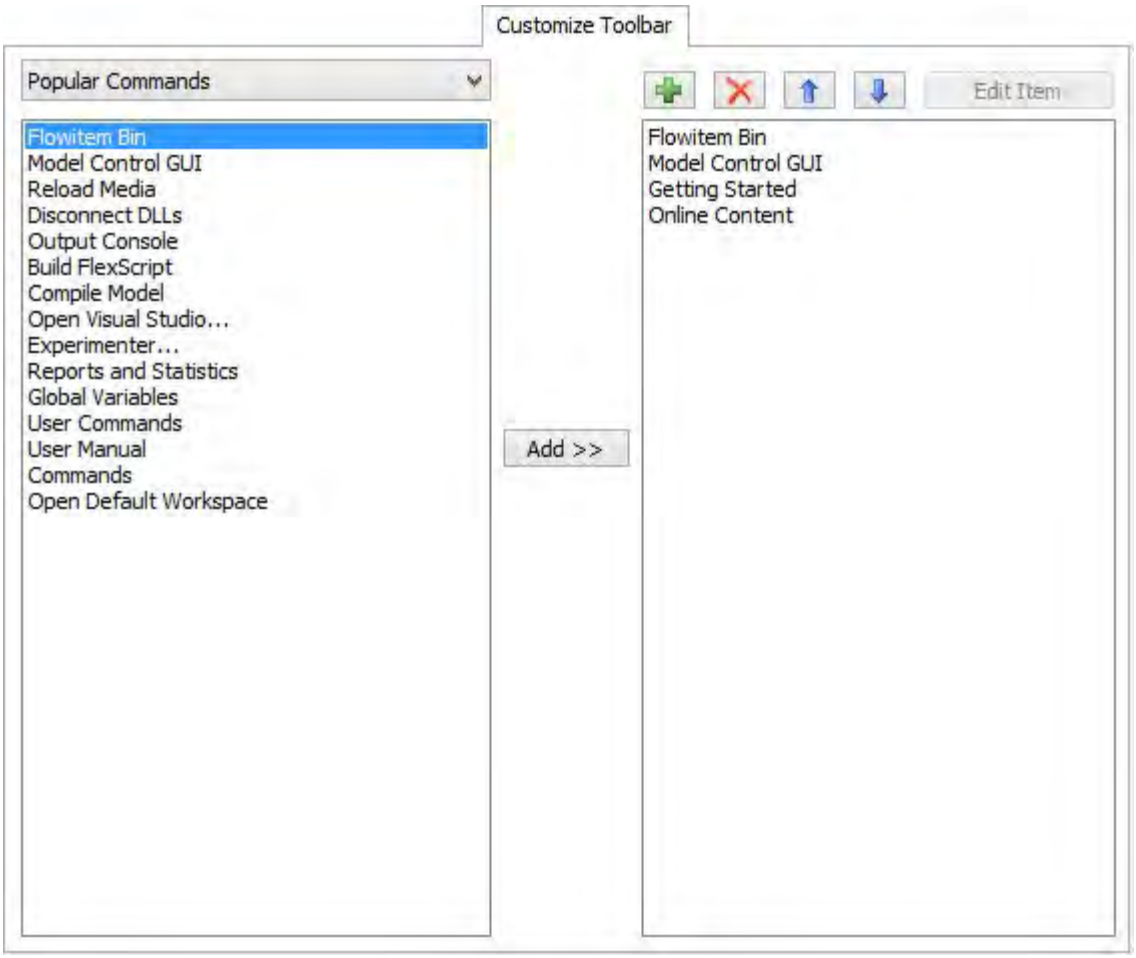
This tab page lets you control libraries that are loaded and displayed in the Drag-Drop Library. The paths specified in the User Libraries to Open on Startup are relative to your libraries folder of the install directory (ie C:/Program Files/FlexSim/libraries). Enabled Libraries allow you to specify which libraries are visible and which order they appear in. Once a User Library is loaded, it will be displayed in this list.

Dynamic Content



In this tab page you can specify various settings that send or pull content from the FlexSim server in order to give you a more dynamic experience or to help FlexSim better understand how to improve the software.

Customize Toolbar



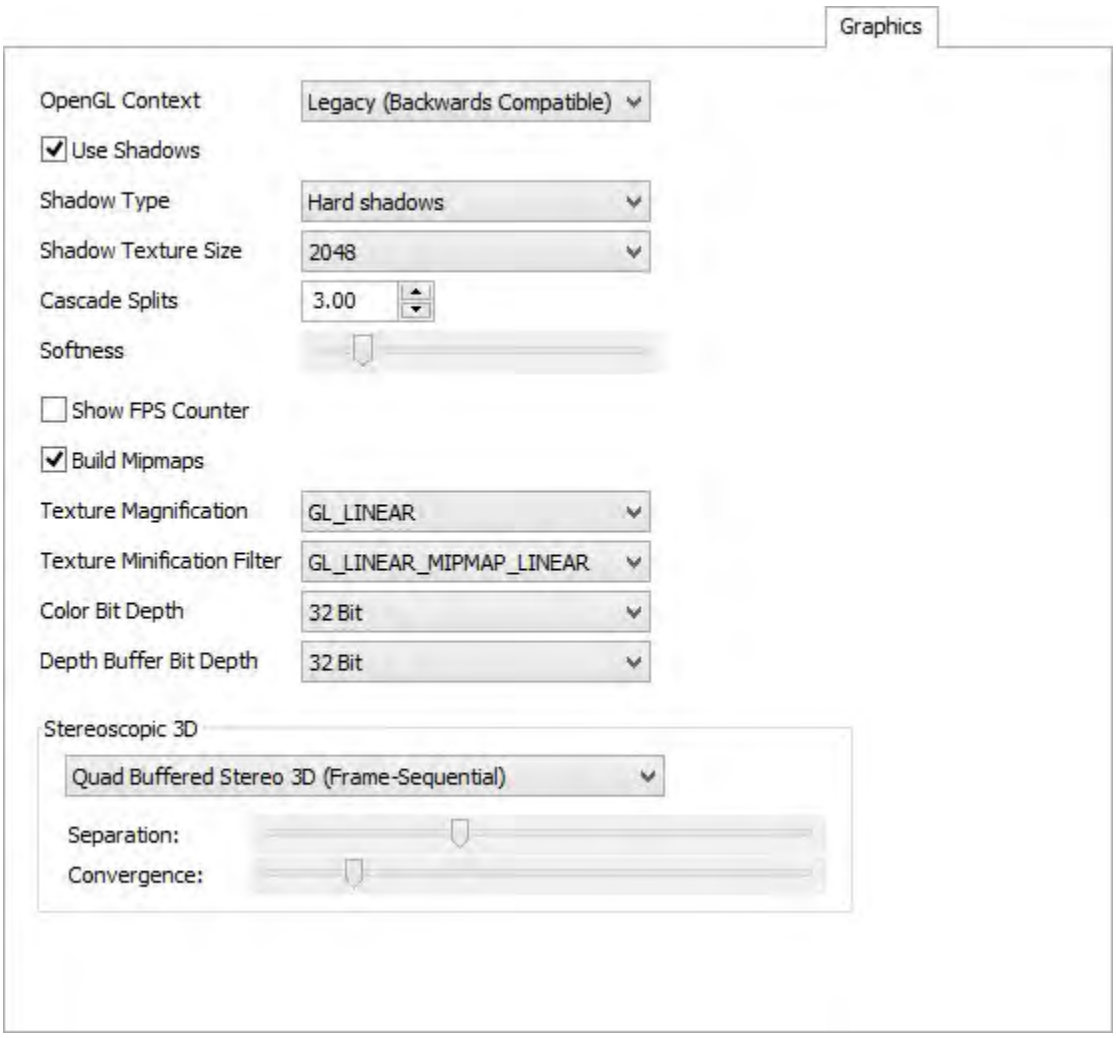
In this tab page you can customize which menu commands are accessible easily through the customizable section of the top toolbar.

Graphics

This tab is used for customizing 3D graphics settings so that FlexSim will run the best on your hardware.

Several of these options provide suggestions or hints to the graphics card as to the quality you would like. If the graphics card does not support the feature requested, such as Quad Buffered Stereo 3D or a 32 Bit Depth Buffer, it will fall back to using the next best option that it supports.

If no graphics acceleration is available with your system configuration, FlexSim will automatically fall back to using a generic software OpenGL implementation and ignore the settings defined here.



OpenGL Context- This option controls which type of OpenGL rendering context FlexSim will request from the graphics driver on your computer.

- Generic (No GPU Acceleration) - A generic software renderer by Microsoft that supports OpenGL 1.1 on any hardware. Graphics processing will be done on the CPU and not the graphics card (GPU). This type of rendering context is slow, and certain graphics features, such as bone animations and shadows, will not work with this context. This option was previously known as Compatibility Mode.
- Legacy (Backwards Compatible) - A hardware-accelerated OpenGL rendering context that supports almost all features of OpenGL, including deprecated features. This context is backward compatible with legacy OpenGL functions and is also known as the OpenGL Compatibility Profile.
- Modern (Core Profile) - A hardware-accelerated OpenGL rendering context that supports only modern OpenGL features. Deprecated features have been removed. Some graphics driver implementations and system configurations (such as VMware virtual machines) only perform hardware graphics acceleration with a modern OpenGL Core Profile.

Note: If drag-dropping objects, making connections, or navigating the 3D view is not working properly, your graphics card may not be compatible with FlexSim. You should first try using the Modern (Core Profile) OpenGL Context. If that doesn't work, try updating your graphics card driver from your computer's manufacturer. If that does not help, try updating the driver from the graphics card vendor (typically Nvidia, AMD, or Intel). As a last resort if updated drivers do not resolve your issues, select the Generic OpenGL Context to use your CPU for graphics processing.

Use Shadows- This option controls whether shadows will be rendered in the 3D environment.

Shadow Type- This specifies whether shadows should be rendered with hard or soft edges.

Shadow Texture Size- This specifies the resolution of the shadow map. A higher resolution will improve shadow quality, but will require more graphics processing.

Cascade Splits- This specifies the number of cascaded shadow maps to use. More cascade splits will improve shadow quality, but will require more graphics processing.

Softness- This affects how much shadows are blurred around the edges when using soft shadows.

Show FPS Counter- If checked, FlexSim will display a counter showing how quickly the 3D view is rendering in frames per second.

Build Mipmaps- If checked, FlexSim will generate multiple resized "thumbnails" of each texture it loads. This allows for faster and better texture rendering when zoomed out, but requires approximately double the graphics card memory to store the texture.

Texture Magnification- Select how the graphics card should resolve the color of each pixel when you are zoomed in close to a texture.

- GL_LINEAR - This will do a linear interpolation of the color values of multiple texture elements surrounding the pixel. While this looks much better when zoomed in, the linear interpolation calculation can be slower than the GL_NEAREST option.
- GL_NEAREST - This will just take the color value of the texture element closest to the pixel. While faster, this doesn't look as good when zoomed in.

Texture Minification- Select how the graphics card should resolve the color of each pixel when you are zoomed out from a texture.

- GL_LINEAR - This will do a linear interpolation of the color values of multiple texture elements near the pixel. This allows for a more smooth look when zooming out, but the linear interpolation calculation can be slower than the GL_NEAREST option.
- GL_NEAREST - This will just take the color value of the texture element closest to the pixel. While faster, can cause weird "flicker" artifacts when zooming out.
- The other options are more or less similar to the above options, but allow more flexibility in defining which mipmap "thumbnail" to use as well.

Color Bit Depth- Choose the format in which you want the graphics to store color data.

Depth Buffer Bit Depth- Choose the format in which you want the graphics to store depth buffer data. Depth buffer values span from a 3D view's near plane to its far plane, so the depth buffer bit depth defines the granularity by which the graphics card can distinguish which objects are in front of or behind other objects.

Stereoscopic 3D- This option controls how stereoscopic 3D should be rendered.

- No Stereoscopic 3D - Do not attempt to do any stereoscopic 3D rendering.
- Quad Buffered Stereo 3D (Frame-Sequential) - Certain system configurations can use this mode to sync the 3D display with active-shutter glasses using four OpenGL render buffers to display every other frame to each eye. This mode requires specialized hardware and software designed for 3D, such as Nvidia 3D Vision.

  Once your computer is properly configured with a 3D display, this mode will automatically render 3D views in stereoscopic 3D. If you are using an Nvidia Geforce graphics card, the view must be fullscreen (F11) to trigger the effect. On Nvidia Quadro or AMD FirePro graphics cards, the effect will work with all 3D views.

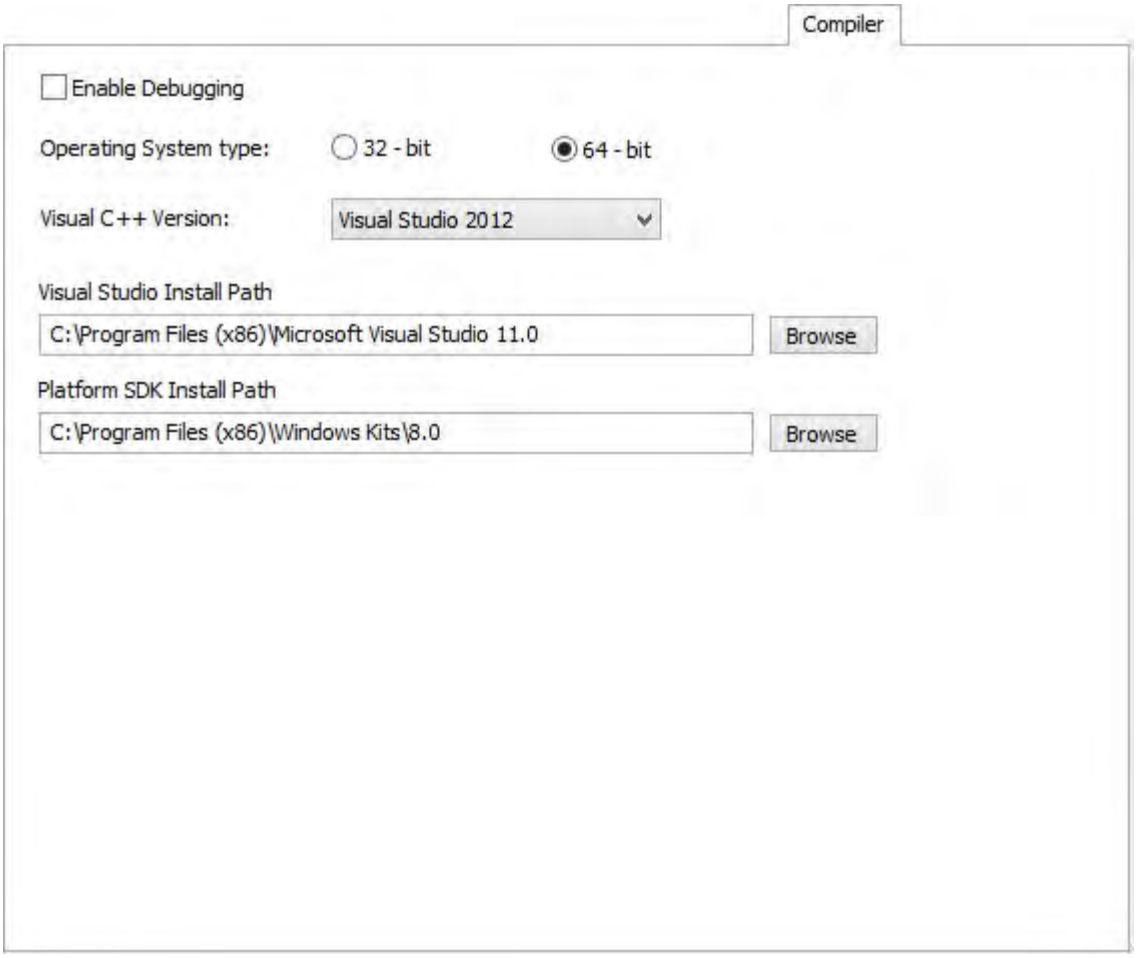
This mode only works with certain configurations of hardware and software. The graphics card, 3D display monitor, cable/port (DVI-D, DisplayPort, HDMI 1.4a), graphics driver, and operating system must all be compatible for this mode to work correctly. If any of those pieces are not compatible, then FlexSim will automatically fall back to rendering without stereoscopic 3D.

- Side-by-Side - This option will render each eye to one side of the screen. This can be used to record 3D screenshots or video to be later played with a stereoscopic 3D picture viewer or video player. It can also be used in full-screen mode with certain 3D TVs or monitors that support side-by-side stereo 3D input.
- Top-Bottom - This option works similarly to side-by-side, but one eye will be rendered at the top of the screen and the other will be rendered at the bottom.

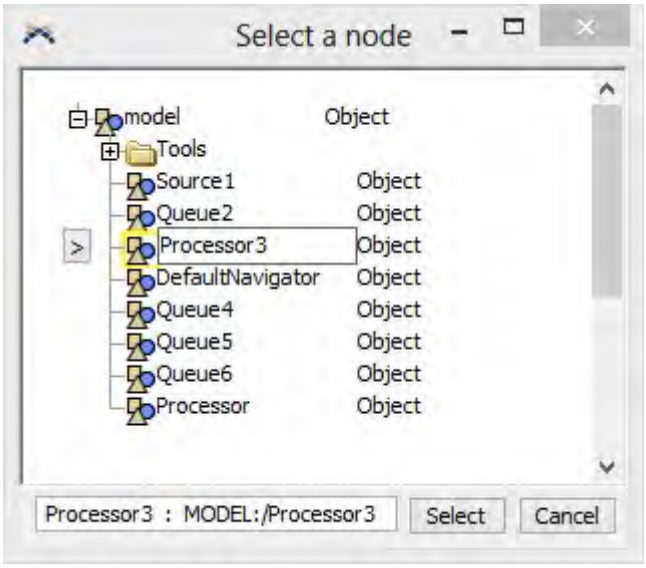
Separation- This controls the difference in horizontal position between the right and left images, changing the depth of the stereo effect.

Convergence- This controls the focal point of the stereo effect. Increasing the convergence makes near objects appear to pop out of the screen.

Compiler

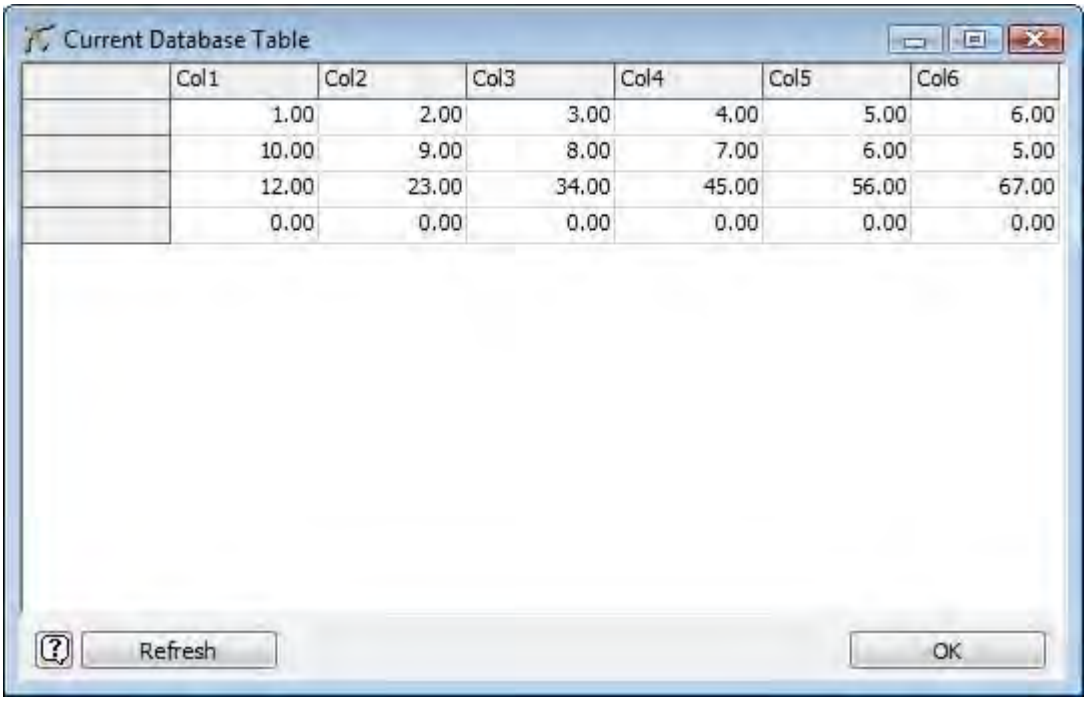


This tab is used for configuring Visual Studio to compile or debug C++ code.



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. You may also select objects or nodes in the Global Variables window.

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.



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 table editor is used to edit a simple table in FlexSim, and can be opened by right-clicking on a node and choosing Explore > As Table.

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.

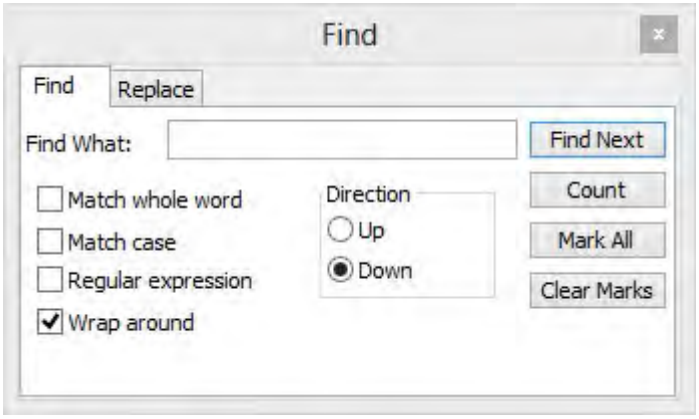
- Pins the entire table to a Dashboard as either a table of values, bar chart or line graph.

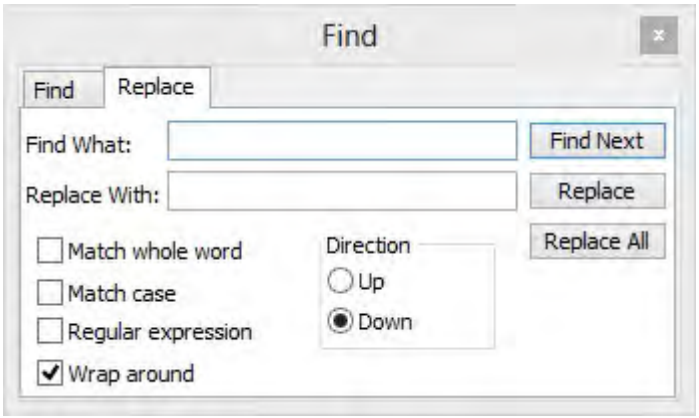
Rows - This is the number of rows in the table. After changing it, press Apply to update the table onscreen. Any new rows that were created can now be edited.

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.

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 Assign String Data.

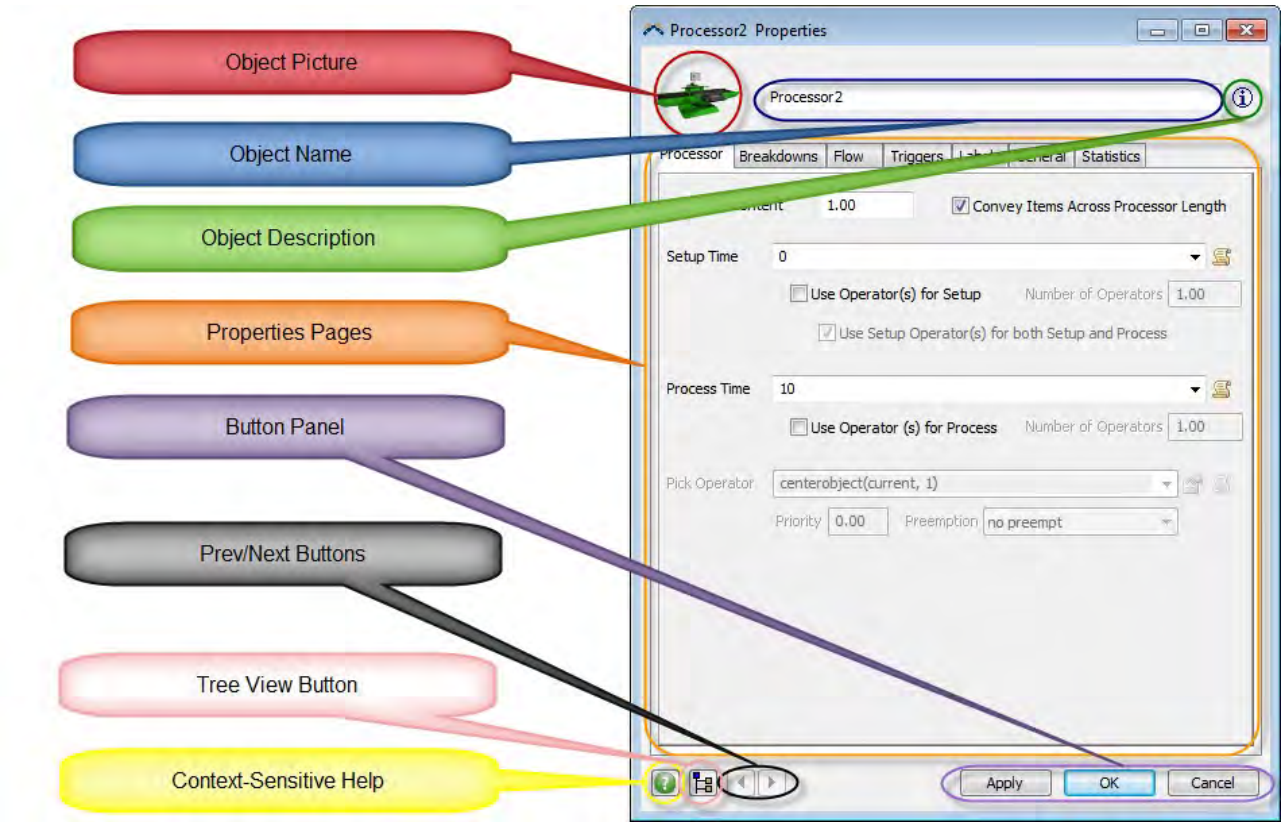




The Find/Replace window is available by holding the Ctrl key and pressing F. This window allows you to search the currently active window or view for the specified text and replace that text if desired.

The Properties 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 Properties window of the Queue is therefore different from the Properties window of the Combiner. However, there are often commonalities between Properties 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 Properties 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 "Properties" option from the popup menu.



Object Picture

This picture shows you what type of object you are editing.

Object Name

Here you can give the object a different name. In specifying the name of the object, do not use any special characters like >,<,*,-, (,), etc. This may cause Flexsim to not behave correctly. Spaces and underscores should be the only non-alpha-numeric characters used. Also, do not begin the name with a number. You must press Apply or OK for this change to be applied. It is advisable to give each object in your model a unique name for reporting and other purposes. An error message will appear when you press Apply if another object has the same name as name you entered for this object.

User Description

This button opens an edit field where you can type a description or notes about the object. This description is saved with the object and can be viewed at a later time the same way it is edited: by pressing this button. Click anywhere in the window for the User Description edit field to apply and disappear.

Properties Pages

Properties windows are made up of a set of tab pages, depending on the object. Many pages are used by several objects. This reference provides documentation for each properties page. The properties pages are listed as follows. Within each help page are links to each object that uses that page. Fixed Resource Pages - These pages are used by Fixed Resource objects.

- BasicFR Advanced
- Breakdowns
- Combiner
- Flow
- MultiProcessor
- Processor
- ProcessTimes
- Queue
- Rack
- Separator
- Sink
- SizeTable
- Source

Task Executer Pages - These pages are used by Task Executer objects.

- ASRSvehicle
- BasicTE
- Breaks
- Collision
- Crane
- Dispatcher
- Geometry
- Robot
- TaskExecuter
- Transporter

Fluid Pages - These pages are used by fluid objects.

- Blender
- FluidConveyor
- FluidLevelDisplay

- FluidProcessor
- FluidToItem
- Generator
- Initial Product
- Inputs / Outputs
- ItemToFluid
- Layout
- Marks
- Mixer
- Percents
- Pipe
- Recipe
- Sensors
- Splitter
- Steps
- Tank
- Terminator
- Ticker

Shared Pages - These pages are shared by multiple objects.

- General
- Labels
- Triggers

Other Pages

- Container Functionality
- Display
- NetworkNode
- NetworkNodes
- Speeds
- Traffic Control

Button Panel

Apply - Applies any changes made in the window to the object.

OK - Applies any changes made in the window to the object and closes the window.

Cancel - Closes the window without applying any changes made in the window. Some properties of the object are updated immediately when changing them in the window. These changes will still be applied and will not be undone when pressing Cancel. Examples of these immediately changing properties are Breakdowns, Labels, Appearance, Position, Rotation, Size, and Ports.

Prev/Next Buttons

These buttons will apply any changes made in the window to the object and change the focus of the window to the previous or next object of the same class type in the tree. This is helpful for quickly modifying properties of multiple objects with the same class type.

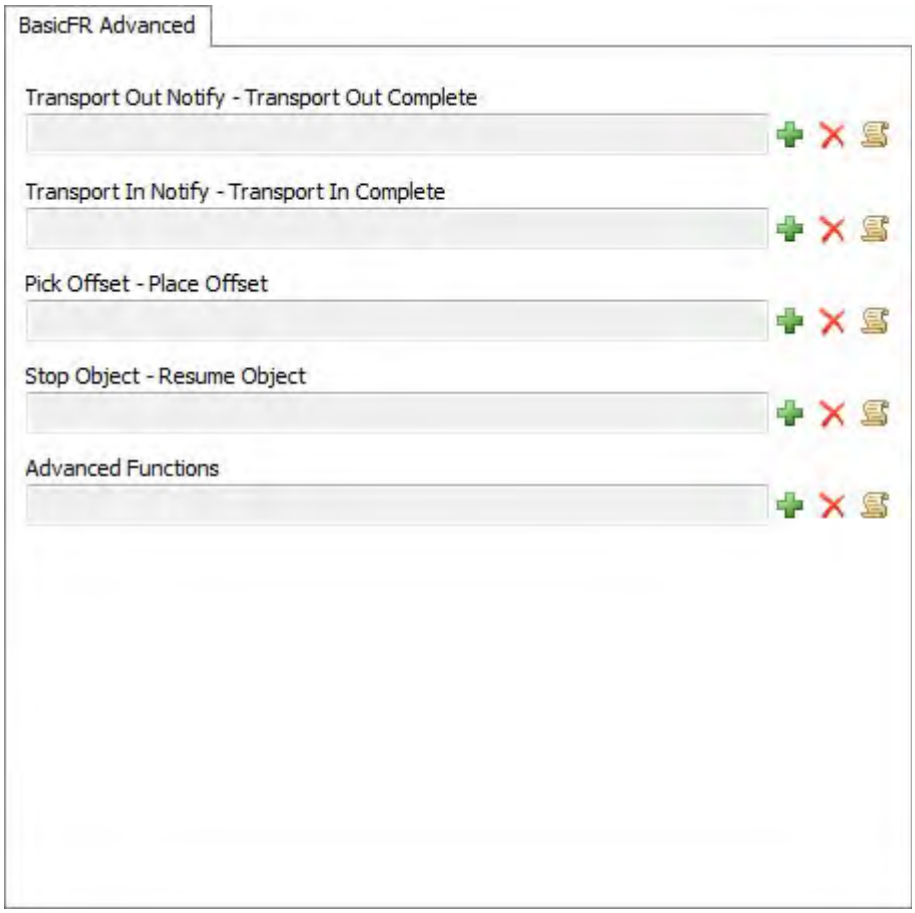
Tree View Button

This button opens a tree edit window showing just this object.

Context-Sensitive Help

This button will open the User's Manual to the page describing the currently selected tab.

BasicFR Advanced Page



Transport Out Notify - Transport Out Complete - This trigger 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 nroftransportsout 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.

nroftransportsoutnode: This is a reference to the object's nroftransportsout 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 trigger 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 nroftransportsin 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.

nroftransportsinnode: This is a reference to the object's nroftransportsin 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 resume object 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 trigger 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 trigger 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.

nrofstopsnode: 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 trigger 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.

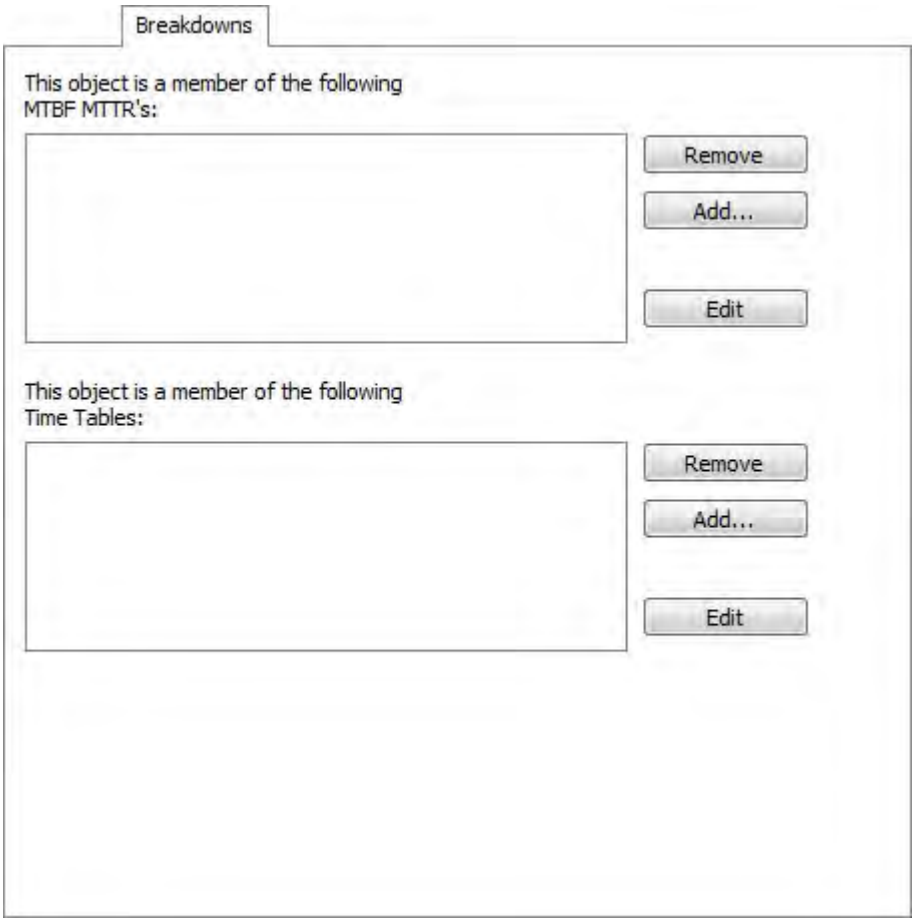
ADV_FUNC_KEYEDCLICK: This function is called when a key on the keyboard is held down and the object is clicked on. Here the view is passing as parnode(2), the click code is passed in as parval(3), and

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

Breakdowns Page



MTBF MTTR Member List - This is a list of all the MTBF MTTR objects that have this object as one of its members. 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. For more information about MTBF MTTR objects, refer to the Modeling Tools section about MTBF/MTTR objects.

Remove - This button removes the object from the selected MTBF MTTR object's member list.

Add.. - This button opens a listbox of all the MTBF MTTR objects in the model. You can select an MTBF MTTR object from the list to add this object to that MTBF MTTR object. You can also select "Add New MTBF MTTR" to create a new MTBF MTTR object in your model and add this object to that MTBF MTTR object.

Edit - This button allows you to edit the MTBF MTTR object's properties, including the timing of the breakdowns and repairs. For more information about MTBF MTTR objects, refer to the Modeling Tools section about MTBF/MTTR objects.

Time Tables Member List - This is a list of all the Time Table objects that have this object as one of its members. Each Time Table object can be connected to more than one object in the model. And each object can be controlled by more than one Time Table object. For more information about Time Table objects, refer to the Modeling Tools section about Time Tables.

Remove - This button removes the object from the selected Time Table object's member list.

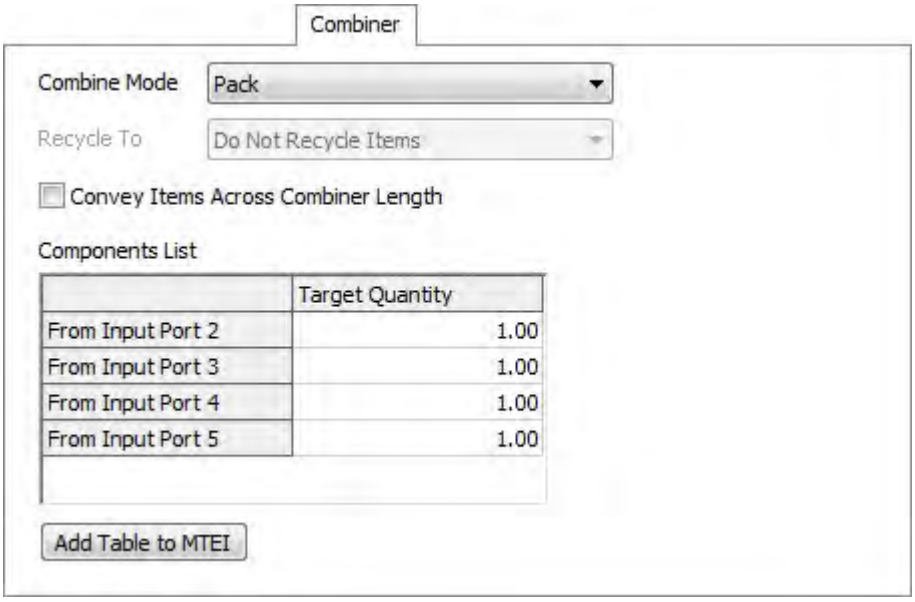
Add.. - This button opens a listbox of all the Time Table objects in the model. You can select an Time Table object from the list to add this object to that Time Table object. You can also select "Add New Time Table" to create a new Time Table object in your model and add this object to that Time Table object.

Edit - This button allows you to edit the Time Table object's properties, including the timing and duration of scheduled breakdowns. For more information about Time Table objects, refer to the Modeling Tools section about Time Tables.

This Page is Used By

Processor
Combiner
Separator

Combiner Page



Pack/Join/Batch - Selects the mode that the Combiner is operating in.

Recycle To (Join mode only) - In join mode, the objects that come in through ports greater than 1 are destroyed after the combiner is finished processing. Rather than destroying the extra flowitems, you can use this option to recycle them to a specific flowitem recycling bin. To learn more about recycling, refer to the Sink Page.

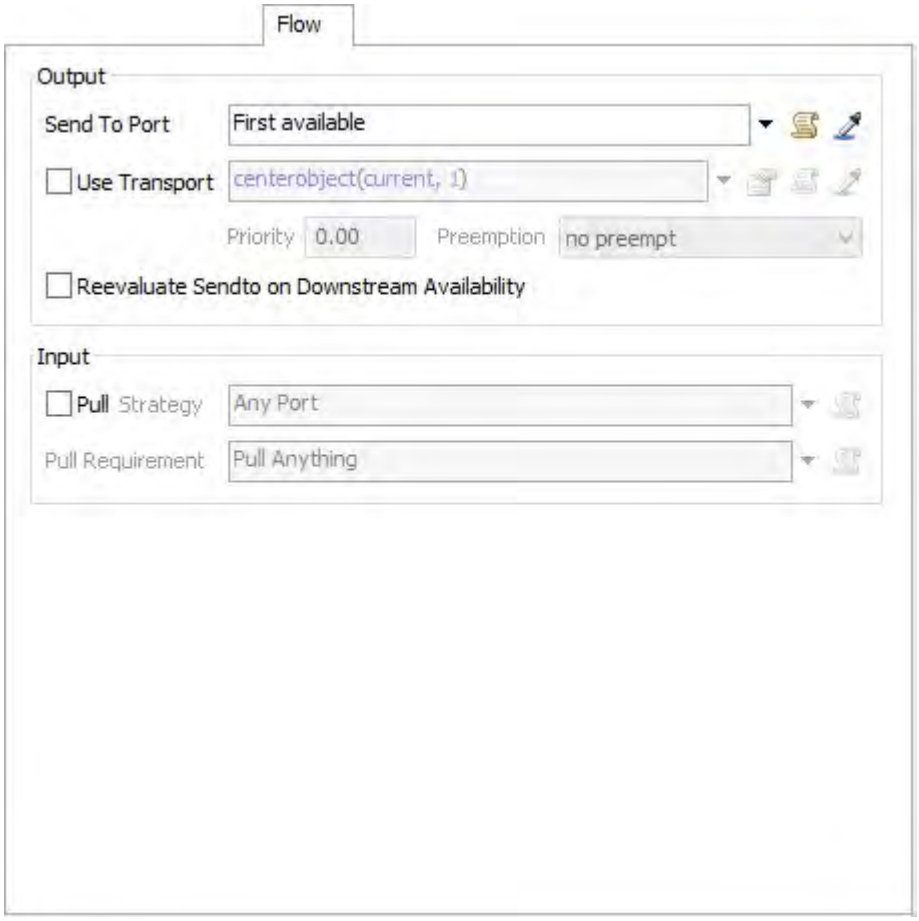
Convey Items Across Combiner Length - If checked, flowitems will travel across the Combiner during their process time.

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 Properties window and reopen it in order for your changes to register. To update this list dynamically during a model run, use the Update Combiner Component List picklist option in the OnEntry trigger.

TargetQuantity - The required number of flowitems to be received through the associated input port for each operation.

This Page is Used By

Combiner



For detailed information on this functionality, refer to the FixedResource.

Output

These properties determine how the object sends flowitems downstream.

Send To Port - This picklist 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 picklist.

Use Transport - If this box is checked, the object will request a transport using the Request Transport From picklist to move the flowitem downstream. If it is not checked, the flowitem will be moved automatically.

Request Transport From - This picklist is only available 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 picklist.

Priority - This parameter is only available 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 available if "use transport" is checked. If set to one of the preempting values, 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. For more information on preempting task sequences, see Task Sequence Preempting.

Reevaluate Sendto on Downstream Availability - If checked, the Send To Port for a released flow item 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.

Input

These properties define how an object pulls flowitems from upstream objects. See

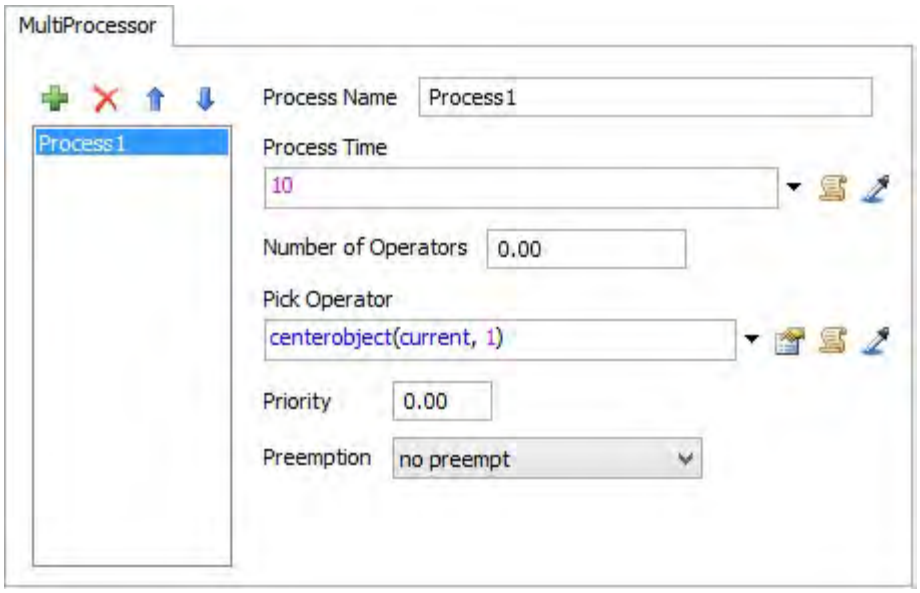
Pull - If this box is checked, the object will pull flowitems from upstream objects. The upstream objects should open all their output ports to allow the object to pull the flowitems it needs.

Pull Strategy - This parameter is only visible if "Pull" is checked. This picklist 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 Strategy field will only be evaluated once right after each flowitem exits the Processor. See Receive From Port picklist.

Pull Requirement - This parameter is only accessible if "Pull" is checked. This picklist 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. See Pull Requirement picklist.

This Page is Used By

Source
Queue
Processor
Combiner
Separator
MultiProcessor
Rack



- Click this button to add a new process to the process list below.

- Click this button to delete a process from the process list below.

- Click this button to move the selected process up in the process list below.

- Click this button to move the selected process down in the process list below.

The process list contains each of the processes for this multiprocessor. To edit a process, highlight it in the list and change its name and other details to the right.

Process Details

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 picklist determines how long a processor spends processing a single flowitem. See also Cycle Time picklist.

Number of Operators - This number determines how many operators the object will use during its process time.

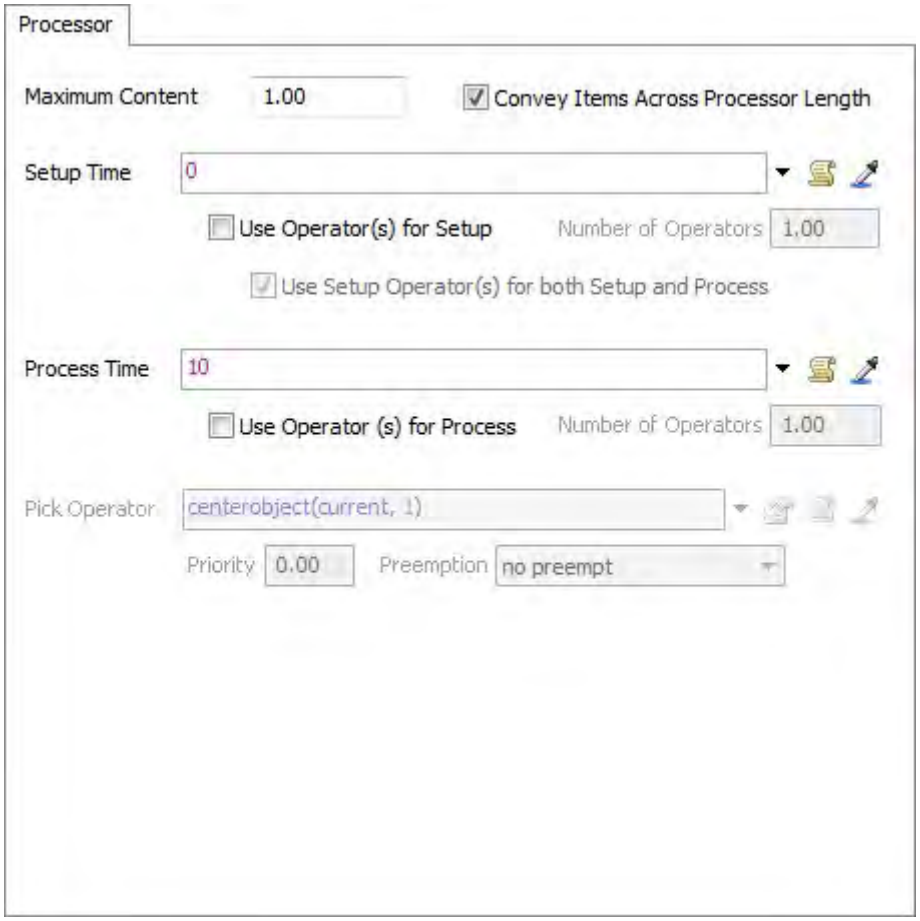
Pick operator - This picklist returns a reference to the operator or dispatcher that the object is using during the given process. See also Pick Operator picklist.

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.

Preemption - Sets the preempt value for calling operators. See the Task Sequence Preempting page for more information on preemption.

This Page is Used By

MultiProcessor



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.

Setup

Setup Time - This picklist defines the amount of time that the processor waits after receiving a flowitem to begin processing that flowitem. See Setup Time pick list.

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.

Number of 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.

Use the Setup Operator(s) for both Setup and Process - 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.

Process

Process Time - This picklist determines how long a processor spends processing a single flowitem.

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 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.

Pick Operator

These fields are only visible when either of the “Use Operator(s)” boxes is checked.

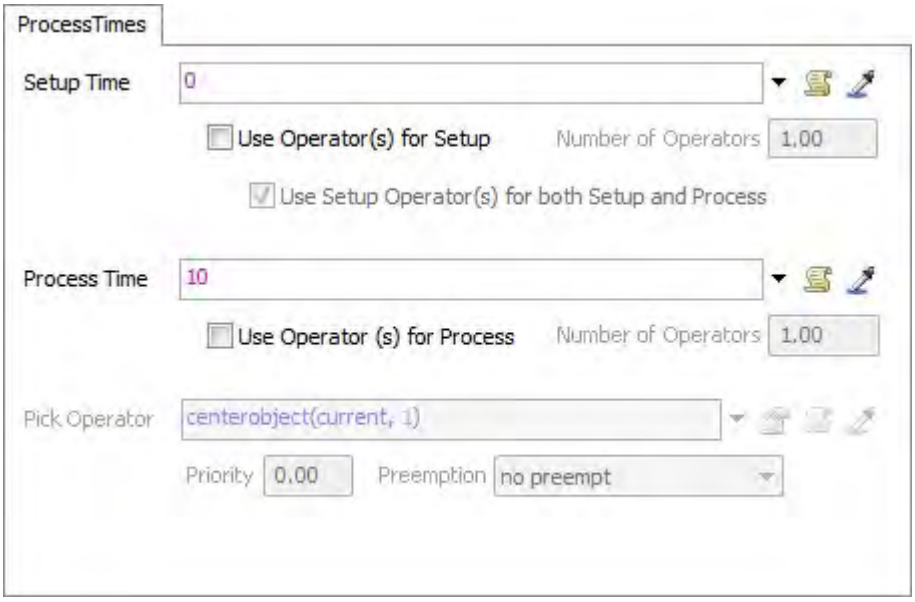
Pick operator - This picklist returns a reference to the operator or dispatcher that the object is using during setup or process time. See Process Dispatcher picklist.

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.

Preemption - Sets the preempt value for calling operators. This may cause the operator to perform tasks that would normally not be allowed. See the Task Sequence Preempting page for more information on preemption.

This Page is Used By

Processor



Setup

Setup Time - This picklist defines the amount of time that the processor waits after receiving a flowitem to begin processing that flowitem. See Setup Time pick list.

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.

Number of 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.

Use the Setup Operator(s) for both Setup and Process - 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.

Process

Process Time - This picklist determines how long a processor spends processing a single flowitem.

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 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.

Pick Operator

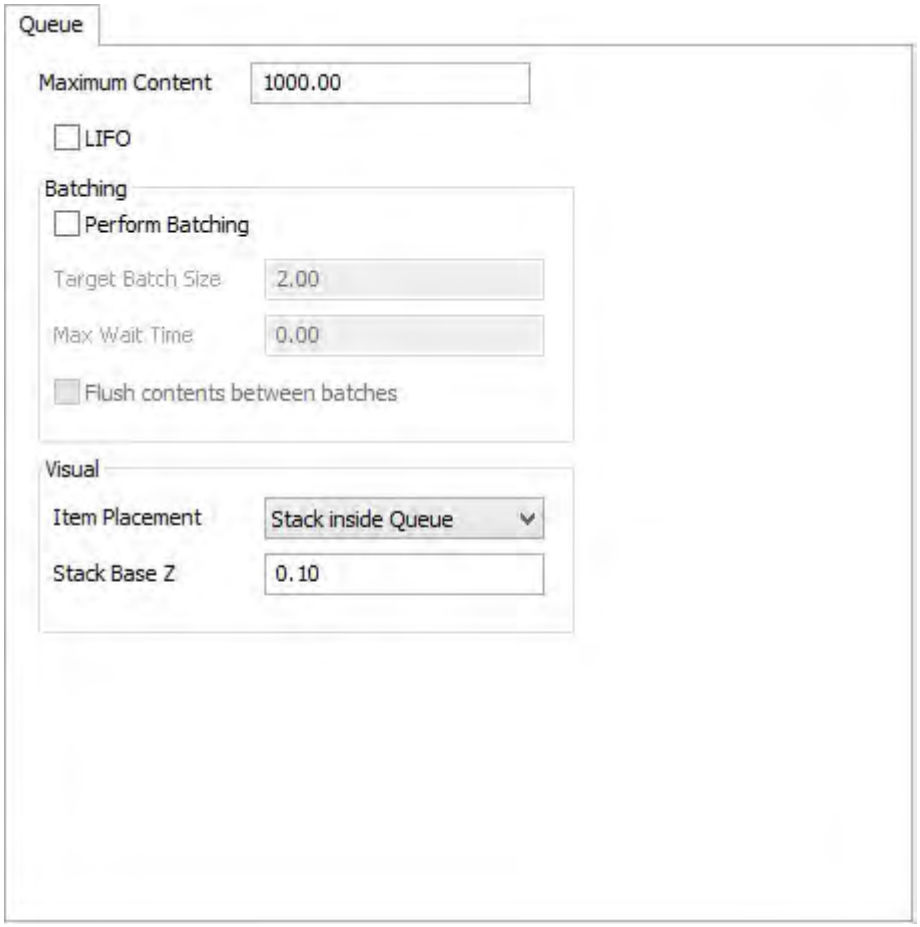
These fields are only visible when either of the “Use Operator(s)” boxes is checked.

Pick operator - This picklist returns a reference to the operator or dispatcher that the object is using during setup or process time. See Process Dispatcher picklist.
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.

Preemption - Sets the preempt value for calling operators. This may cause the operator to perform tasks that would normally not be allowed. See the Task Sequence Preempting page for more information on preemption.

This Page is Used By

Combiner
Separator



Maximum Content - This is the maximum number of flowitems the queue can hold at once.

LIFO - If this box is checked the queue will act as a "last in first out" (LIFO) queue, otherwise it will act as a "first in first out" (FIFO) queue.

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

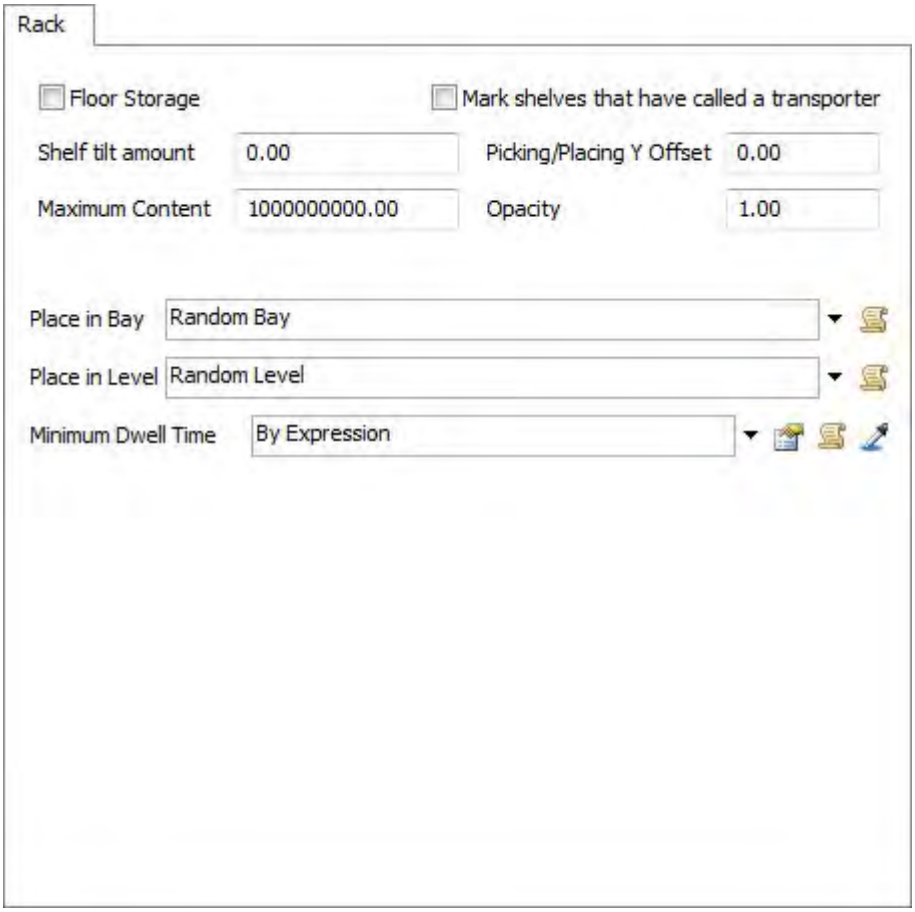
These properties define how the queue locates the flowitems within itself when they enter. Item Placement - This defines how the flowitems are placed in the queue visually.

- 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.

This Page is Used By

Queue



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.

Mark Shelves that have called a transporter - This check will highlight the shelve in a red color when it has called the transporter for pickup.

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.

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.

Maximum Content - This number defines how many flowitems the rack will be allowed to hold at a given time.

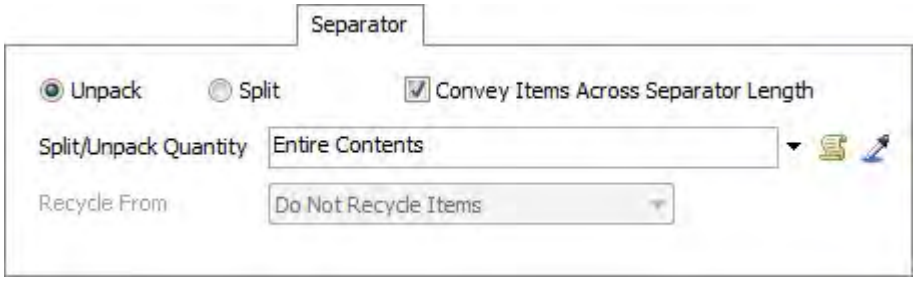
Opacity - This value allows the drawing of the rack to be translucent, so that if there are several racks in the same area, many of them can be seen because of the translucency of the racks in front. A value of 0 means totally transparent, and a value of 1 means totally opaque.

Place in bay - This picklist is called when a flowitem is entering the rack. It returns which bay the flowitem will be placed in. See Place in Bay picklist.

Place in level - This picklist is called when a flowitem is entering the rack. It returns which level the flowitem will be placed in. See Place in Level picklist.
Minimum Dwell Time - This picklist 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 picklist.

This Page is Used By

Rack



Unpack - If this button is checked, the separator will assume that the flowitem that entered contains other flowitems that need to be removed.

Split - If this button 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.

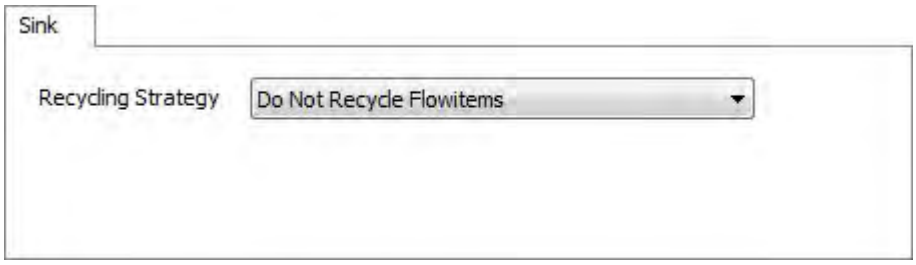
Split/Unpack Quantity - This picklist returns how many flowitems will be unpacked or duplicated by the separator. Refer to the split quantity picklist.

Recycle From (Split mode only) - In split mode, the flowitems that are created can be pulled from recycled flowitems rather than creating a new copy. You can use this option to specify what type of flowitem should be created. To learn more about recycling, refer to the Sink Page.

This Page is Used By

Separator

Sink Page

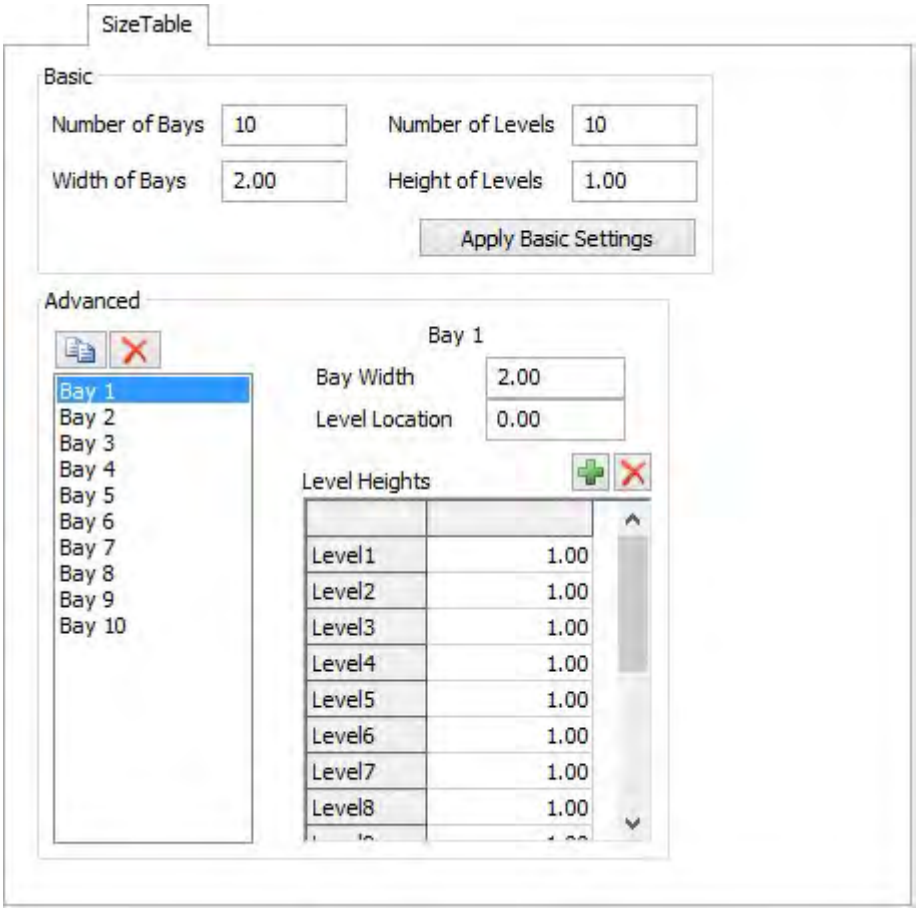


Recycling Strategy - The recycling strategy drop-down list lets you specify how the Sink recycles flowitems. Recycling flowitems can significantly increase the speed of your model. By default, the Sink simply destroys flowitems that enter. To configure the sink to recycle flowitems, select from the drop-down 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



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 at the top. If you want to specified different sizes for different bays and levels of the rack, then you can use the Advanced panel at the bottom to configure each bay individually.

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.

 - This button duplicates the currently selected bay and adds it to the end of the rack.

 - This button deletes the currently selected bay in the rack.

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.

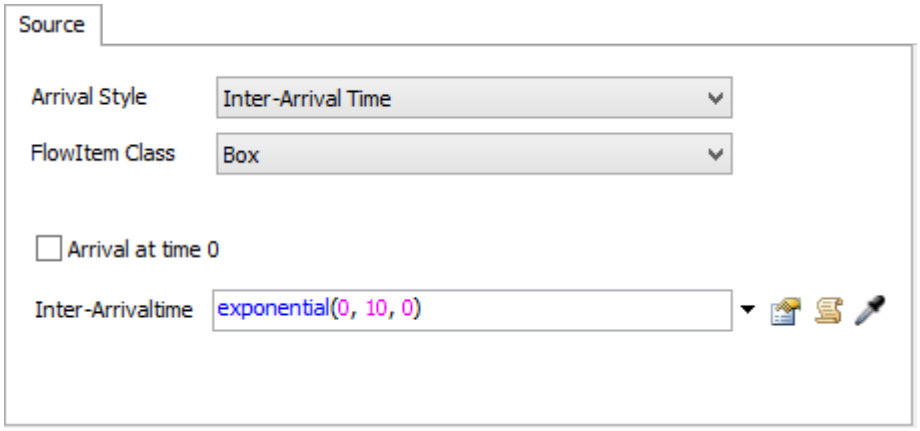
Level Heights Table - Here you can specify the height of each level in the currently selected bay.

 - This button adds a level to the end of the currently selected bay.

 - This button deletes a level from the end of the currently selected bay.

This Page is Used By

Rack



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 types 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

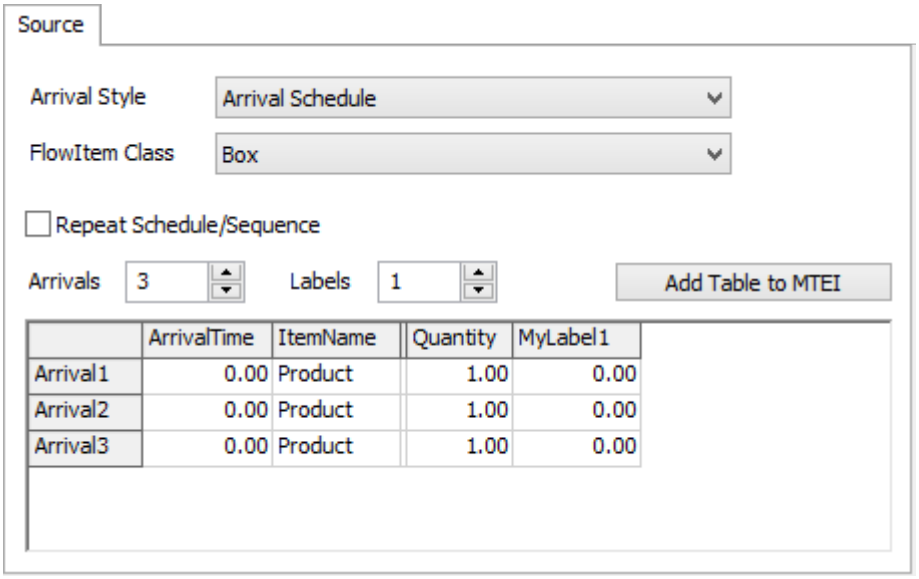
This is used to define what class of flowitem the source will create. To view and edit flowitem classes, select *Go to Flowitem Bin...* in the drop down, or go to the Toolbox Flowitem Bin.

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.

Inter-Arrival time - A function that returns the amount of time the source should wait before creating the next flowitem.

Arrival Schedule/Sequence



These fields define how the source creates flowitems when Arrival Schedule or Arrival Sequence is selected as the Arrival Style.

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.

Number of Arrivals - This specifies how many rows are in the arrival table. Change the value in this field and hit Refresh Arrivals to update the table view.

Number of Labels - This specifies how many columns are in the arrival table. Change the value in this field and hit Refresh Labels to update the table view. Additional columns will be added for labels and their initial values that will be added to the flowitems when they are created.

Add Table to MTEI - This buttons adds the table as a row in the Multiple Table Excel Import.

ArrivalTime - (Arrival Schedule only) This is the time in model time units that the arrival will occur.

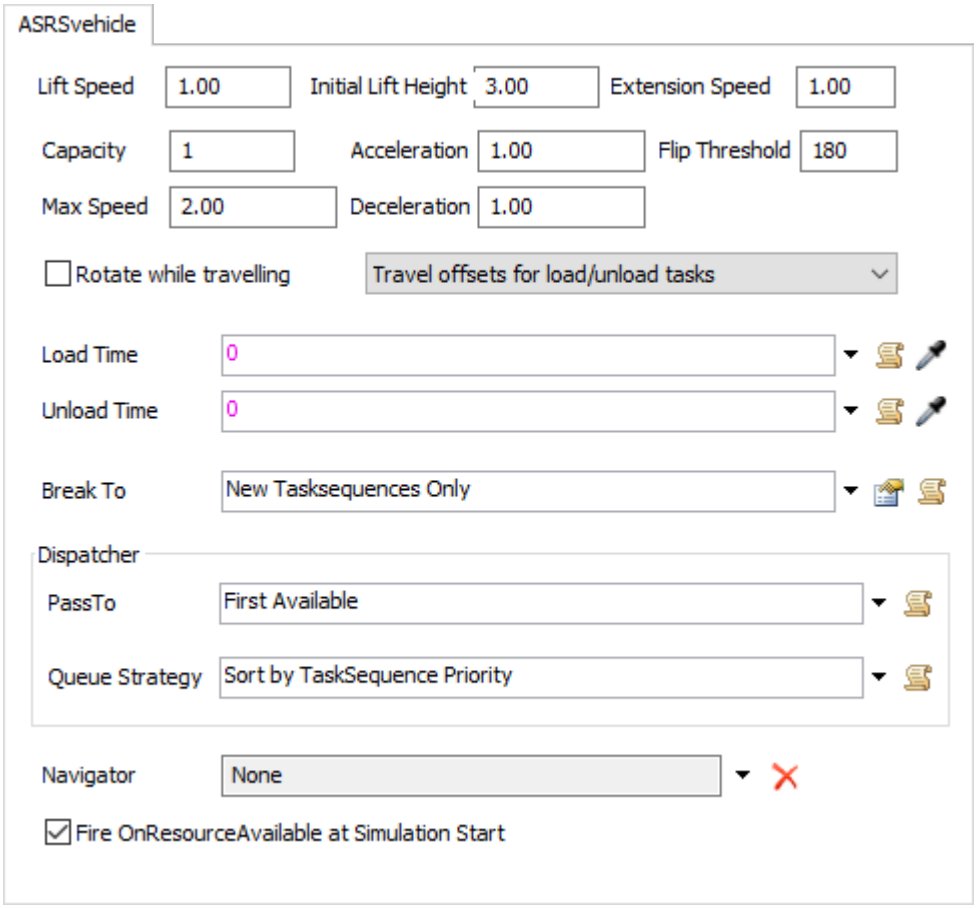
Item Name - Specifies the name of the flowitem when it's created.

Quantity - This number specifies how many flowitems will be created during this arrival.

Labels - This specifies all labels that will be added to created flowitems and their initial values.

This Page is Used By

Source



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.

Task Executer fields - This page has many of the same controls as the TaskExecuter Page. They are described in more detail there.

This Page is Used By

ASRSvehicle

BasicTE Page

Note: For more information on offset travel, refer to the TaskExecuter's documentation on offset travel.

OnBeginOffset - This picklist 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

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.

OnUpdateOffset - This picklist is fired before the view is refreshed. Here is where the object updates its location based on its current offset operation.

OnFinishOffset - This picklist 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 picklist 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 picklist 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.

nrofstopsnode: 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 picklist 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:

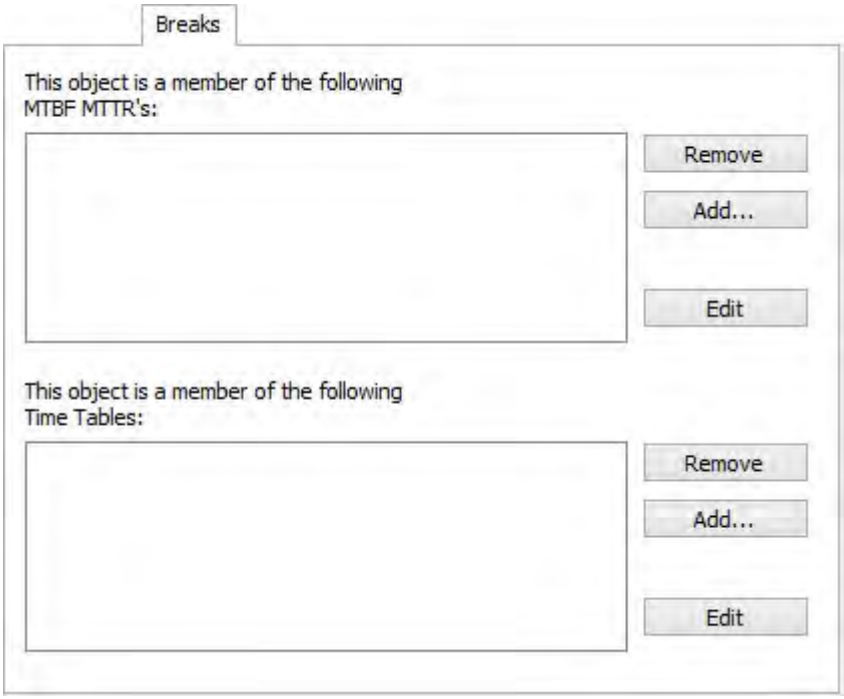
ADV_FUNC_CLASSTYPE: This is a request to get the type of class of the object. The classtype should be returned as an integer with certain bits set high. You can construct this value using the bitwise OR operator | and several classtype macros. For example, a FixedResource's classtype is:
CLASSTYPE_FLEXSIMOBJECT > 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



MTBF MTTR Member List - This is a list of all the MTBF MTTR objects that have this object as one of its members. 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. For more information about MTBF MTTR objects, refer to the Modeling Tools section about MTBF/MTTR objects.

Remove - This button removes the object from the selected MTBF MTTR object's member list.

Add.. - This button opens a listbox of all the MTBF MTTR objects in the model. You can select an MTBF MTTR object from the list to add this object to that MTBF MTTR object. You can also select "Add New MTBF MTTR" to create a new MTBF MTTR object in your model and add this object to that MTBF MTTR object.

Edit - This button allows you to edit the MTBF MTTR object's properties, including the timing of the breakdowns and repairs. For more information about MTBF MTTR objects, refer to the Modeling Tools section about MTBF/MTTR objects.

Time Tables Member List - This is a list of all the Time Table objects that have this object as one of its members. Each Time Table object can be connected to more than one object in the model. And each object can be controlled by more than one Time Table object. For more information about Time Table objects, refer to the Modeling Tools section about Time Tables.

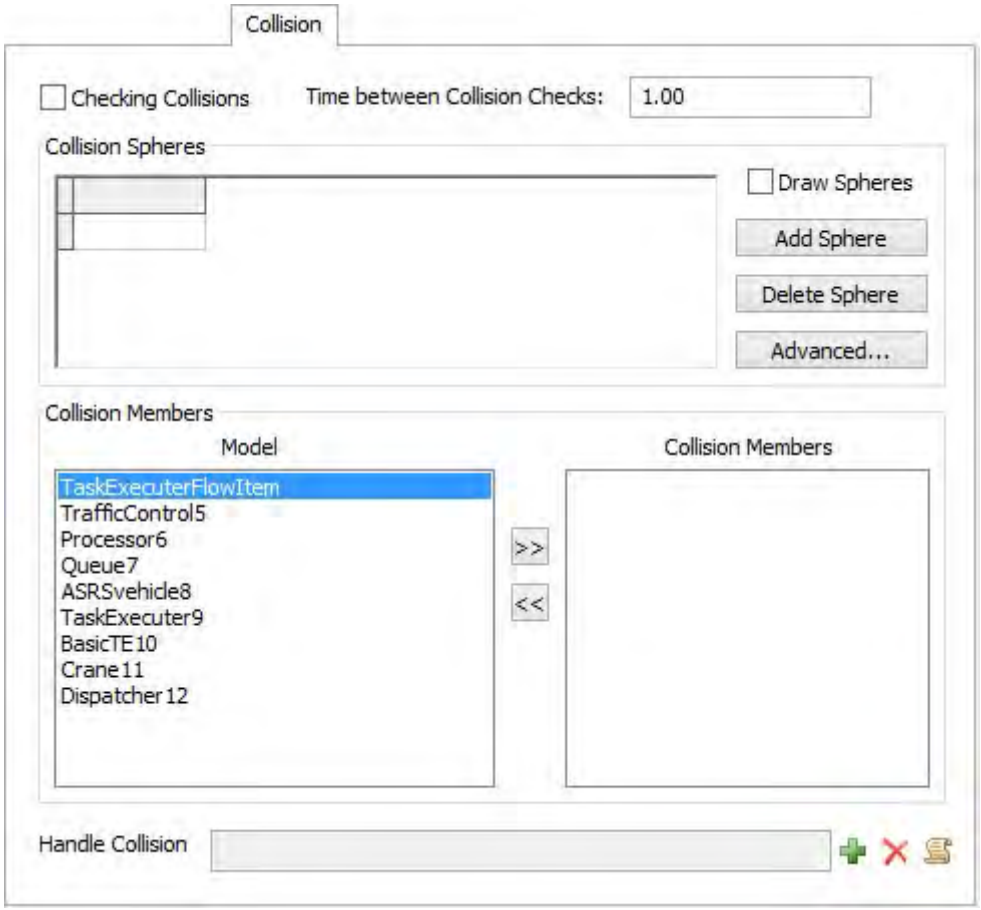
Remove - This button removes the object from the selected Time Table object's member list.

Add.. - This button opens a listbox of all the Time Table objects in the model. You can select an Time Table object from the list to add this object to that Time Table object. You can also select "Add New Time Table" to create a new Time Table object in your model and add this object to that Time Table object.

Edit - This button allows you to edit the Time Table object's properties, including the timing and duration of scheduled breakdowns. For more information about Time Table objects, refer to the Modeling Tools section about Time Tables.

This Page is Used By

TaskExecuter
Transporter
Operator
Crane
ASRSvehicle
Robot
Elevator



For more information on collision detection functionality, refer to the TaskExecuter.

Checking Collisions - Check this box to turn on collision detection. time between its collision members' collision checks.

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.

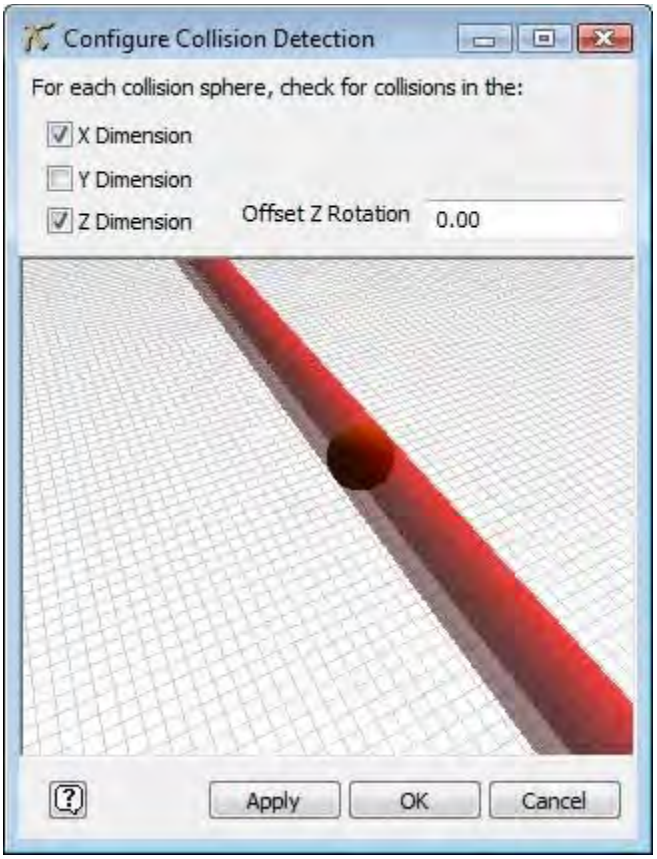
Collision Spheres - This table is used to define one or more collision spheres on the object.

Draw Spheres - Check this box if you want the collision spheres visible around the CollisionObject.
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 TaskExecuter to exclude certain axes when checking its spheres for collisions. This let's 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

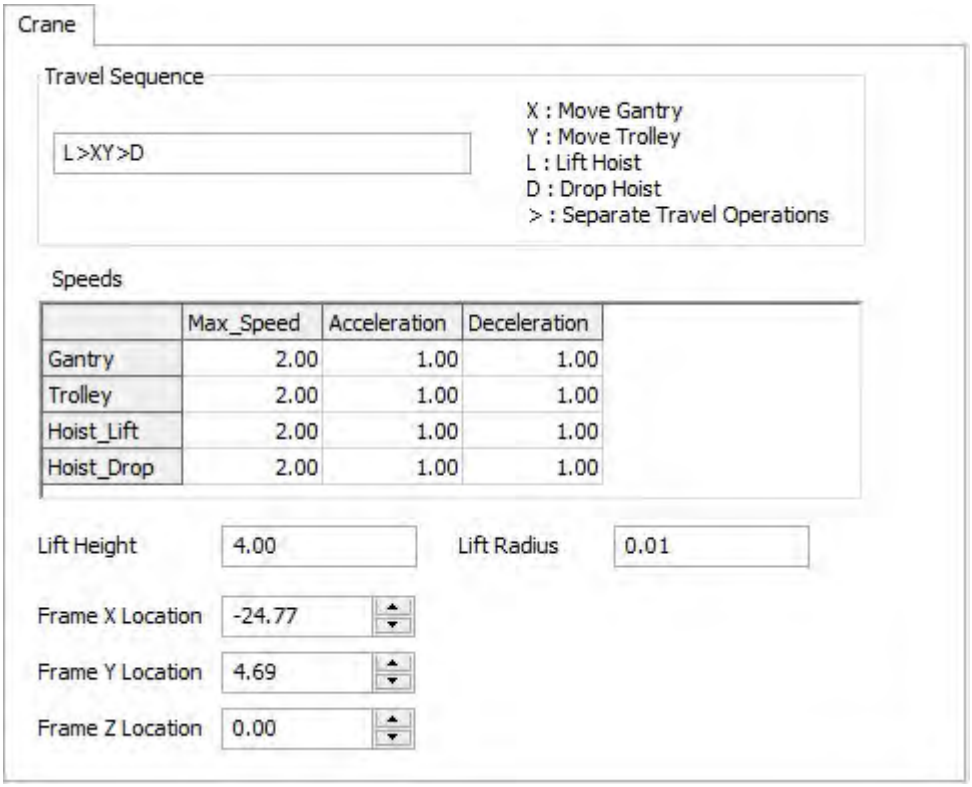
axis that is not parallel with the normal axis. Note that the X, Y, and Z dimensions are according to the global coordinate system, and not according to the individual object's coordinate system.



Collision Members - On the left is a list of model objects that can be added to the TaskExecuter'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 TaskExecuter'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 picklist allows the user to define what happens when a collision takes place. See Collision Trigger picklist.

This Page is Used By

TaskExecuter
Transporter
Operator
Crane
ASRSvehicle
Robot
Elevator



Travel Sequence - Here you can specify the order in which the crane performs travel operations. Refer to the crane specifications for more information.

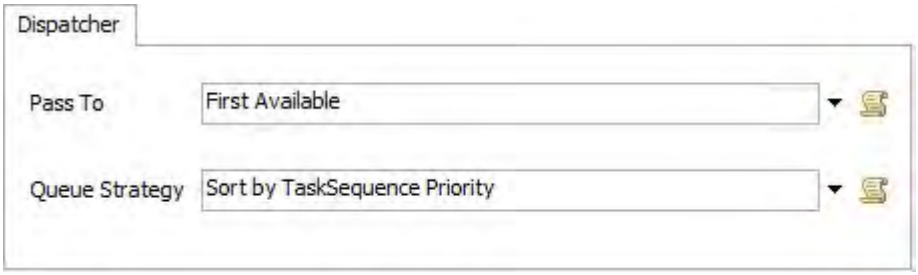
Speeds Table - Here you specify the max speed, acceleration and deceleration for each of the 4 operations the crane will do. Note that these operations only apply to offset travel. If the crane is connected to a network, then when it is on the network, only the normal maxspeed, acceleration and deceleration specified in the TaskExecuter 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.

This Page is Used By

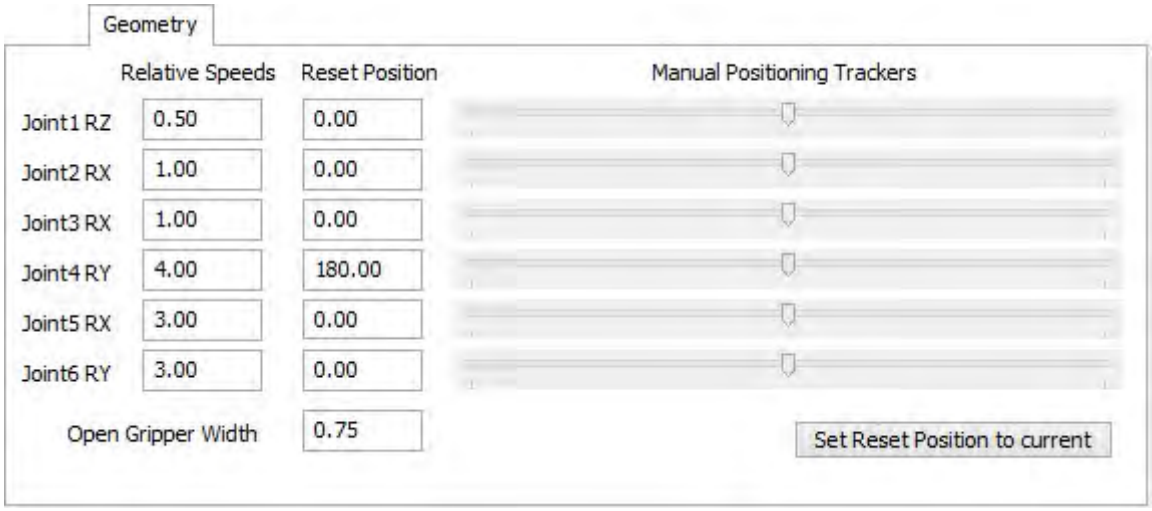


Pass To - This picklist returns the output port number that the task sequence should be dispatched to. If 0 is returned, then the task sequence will be queued up using the below mentioned queue strategy, and then will be dispatched to the first available mobile resource. If -1 is returned, then the Dispatcher will do absolutely nothing. In such a case you would use the movetasksequence() and dispatchtasksequence() commands to execute dispatching logic yourself. See Pass To picklist.

Queue Strategy - This picklist 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 picklist.

This Page is Used By

Dispatcher



Relative Speeds- The options in this column control the relative speeds for each joint..

Reset Position- The options in the column control the position to which the robot will return on reset.

Manual Positioning Trackers- The sliders in this column control the current position of each joint.

Open Gripper Width- This option controls how far open the gripper is when it's open.

Set Reset Position to current- This option saves the current position of the robot as its reset position.

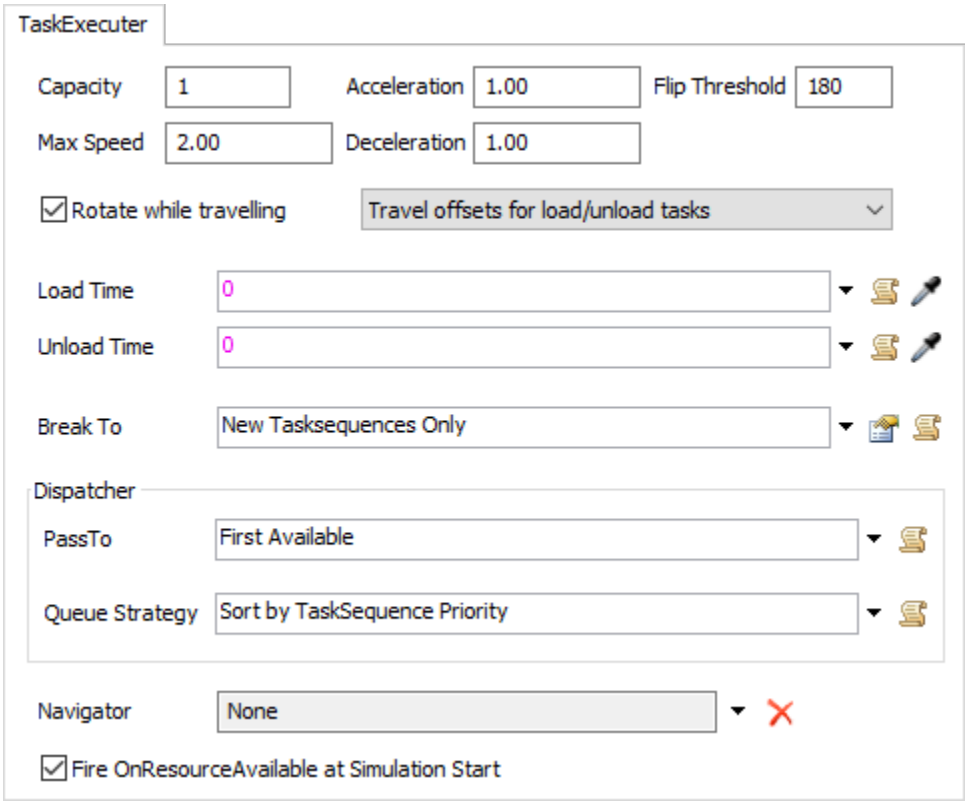
This Page is Used By

Robot

For more information, refer to the Robot topic.

This Page is Used By

Robot



Capacity - This number is maximum number of flowitems that the Task Executer can carry at a given time.

Max speed - This is the fastest that the Task Executer can travel.

Acceleration - This number is how fast the Task Executer 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 Task Executer loses speed as it approaches its destination.
Flip Threshold - When the angle between the transporter/operator and the destination node meets or exceeds this value, the transporter/operator will flip (mirror image) in order to be facing the correct direction. This option will not affect the statistics of the model if checked or unchecked. It is simply for visualization.

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.
Travel offsets for load/unload tasks - This box determines how the object will perform offset travel operations.

- Travel offsets for load/unload tasks - If this option is selected, the transporter/operator will move to the exact point where the flowitem is being picked up or dropped off.
- Do not travel offsets for load/unload tasks - If this option is selected, the transporter/operator 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.
- Do not travel offsets and block space on network - This option only applies when the object is connected to a network of nonpassing network node 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.

- Use navigator for offset travel - If this option is selected, then the object will not use its own mechanism for traveling offsets, but will instead make a request to its navigator to travel to the target location. This can be useful especially when the object is connected to an AStar travel network.

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.

Load time - This picklist returns how long it takes this Operator or Transporter to load the flowitem.

Unload time - This picklist 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. Dispatcher Pass To - This picklist 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 picklist.

Dispatcher Queue Strategy - This picklist 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 picklist.

Navigator - This specifies which Navigator the Task Executer object will use. If navigators are available in the model, they will be displayed in the drop down. To remove the TaskExecuter from using any Navigator, press the  .

Depending on what objects are in your model, various navigators may be available. Below is a list of those navigators and their purpose:

- *None* - When set to None, the TaskExecuter will execute travel tasks by moving in a straight line from their current position to the end position. They will also offset travel in a straight line to pickup flowitems when necessary.

- *DefaultNetworkNavigator* - The DefaultNetworkNavigator is available when NetworkNodes have been added to the model. Connecting a TaskExecuter to a NetworkNode using an A connection will automatically set their Navigator to be the DefaultNetworkNavigator. The DefaultNetworkNavigator will build a distance/routing table for all network nodes in the model. When a travel task is given to the TaskExecuter, the navigator will move the TaskExecuter from NetworkNode to NetworkNode along the shortest path to the final destination.

- *AGVNetwork* - The AGVNetwork is available if AGV paths have been added to the model. TaskExecuters using the AGVNetwork will travel along AGV paths based upon the properties setup in the AGV Network Properties.

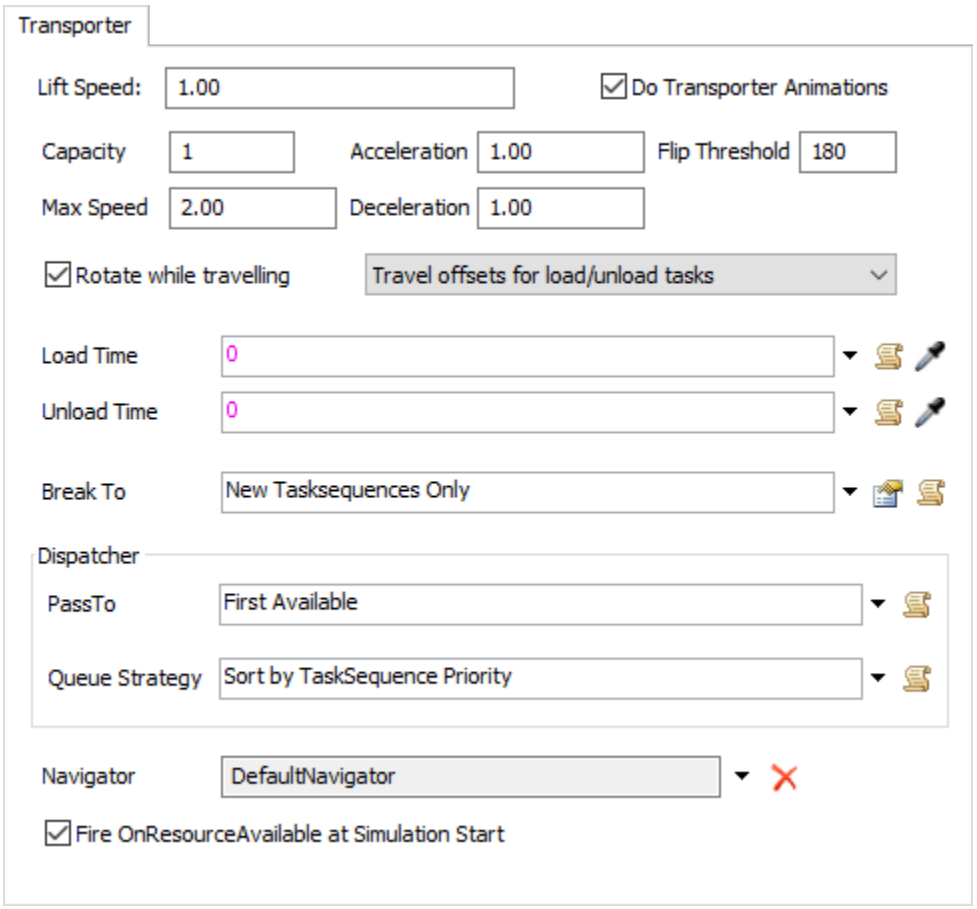
- *AStarNavigator* - The AStarNavigator is available if the AStar Navigator has been added to the model. When connected to the AStarNavigator, the TaskExecuter will travel the shortest path from the current position to the final destination based upon any dividers or barriers that may be in the way. For more information see the AStar page.

Fire OnResourceAvailable at Simulation Start - If checked, the object's OnResourceAvailable trigger will be fired when the simulation starts.

This Page is Used By

TaskExecuter
Transporter
Operator
Crane
ASRSvehicle
Robot
Elevator

Transporter Pages



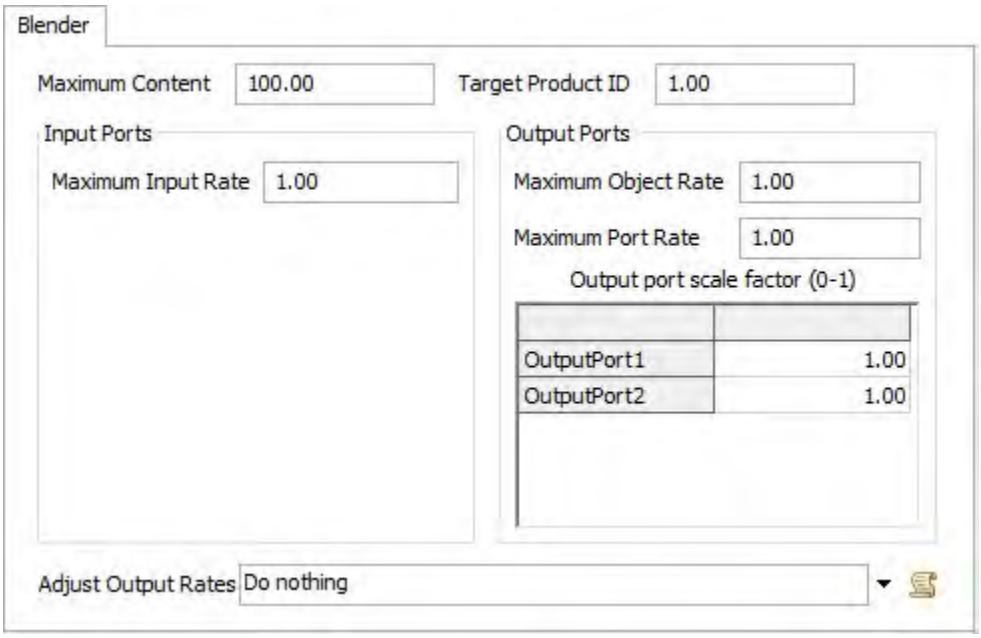
Lift speed - This number is how fast the lifts on the Transporter move up and down.

Task Executer fields - This page has many of the same controls as the TaskExecuter Page. They are described in more detail there.

This Page is Used By

Transporter

Blender Page



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 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 properties 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 properties 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.

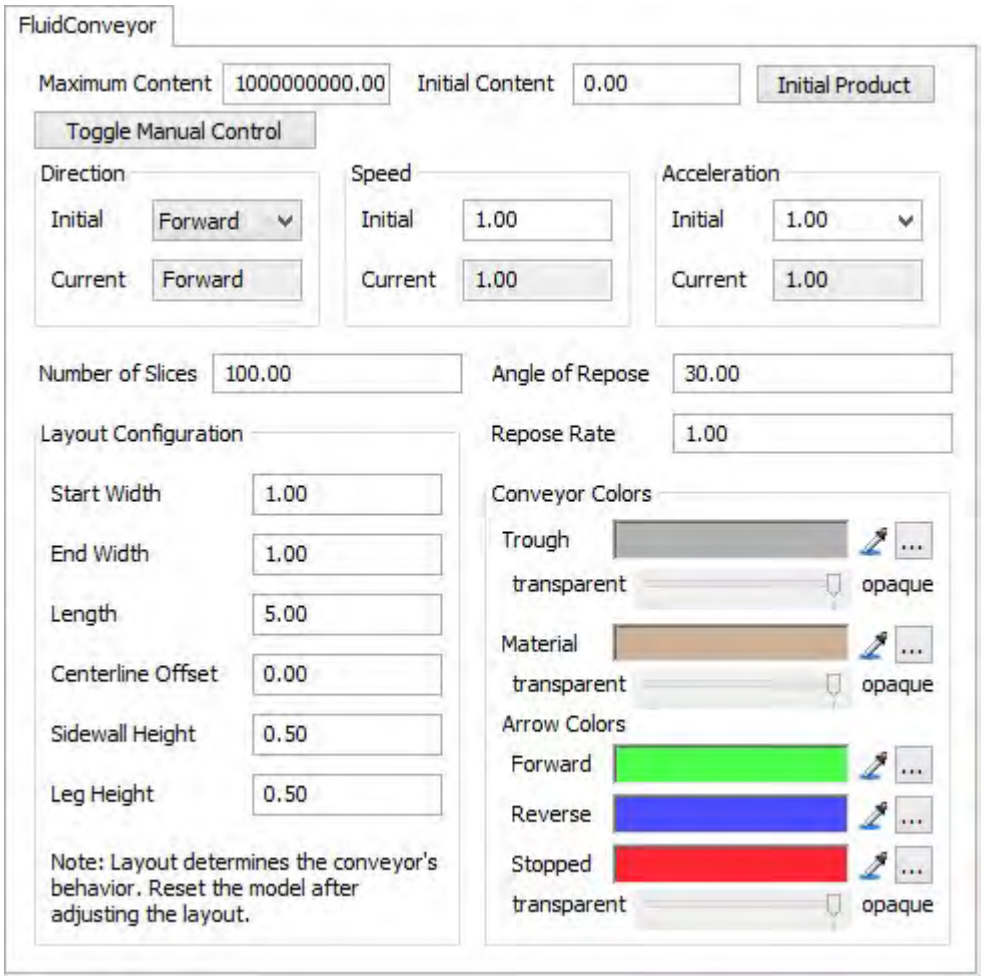
Maximum Port Rate - The maximum rate that material will leave this object through any one port.

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 modeler to change the rate of individual output ports during a model run.

This Page is Used By

FluidBlender

FluidConveyor Page

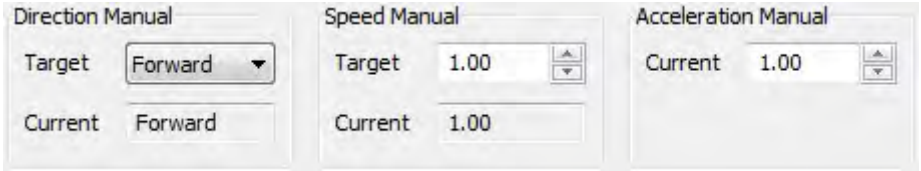


Maximum Content - The maximum amount of fluid material that this object can hold at any time.

Initial Content - The amount of material that is in the object when the model is reset.

Initial Product - This opens the Initial Product Window which allows the modeler to define the Product ID and sub-component mix of the material that is in this object.

Toggle Manual Control - Toggles manual speed control. You can change the direction, target speed and acceleration while the model is running. The manual speed control is designed to help you understand more about how the conveyor will react to changes in speed or direction.



Direction - Specifies the initial direction of the conveyor. The current direction is also displayed while the model is running.

Speed - Specifies the initial speed of the conveyor in the given initial direction. Speed values for the fluid conveyor cannot be negative. The current speed is also displayed while the model is running.

Acceleration - Specifies the initial acceleration of the conveyor. Infinite acceleration is defined as 0. The current acceleration is also displayed while the model is running.

Number of Slices - The number of slices of fluid material that are placed along the length of the conveyor. The more slices, the better the resolution for displaying the volume of fluid. However, the more slices in the fluid conveyor, the more computations, causing your model to run slower.

Angle of Repose - Defines the material's steepest angle of descent of the slope relative to the horizontal plane. This angle ranges between 0 and 90 degrees.

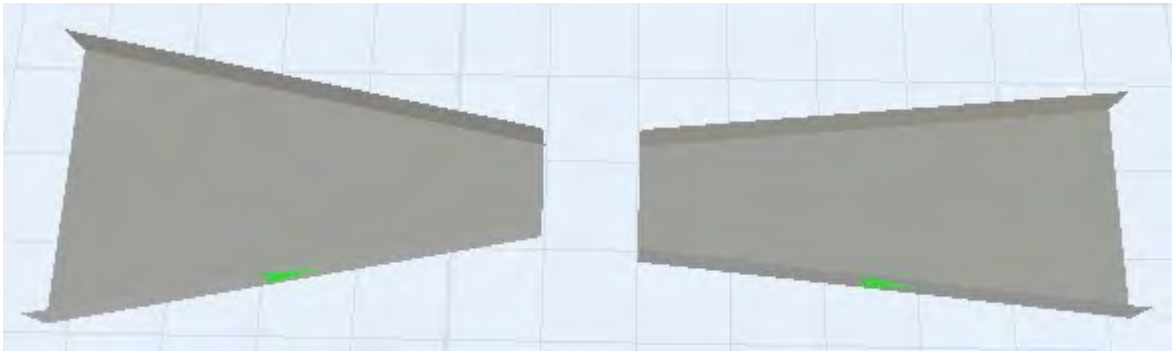
Repose Rate - The repose rate defines how quickly a reposing pile of material will reach its natural resting state (based on the angle of repose). A value of 0 will cause the Angle of Repose to be ignored. The repose subroutine will be run the number of times specified in this field (the larger the number, the more processing time it well take each tick te repose).

Layout Configuration

The layout of the fluid conveyor affects the conveyor's behavior, so the model should be reset after the layout has been changed to apply the changes.

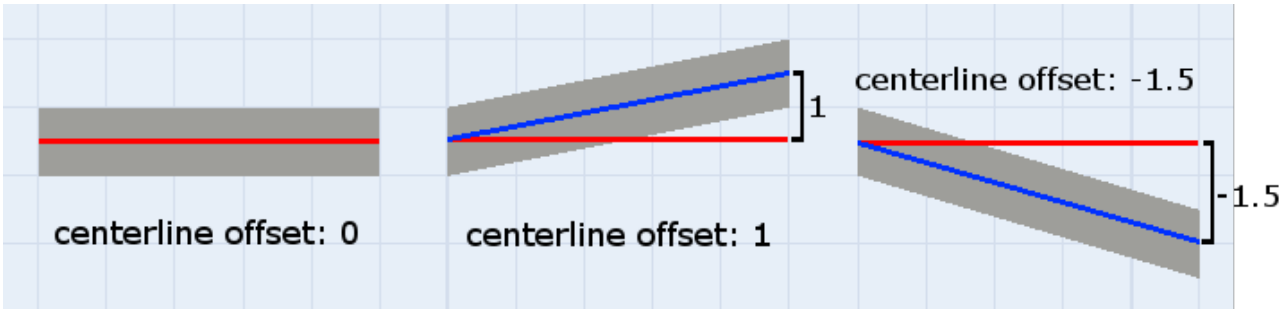
Start Width - Specifies the width of the start of the conveyor

End Width - Specifies the width of the end of the conveyor



Length - Specifies the length, or X dimension of the conveyor.

Centerline Offset - The centerline offset skews the conveyor's trough. The value specifies the distance and direction (can be positive or negative) that the centerline of the trough is offset from the standard centerline



Sidewall Height - Specifies the sidewall height of the conveyor's trough. This value is purely visual and has no effect on the behavior of the fluid conveyor.

Leg Height - Specifies the leg height of the conveyor. This value is purely visual and has no effect on the behavior of the fluid conveyor.

Conveyor Colors

Use the or to Sample a color or press "..." to choose a color.

Trough - Sets the color of the trough and legs. You may also change the transparency of the trough (allowing you to see the material's height profile through the sidewall).

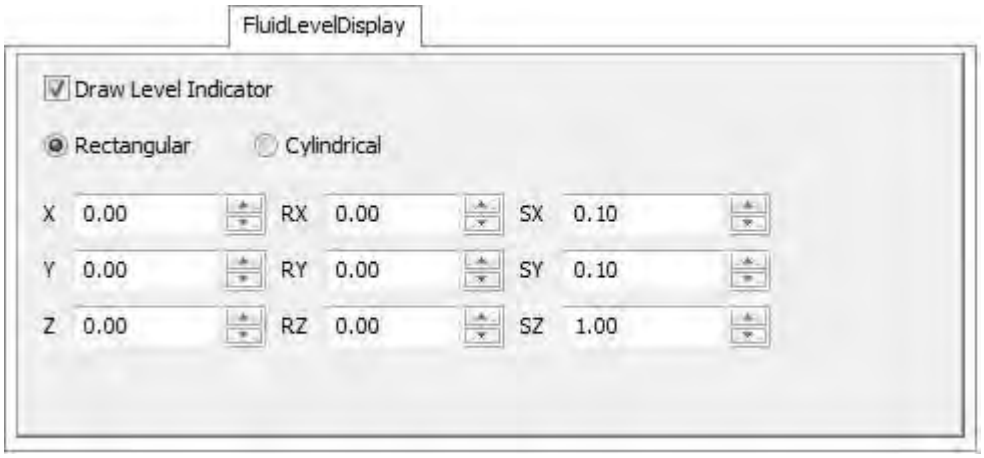
Material - Sets the color of the material. You may also change the transparency.

Arrow Colors - Set the color of the three directions of the fluid conveyor, Forward, Reverse, and Stopped. You may also change the transparency.

This Page is Used By

Fluid Conveyor

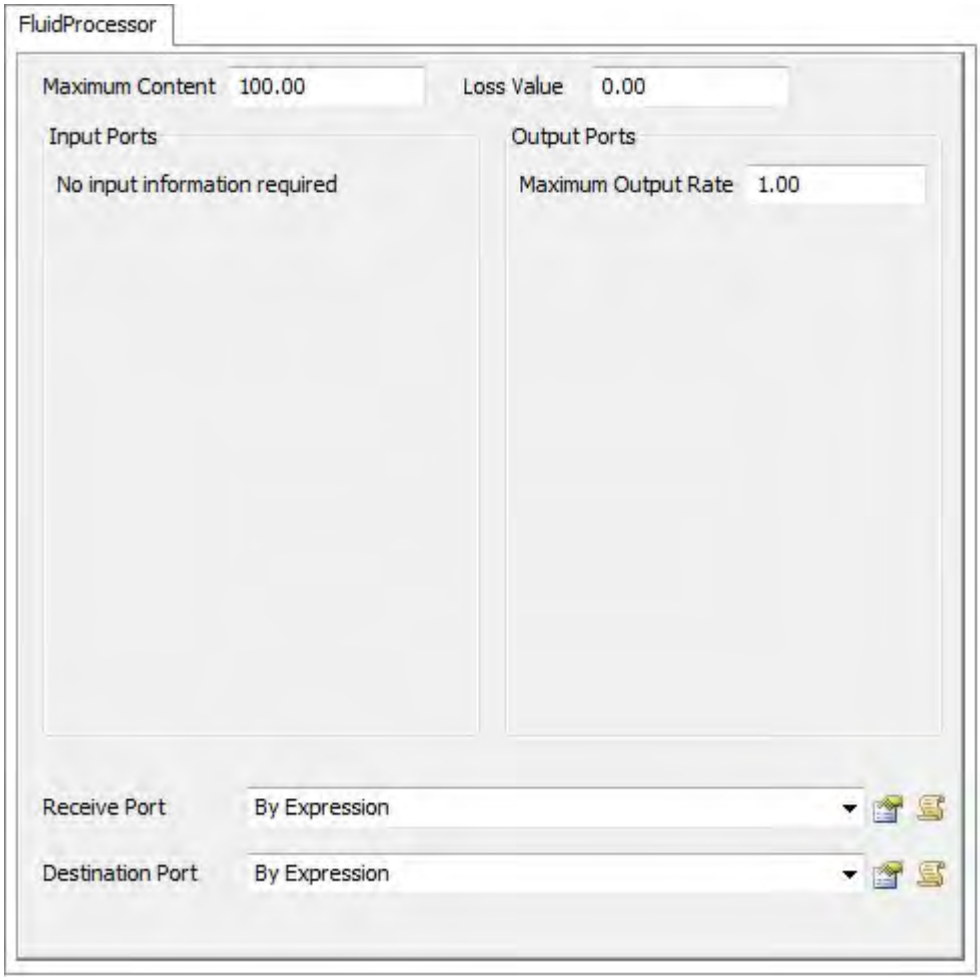
FluidLevelDisplay Page



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 (1.0 being 100%) of the size of the object.

This Page is Used By

FluidBlender
FluidGenerator
FluidMixer
FluidProcessor
FluidSplitter
FluidTank
FluidToItem
ItemToFluid



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 - 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 - 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 modeler has to define.

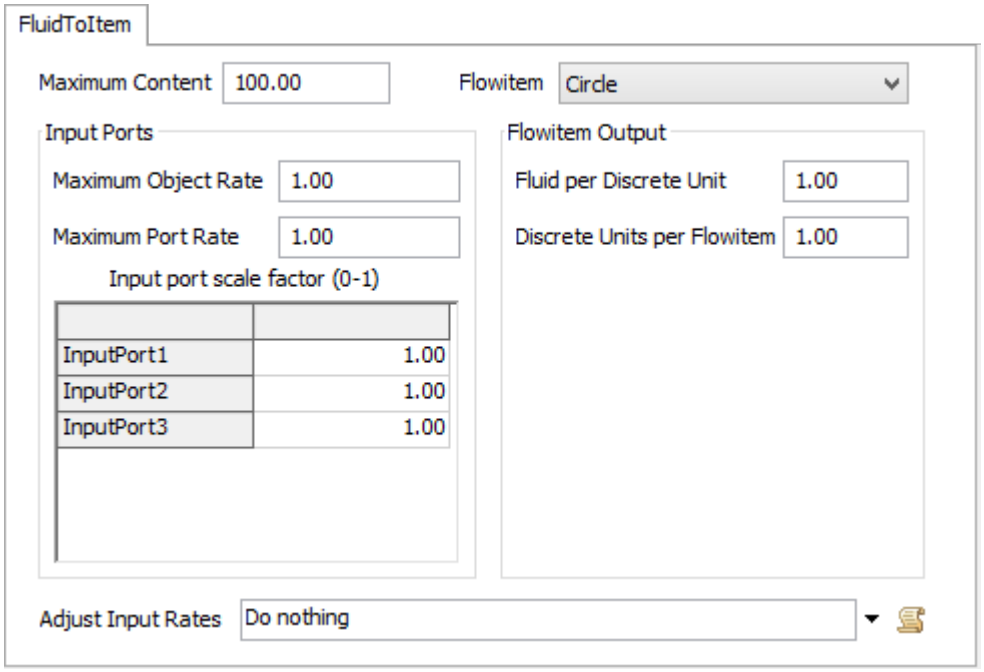
Output Ports

These properties 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 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 properties 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.

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 modeler to change the rate of individual input ports during a model run.

Flowitem Output

These properties define when the FluidToItem creates a flowitem and some information that will be defined on the flowitem when it is created.

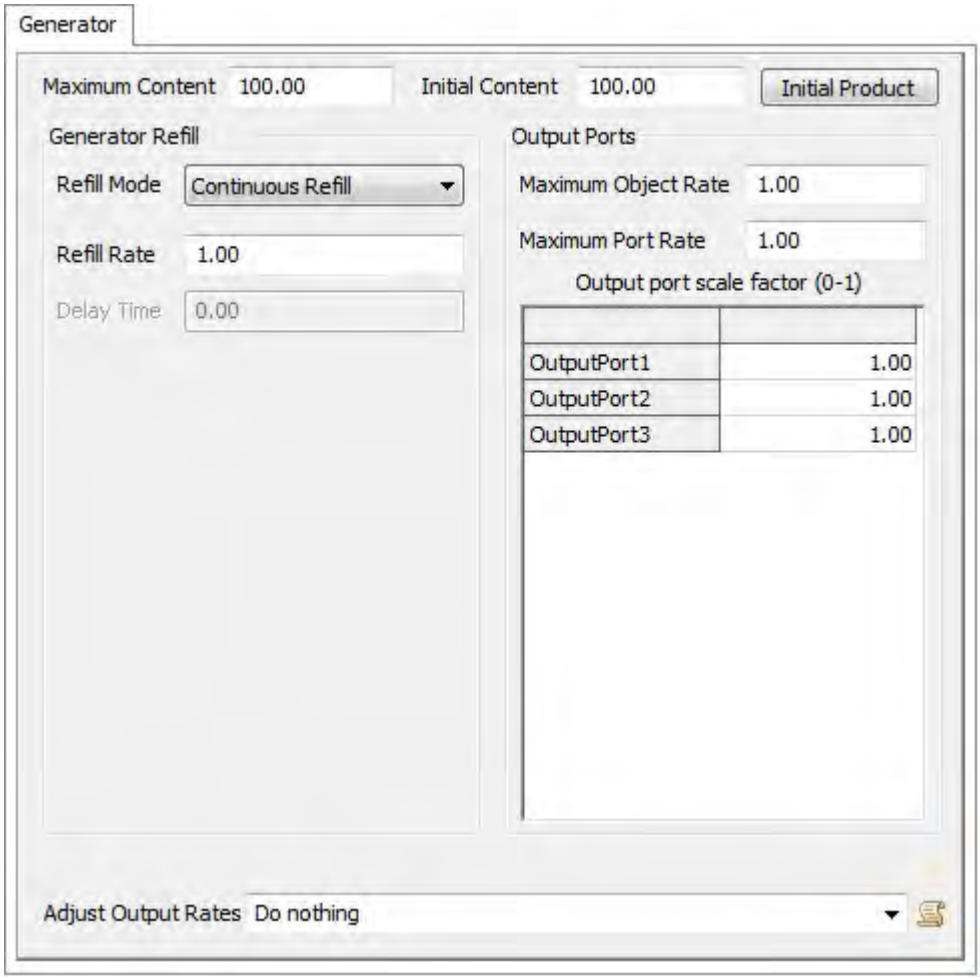
Fluid per Discrete Unit - This is the number of units of fluid material that are in a single discrete unit in the flowitem. For example: 5 gallons per can.

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.

This Page is Used By

FluidToItem



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 Window which allows the modeler 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 properties 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 properties affect how the object sends material to downstream objects.

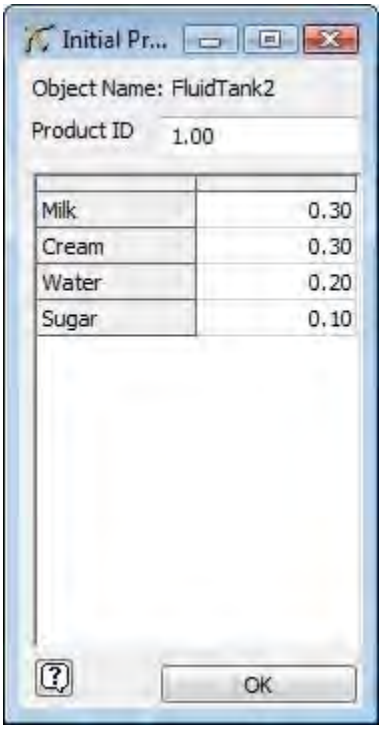
Maximum Object Rate - The maximum rate that material will leave this object through all output ports combined.

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 modeler to change the rate of individual output ports during a model run.

This Page is Used By

FluidGenerator



This window is used to define the initial Product ID and sub-component list for fluid material that is created by the object.

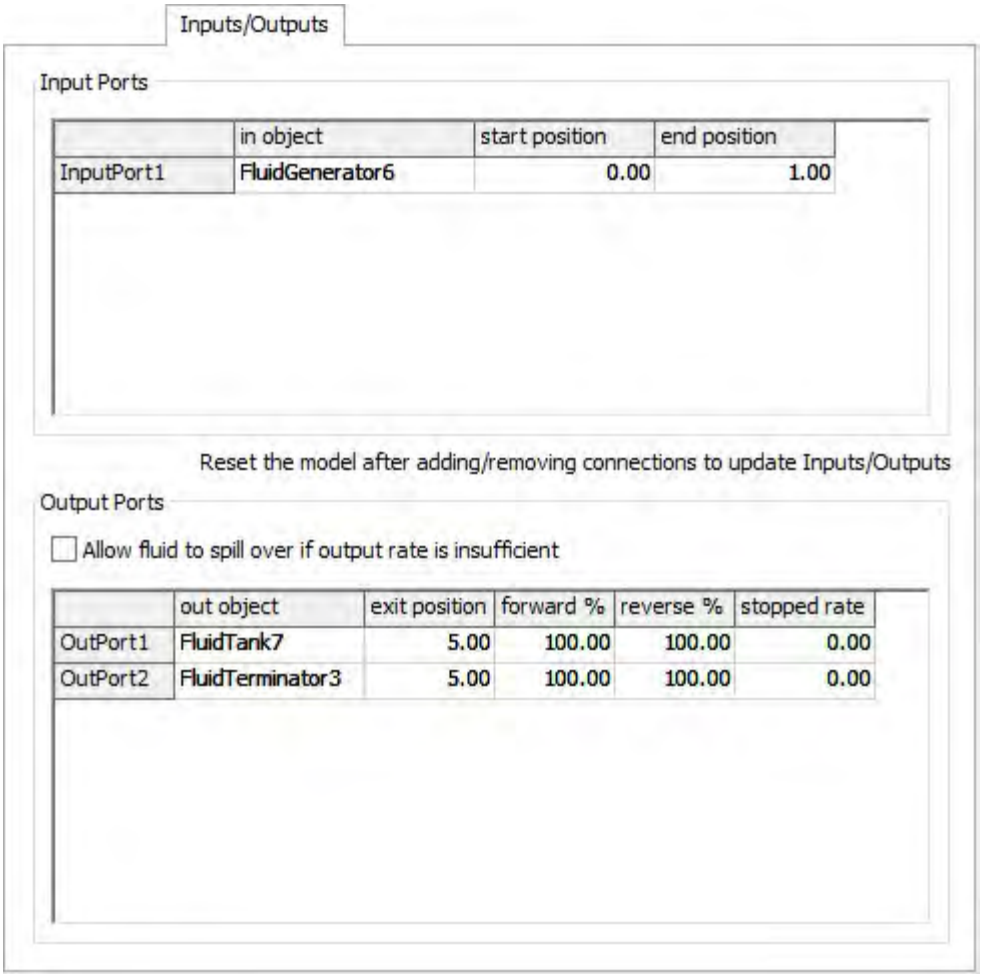
Object Name - The object that is creating this material.

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 Properties tab. 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



Input Ports Table

Displays all of the objects currently connected to an input port of the Fluid Conveyor.

In Object - Displays the name of the input object.

Start Position - The start point of the input. This value is in the model's length units and must be greater than or equal to 0 and less than the end position.

End Position - The end point of the input. This value is in the model's length units and must be less than or equal to the length of the Fluid Conveyor and greater than the start position.

Allow Spillage - If this is checked, fluid will be allowed to spill at outputs if the downstream object cannot take as much fluid as the Fluid Conveyor is sending. Any extra fluid left at the end of the conveyor will also spill. Total spillage is tracked each tick.

Output Ports Table

Displays all of the objects currently connected to an output port of the Fluid Conveyor.

Out Object - Displays the name of the output object.
Exit Position - The exit point of the output. This value is in the model's length units and must be greater than or equal to 0 and less than the end position.

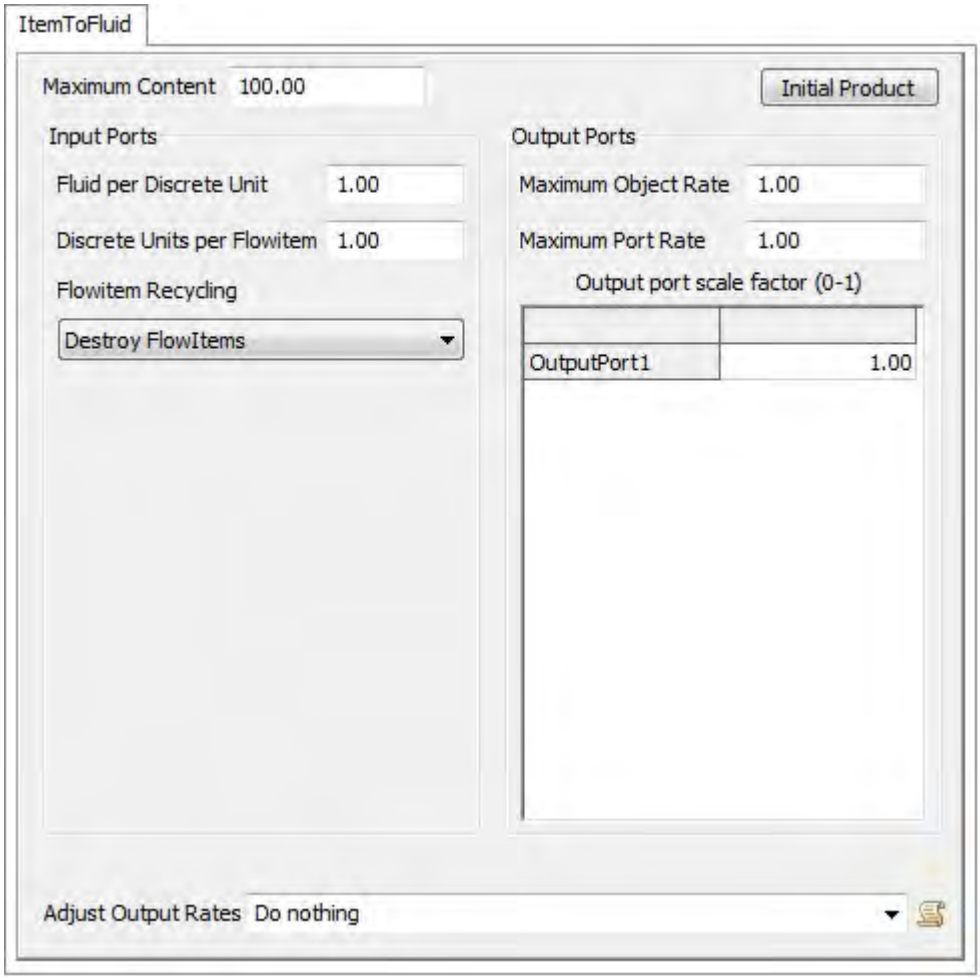
Forward % - This specifies the percentage of fluid that should exit at this output when the Fluid Conveyor is moving forward. The Fluid Conveyor will attempt to send the specified percentage of fluid through the output. If the downstream object cannot handle the total amount of fluid, it will pass the output, unless spillage is allowed, then the excess will spill onto the floor

Reverse % - This specifies the percentage of fluid that should exit at this output when the Fluid Conveyor is moving forward. The Fluid Conveyor will attempt to send the specified percentage of fluid through the output. If the downstream object cannot handle the total amount of fluid, it will pass the output, unless spillage is allowed, then the excess will spill onto the floor

Stopped Rate - The stopped rate specifies how much fluid should leave through the output when the Fluid Conveyor is stopped. For example, if the stopped rate is 0.1 and the material sitting on top of an output is 0.5, the output will take 0.1 fluid units per tick, until there is no more fluid available to take.

This Page is Used By

Fluid Conveyor



Maximum Content - The maximum amount of fluid material that this object can hold at any time.

Initial Product - This opens the Initial Product Window which that allows the modeler 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 modeler 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 properties affect how the object sends material to downstream objects.

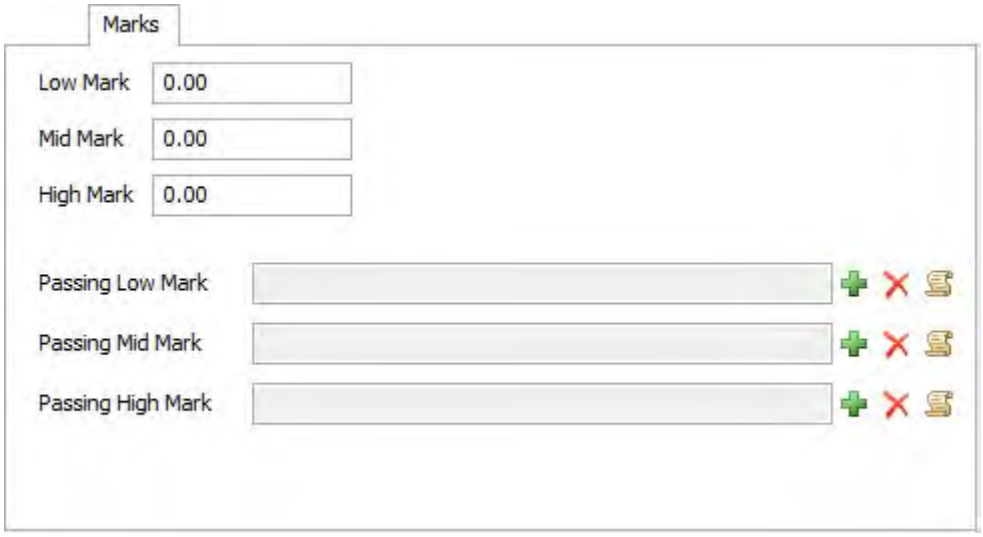
Maximum Object Rate - The maximum rate that material will leave this object through all output ports combined.

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 modeler to change the rate of individual output ports during a model run.

This Page is Used By

ItemToFluid



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.
Passing Low Mark - 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 modeler if the fluid level is rising through the mark or falling.

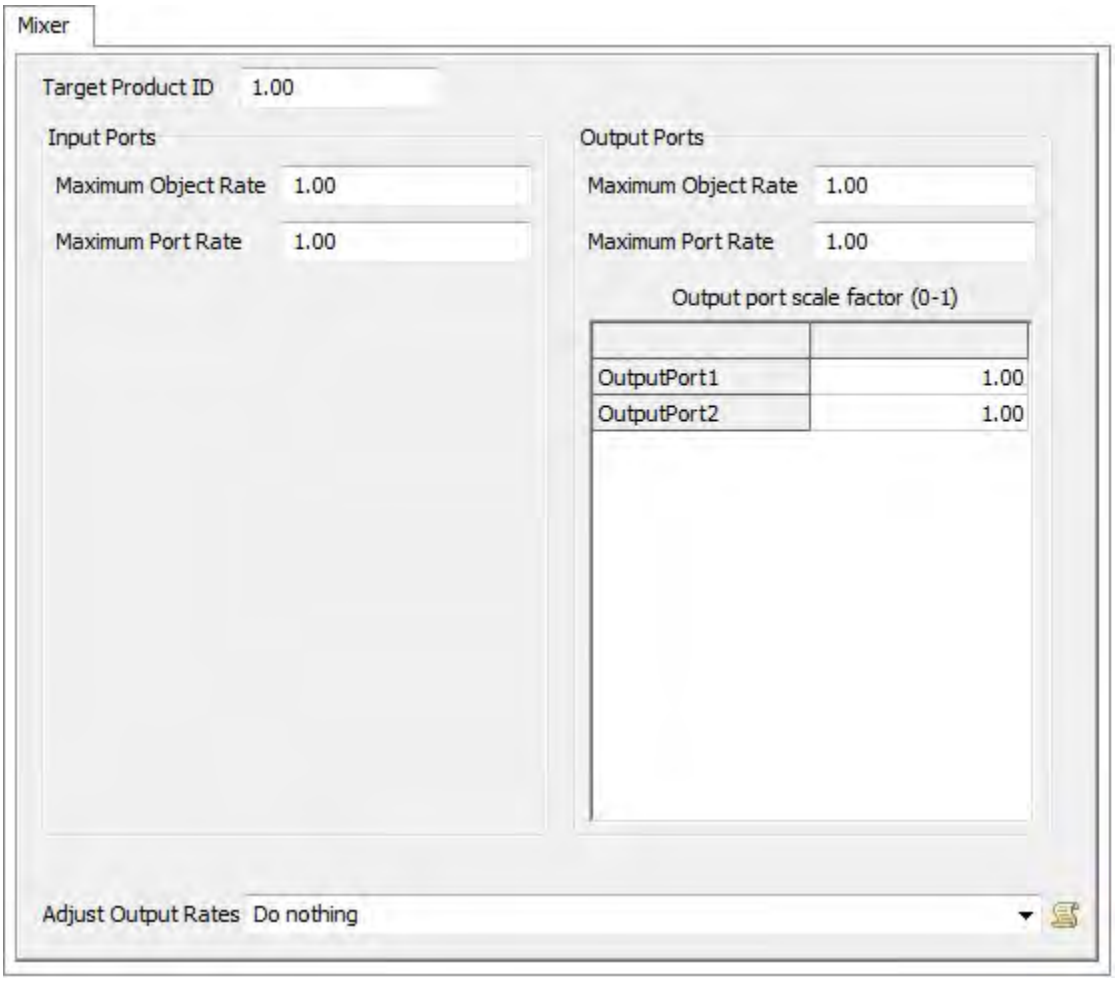
Passing Mid Mark - If the content passes the Mid Mark, this trigger fires. Its common uses include opening and closing ports or sending messages. There is an access variable that informs the modeler if the fluid level is rising through the mark or falling.

Passing High Mark - 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 modeler if the fluid level is rising through the mark or falling.

This Page is Used By

FluidTank

Mixer Page



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 properties 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 properties 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.

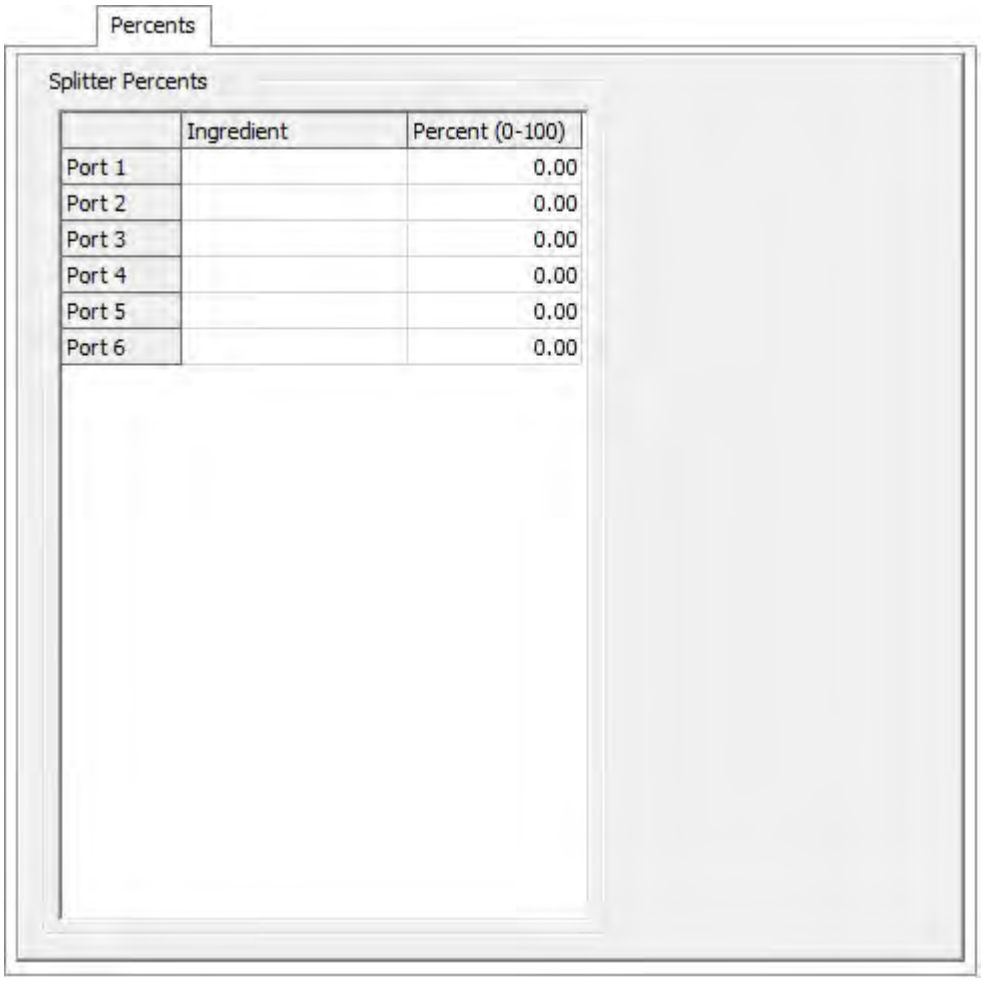
Maximum Port Rate - The maximum rate that material will leave this object through any one port.

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 modeler to change the rate of individual output ports during a model run.

This Page is Used By

FluidMixer

Percents Page



Splitter Percents

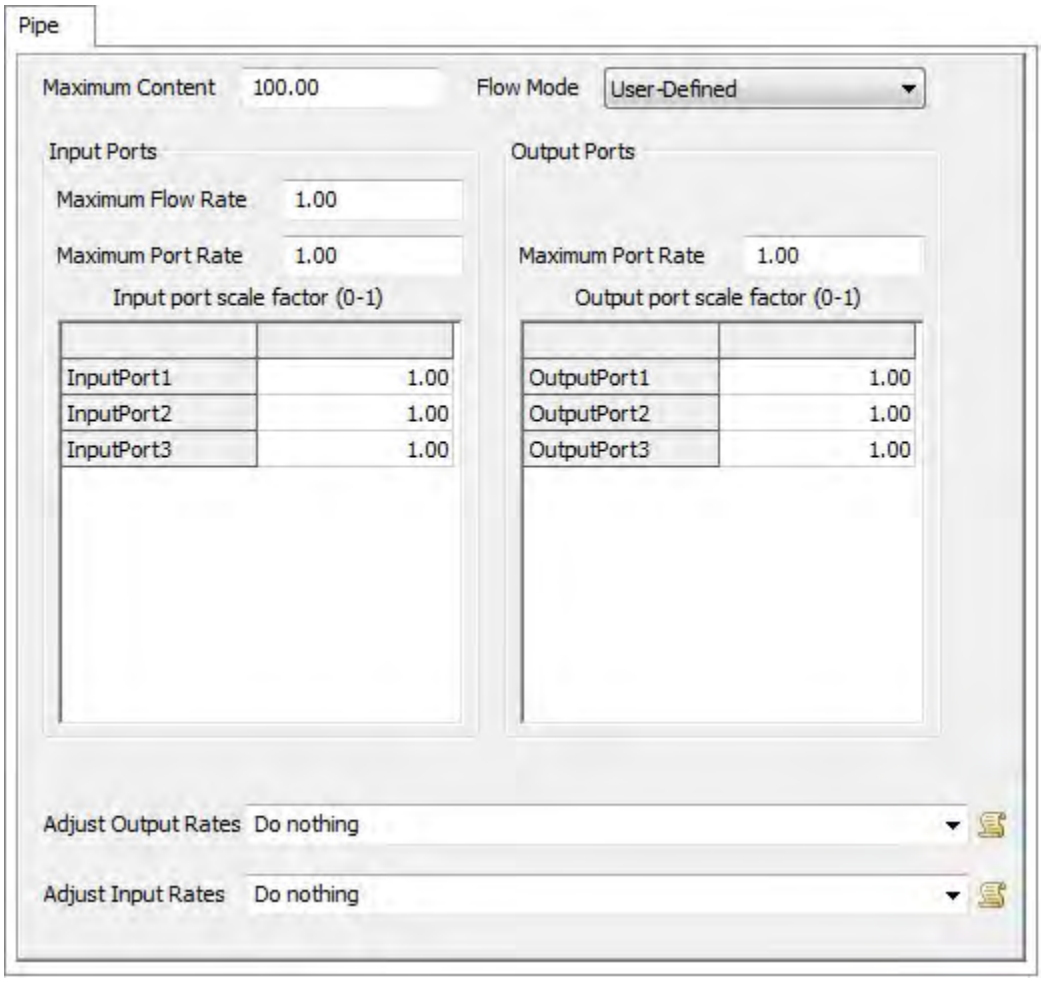
Each row of this table represents a single output port. The rows do not appear in the table unless the object is already connected to downstream objects when this window is opened. There are two columns that the modeler can change in the table:

Ingredient - This is a text description of the material going to the port the row represents. This is for the modeler'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



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 modeler 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 properties 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 modeler to change the rate of individual output ports during a model run.

Output Ports

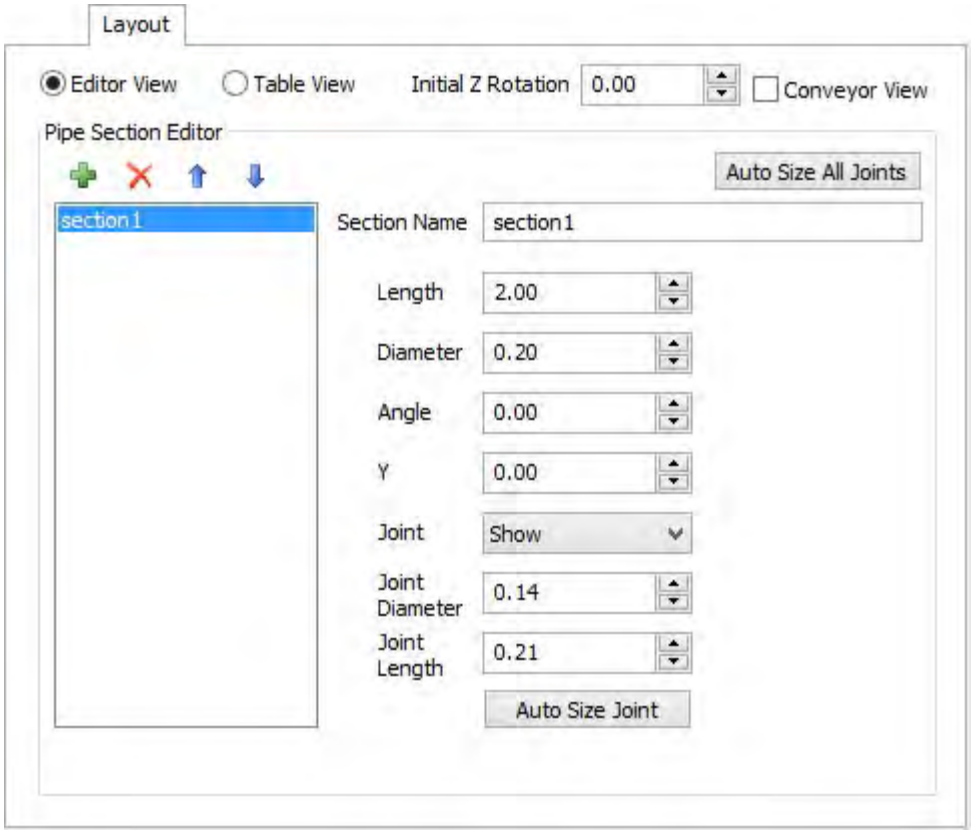
These properties affect how the object sends material to downstream objects.

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 modeler to change the rate of individual output ports during a model run.

This Page is Used By

FluidPipe



Editor View - Displays the Pipe Layout editor view as shown above.

Table View - Displays the Pipe Layout table view as shown below.

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.

Conveyor View - If this box is checked, the Pipe will be drawn as a simple conveyor. This does not change the functionality of the pipe. See the Fluid Conveyor object for conveyor functionality.

Pipe Section Editor

- Adds a new sensor to the table.

- Removes the selected sensor from the table, or if no sensor is selected, removes the last sensor in the table.

- Moves the selected sensor up in the list.

- Moves the selected sensor down in the list.

Section Name - The name of the pipe section. This is purely for the modeler's convenience and has no affect on the model.

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.

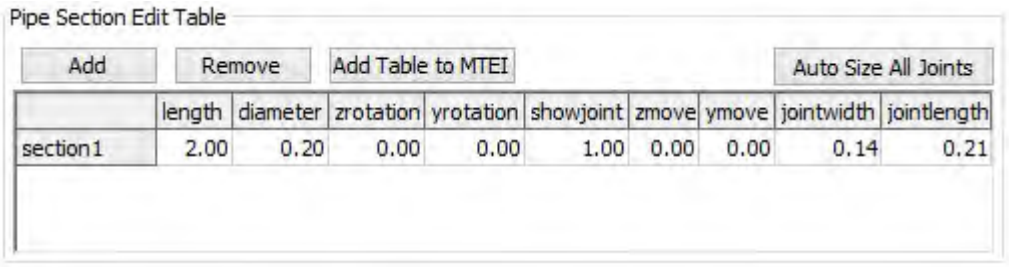
Show Joint - Specifies whether the pipe's joint will be drawn between the end of this section and the start of the next one. This value is ignored if the Pipe is being drawn as a Conveyor.

Join Diameter - Specifies the diameter of the joint.

Joint Length - Specifies the length of the joint.

Auto Size Joint - Automatically adjusts the joint size based on the diameter and rotation of the pipe sections.

Pipe Section Edit Table



Add - Adds a new pipe section by copying the selected section.

Remove - Removes the selected pipe section.

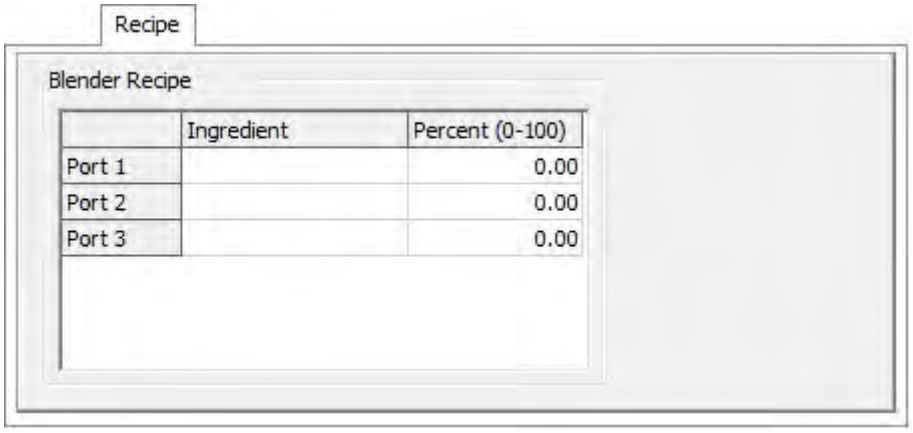
Add Table to MTEI - This buttons adds the table as a row in the Multiple Table Excel Import.

Section Table - Each row in the table represents a single section of the Pipe. The columns are described above.

Note: The zmove and ymove columns should not be changed. They are used by FlexSim.

This Page is Used By

FluidPipe



Blender Recipe

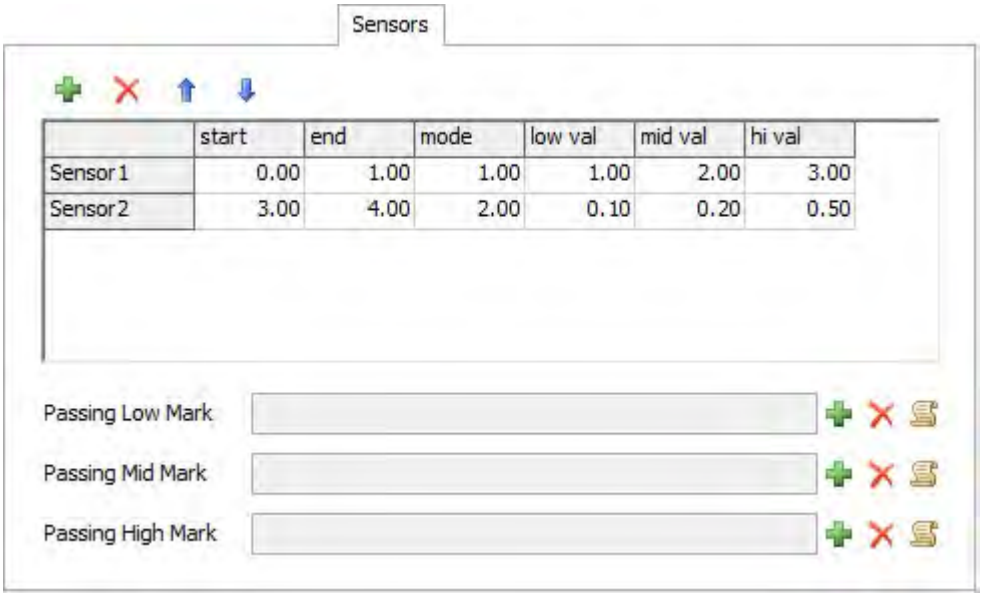
Each row of this table represents a single input port. The rows do not appear in the table unless the object is already connected to upstream objects when the Properties GUI is opened. There are two columns that the modeler can change in the table:

Ingredient - This is a text description of the ingredient coming from the port the row represents. This is for the modeler'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



Sensor Table

Displays all of the sensors for the Fluid Conveyor.

- Adds a new sensor to the table.

- Removes the selected sensor from the table, or if no sensor is selected, removes the last sensor in the table.

- Moves the selected sensor up in the list.

- Moves the selected sensor down in the list.

Start - The starting point of the sensor. This value is in the model's length units and must be greater than or equal to 0 and less than the end point.

End - The end point of the sensor. This value is in the model's length units and must be less than or equal to the total length of the conveyor and greater than the start point.

Mode - The sensor has two modes, Volume(1) and Peak Height(2). Volume will look at the total volume between the starting and ending point. Peak Height will look at the highest point between the starting and ending point.

Low, Mid, High Val - These values specify the low, mid, and high points for the volume or peak height that, when crossed, will trip the sensor. When the volume or peak height of material in a sensor range rises or falls through any of these three values, one of the three passing triggers will fire.

Sensor Triggers

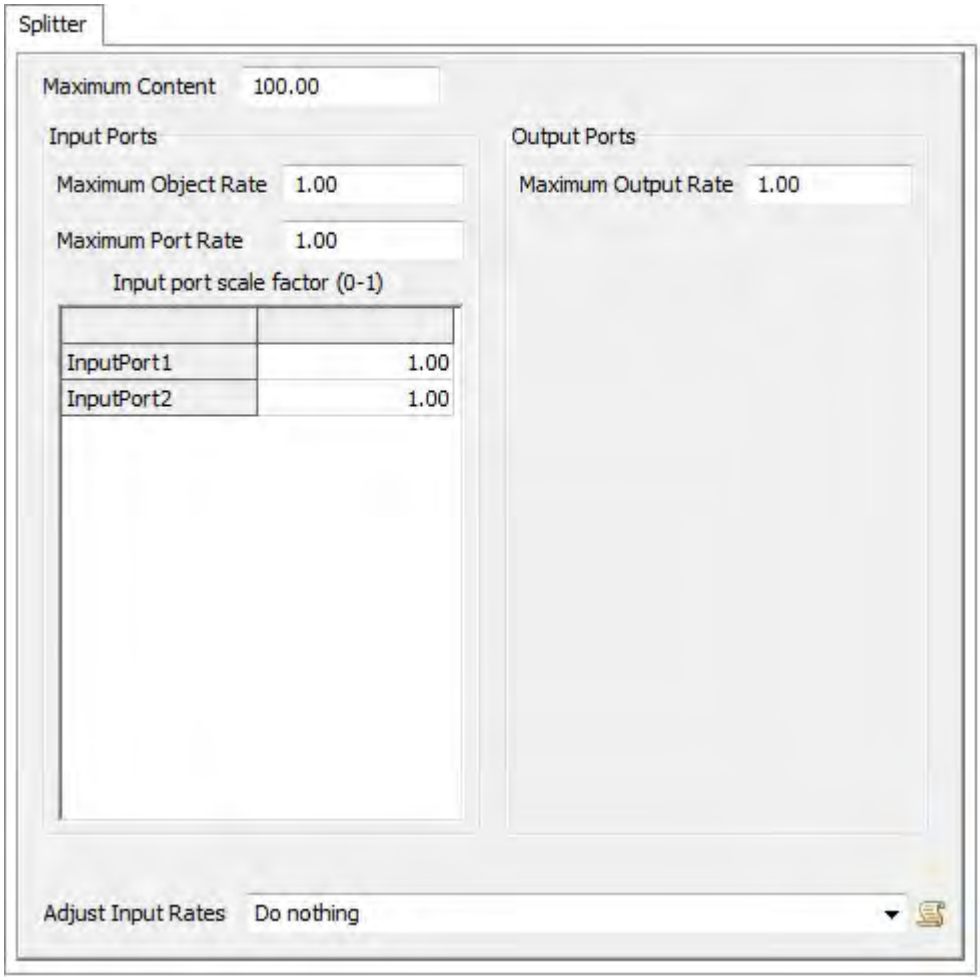
Passing Low Mark - This trigger is fired when fluid volume or peak height rises or palls through the defined Low Val.

Passing Mid Mark - This trigger is fired when fluid volume or peak height rises or palls through the defined Mid Val.

Passing High Mark - This trigger is fired when fluid volume or peak height rises or palls through the defined High Val.

This Page is Used By

Fluid Conveyor



Maximum Content - The maximum amount of fluid material that this object can hold at any time.

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 properties 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.

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 modeler to change the rate of individual input ports during a model run.

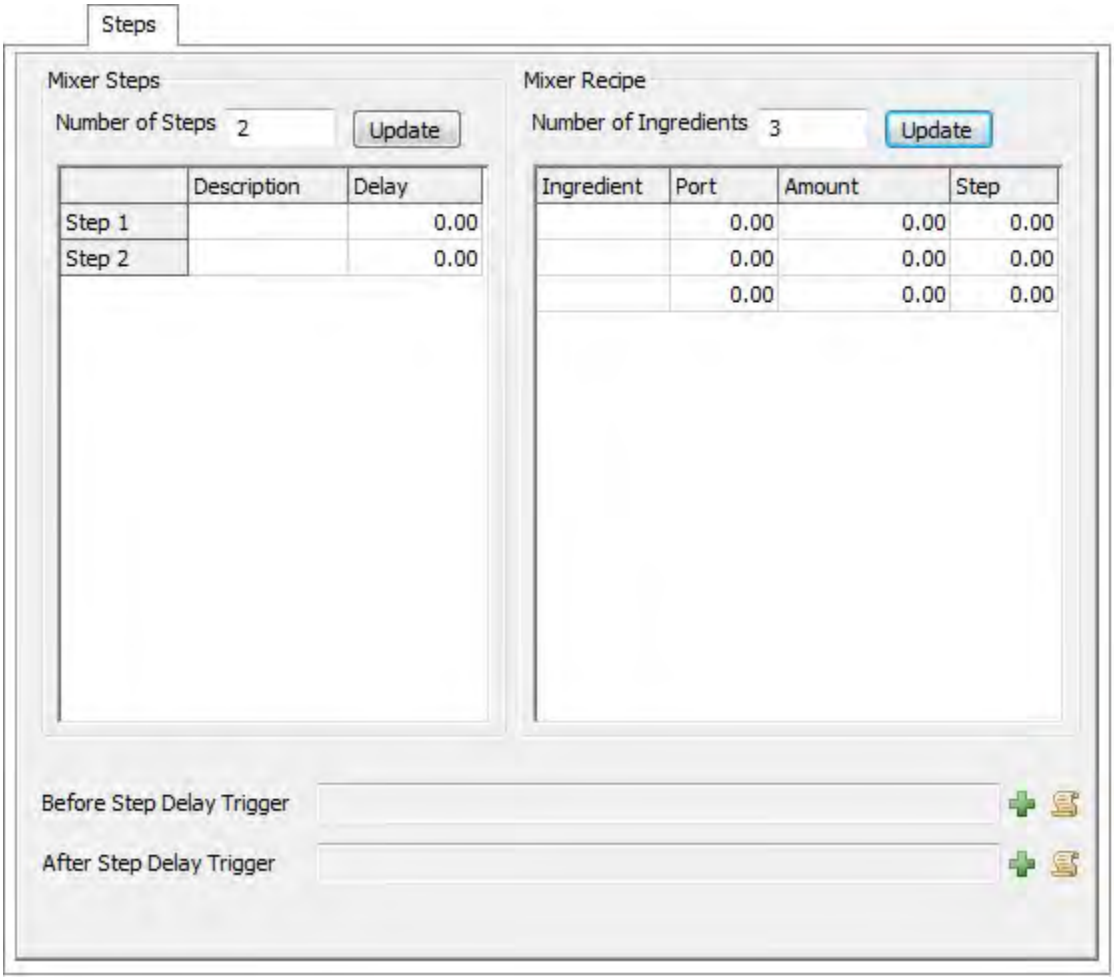
Output Ports

These properties 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



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 modeler 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 modeler 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 modeler.

The Step Table shows all of the steps that the Mixer must go through for each batch. Each step has two columns that the modeler 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 modeler.

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 modeler must fill out:

Ingredient - This is a text description of the ingredient that the row represents. It is only to help the modeler document their model. It does not affect the Mixer's behavior.

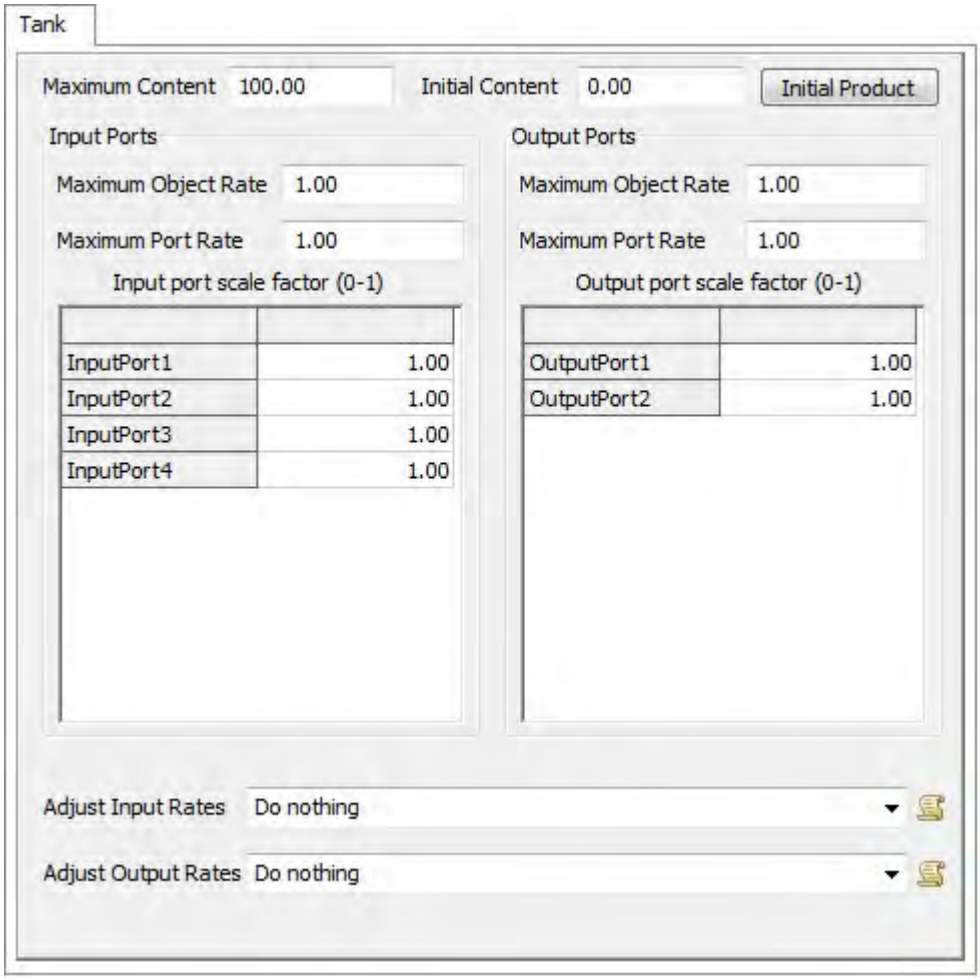
Port - This is the input port that the ingredient will be pulled from.

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



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 Window which that allows the modeler 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 properties 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.

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 modeler to change the rate of individual input ports during a model run.

Output Ports

These properties affect how the object sends material to downstream objects.

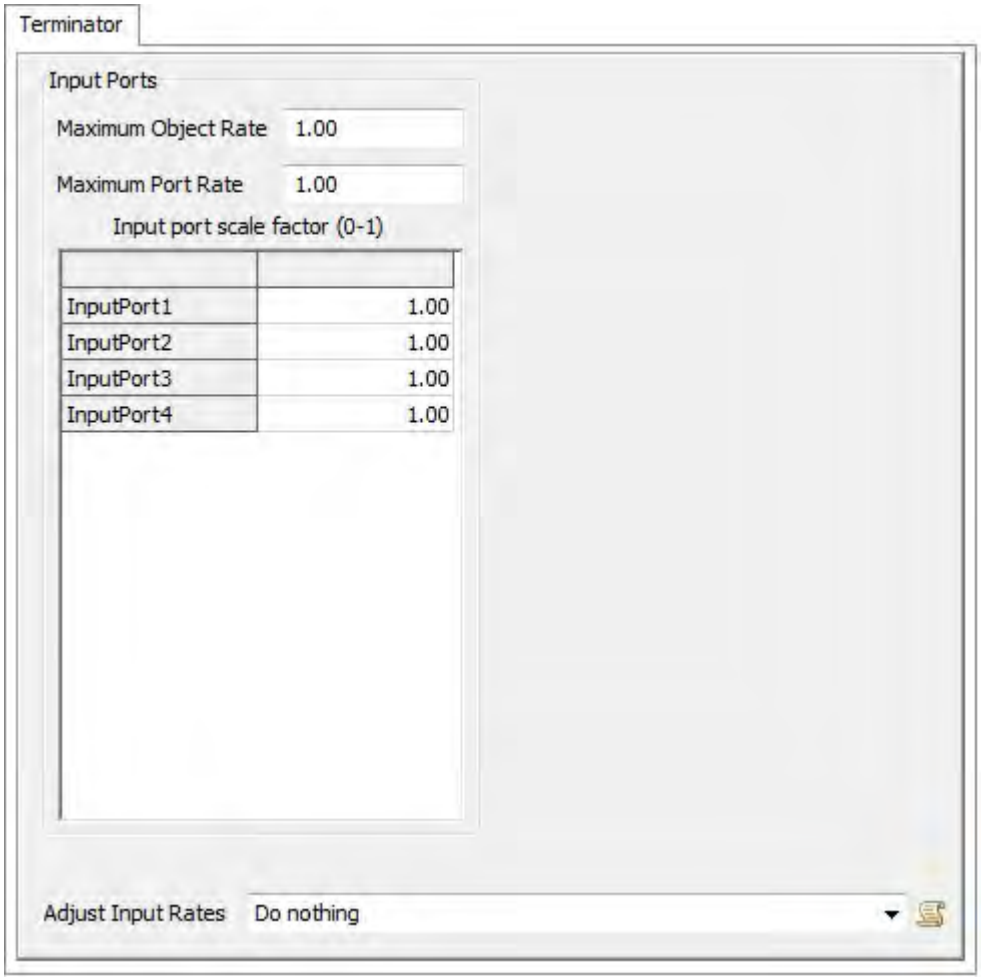
Maximum Object Rate - The maximum rate that material will leave this object through all output ports combined.

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 modeler to change the rate of individual output ports during a model run.

This Page is Used By

FluidTank



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 properties affect how the object receives material from upstream objects.

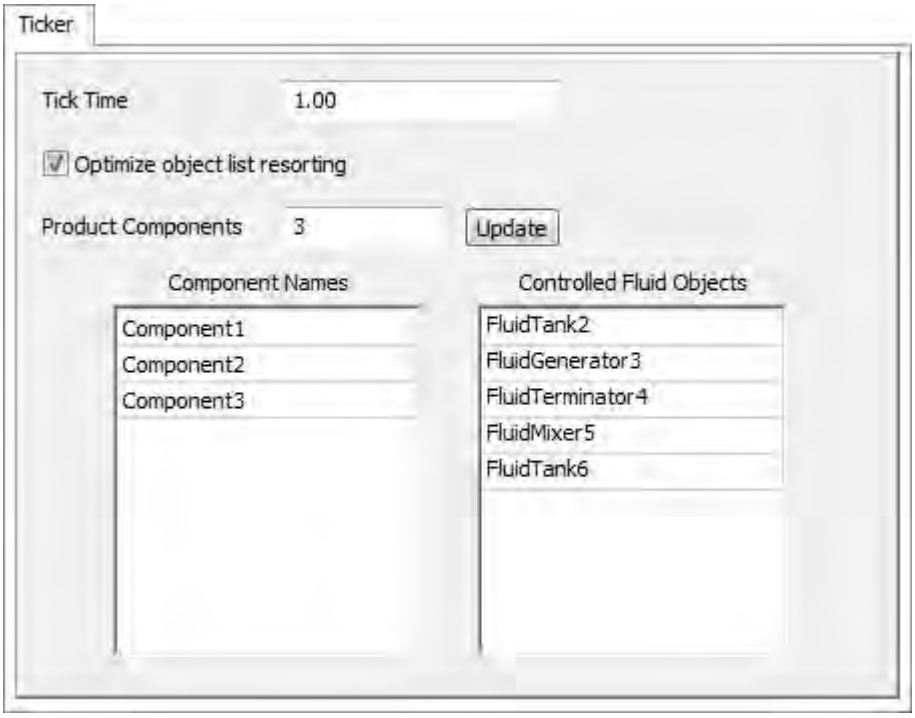
Maximum Object Rate - The maximum rate that material will enter this object through all output ports combined.

Maximum Port Rate - The maximum rate that material will enter this object through any one port.

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 modeler to change the rate of individual input ports during a model run.

This Page is Used By

FluidTerminator



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.

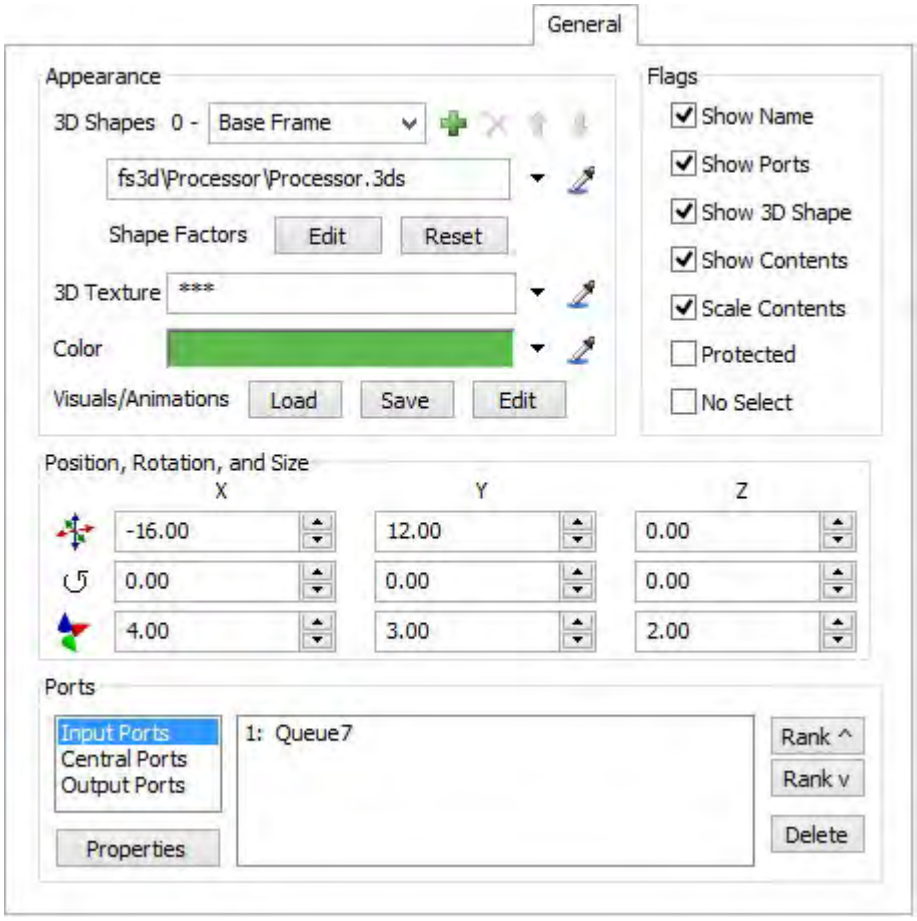
Update - Pressing this button updates the list of component names so that there are the number that the modeler specified.

Component Names - This table lists the names of the sub-components that are available to all of the fluid objects in the model.

Controlled Fluid Objects - This table lists the names of the fluid objects that are controlled by this Ticker.

This Page is Used By

Ticker



Appearance

3D Shapes - This option specifies the 3D shape(s) for the object. The drop down contains all of the shape frames for the object. The number (0 -) is the shape frame index for use in the setframe() command. For more information, see the Shape Frames page. You can specify any of the following types of 3D files:

- .wrl, .3ds, .dxf, .stl, .skp, .dae, .obj, .ac, .x, .ase, .ply, .ms3d, .cob, .md5mesh, .irr, .irrmesh, .ter, .lxo, .csm, .scn, .q3o, .q3s, .raw, .off, .mdl, .hmp, .scn, .xgl, .zgl, .lvo, .lvs, .blend

3D shapes can either be referenced through the or loaded 3D shapes will display in the down arrow menu. You can also select Browse... from this menu to load a new shape.

- Adds a Shape Frame to the object.

- Removes the current Shape Frame.

- Moves the current Shape Frame up in the list, changing its frame index.

- Moves the current Shape Frame down in the list, changing its frame index.

Shape Factors - Click on this button to display the object's shape factors popup. To learn more about shape factors, see the Shape Factors page.

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.

Visuals/Animations

These buttons allow you to save/load visual information for objects. You can also access an object's animations through the 3D views right click menu.

Load/Save - These buttons load/save all of the Appearance settings of an object, allowing you to save shape, texture, color, and animations in a .t file to load into other objects you want to look the same.

Edit - This button opens the Animation Creator, which allows you to create animations for FlexSim Objects.

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.

These flags can also be toggled through the Edit Selected Objects view.

Show Name - Toggles displaying the object's name in the 3D view.

Show Ports - Toggles displaying port connections in the 3D view.

Show 3D Shape - Toggles displaying the object's 3D shape.

Show Contents - Toggles displaying the object's contents.

Scale Contents - If checked, any objects within the content of this object will be scaled according to the size of this object.

Protected - If checked, this object will not allow the user to move, size, rotate, or delete the object.

No Select - If checked, this object will not be able to be clicked on the 3D view. To gain access to an object with the No Select flag checked, find it in the Tree Window.

Position, Rotation, and Size

Here you can set the position, rotation, and size of the object based on X, Y, and Z values.

 : Change the position of the object.

 : Change the rotation of the object.

 : Change the size of the object.

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. Once you have finished editing an object's connections, you will need to reset the model before running it again.

Rank ^ - This button will move the selected connection up in the list.

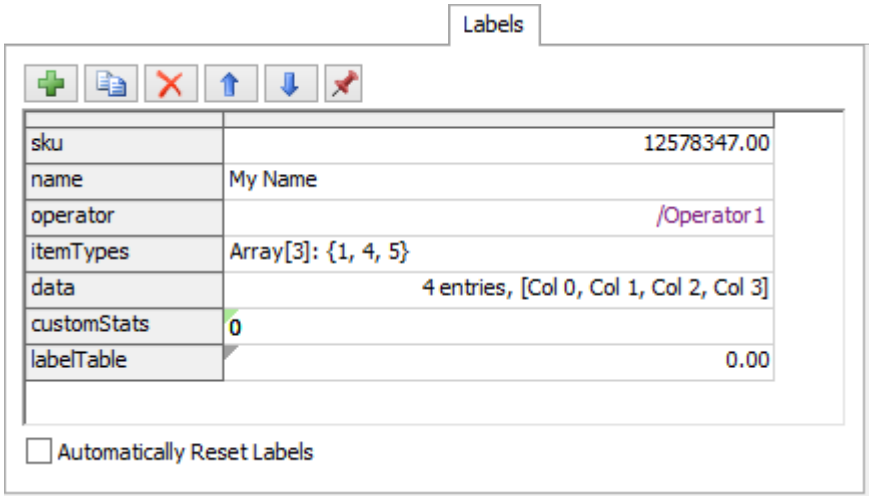
Rank v - This button will move the selected connection down in the list.

Delete - This button will delete the selected connection.

Properties - This button will open a new Properties window for the selected object.

Labels are custom values that you can specify on the object. For example, if you want to keep track of the number of flowitems of type 3 that have entered an object, you can use a label to keep a record of this value. You can access and set labels on an object in code by typing:

```
int value = item.labelName; item.labelName = 5; treenode
labelNode = item.labels["labelName"];
item.labels["labelName"].value = 5;
item.labels.assert("labelName", 0);
```



The main panel shows a list of the labels on this object.

- Adds a new label with number, string, pointer, array, FlexScript, bundle or tracked variable data.

- Duplicates the selected label(s).

- Deletes the selected label(s).

- Moves the selected label(s) up or down in the list.

- Pins the selected label(s) (or all labels if there is no selection) to a Dashboard as either a table of values, bar chart or line graph. Note: Pinning labels with string data will display 0 for its value.

Automatically Reset Labels - If this option is checked, then the object will automatically reset its labels back to their initial values on reset. When you apply the window, the values shown will be saved as the reset values. The reset values will also be automatically set when directly editing the labels on this tab while the model is reset.

Labels can also be edited through the Quick Properties.

Label Data Types

Labels can store a variety of different types of data. These include:

- Numbers: Integer or floating point.
- Strings: Text like names or sentences.
- Pointers: References to other nodes or objects. The value displayed is the path to the node/object and will be purple.
- Arrays: Arrays can have any number of elements. Each element can store a number, string, pointer or array.
- FlexScript: FlexScript toggled nodes allow you to write code that can that be evaluated using the evaluate() method. For example, current.labels["myCode"].evaluate()
- Bundles: Data table that stores numbers and strings.
- Tracked Variables: Tracked variables are often used for statistic collection. For more information see Tracked Variables.

Additionally, a label can also store a table of data. This can be done regardless of the data stored on the label node. To create a label table, right click on the desired label and select Explore As Table. This will open a new table window where you can add rows and columns to the node. If a label has a table, a small gray triangle will be drawn in the upper right corner of the cell. See the Global Table page for more information on editing tables.

Tracked Variable labels will display with a small green triangle in the upper right corner.

If a label has array or bundle data, double-clicking on the label will open a table view that can be edited.

Each object may have a different set of triggers. For more information about how triggers and pick lists work, refer to the Picklists page.

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 tab page. See OnArrival trigger picklist.

OnBreakDown: This function is called on an object when its MTBF expires. See Breakdown/Repair Trigger picklist.

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 picklist.

OnConveyEnd: This function is called on a Conveyor when a flowitem reaches its end. See Process Finish Trigger picklist.

OnCreation: This function is called on a Source when it creates a new flowitem. See Creation Trigger picklist.

OnEndCollecting: This function is called on a Queue when it has reached its batch limit. See Process Finish Trigger picklist.

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 picklist.

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 picklist.

OnEntry: This function is called on an object when a flowitem is moved into it. See Entry/Exit Trigger picklist.

OnExit: This function is called on an object when a flowitem is moved out of it. See Entry/Exit Trigger picklist.

OnFallThroughHighMark: This function is called on a reservoir when the content falls below the designated high mark. See Rise/Fall Through Mark Triggers picklist.

OnFallThroughLowMark: This function is called on a reservoir when the content falls below the designated low mark. See Rise/Fall Through Mark Triggers picklist.

OnFallThroughMiddleMark: This function is called on a reservoir when the content falls below the designated middle mark. See Rise/Fall Through Mark Triggers picklist.

OnFull: This function is called on a fluid object when its content reaches its maximum content. See Empty/Full Trigger picklist.

OnLoad: This function is called on an Operator or Transport when it completes loading a flowitem. See Load/Unload Trigger picklist.

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 picklist.

OnProcessFinish: This function is called on an object when its process time has expired. See Process Finish Trigger picklist.

OnReceiveTaskSequence: This function is called when the TaskExecuter receives a new TaskSequence. See OnReceiveTaskSequence Triggers picklist.

OnRepair: This function is called on an object when its MTTR expires. See Breakdown/Repair Trigger picklist.

OnResourceAvailable: This function is called when a downstream resource of a Dispatcher becomes available. See OnResourceAvailable Trigger picklist.

OnRiseThroughHighMark: This function is called on a reservoir when the content rises above the designated high mark. See Rise/Fall Through Mark Triggers picklist.

OnRiseThroughLowMark: This function is called on a reservoir when the content rises above the designated high low. See Rise/Fall Through Mark Triggers picklist.

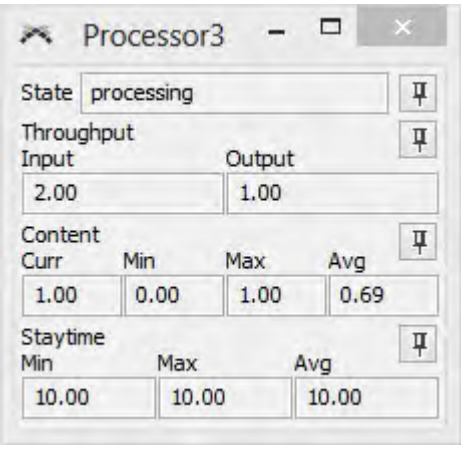
OnRiseThroughMiddleMark: This function is called on a reservoir when the content rises above the designated middle mark. See Rise/Fall Through Mark Triggers picklist.

OnSetupFinish: This function is called on an object when its setup time has expired. See Process Finish Trigger picklist.

OnStateChange This function is called when an object's state is changed with the setstate() command. See OnStateChange Trigger picklist

OnUnload: This function is called on an Operator or Transport when it completes unloading a flowitem. See Load/Unload Trigger picklist.

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 tab page.



The Statistics window is not part of an object's properties window, rather it is a separate window that is accessible from the Quick Properties window under the Statistics Tab. This Statistics tab only displays when an object is selected in the 3D view and the model time is greater than 0.

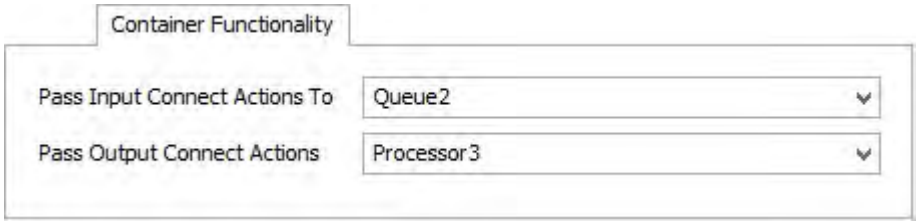
The statistics shown in this window include the object's current state, throughput, content and staytime. If the object is a TaskExecuter object, it will also display the total distance travelled.

- Pins a statistic to a dashboard. This button creates a new dashboard statistic object and pins it to the selected Dashboard.

This page will appear in a flowitem that is a Container type. For more information see the Flowitem Bin. Pack Contents - This drop down will show a list of all available packing methods. Setting a packing method will cause the packing method code to be fired any time a flowitem is placed inside this flowitem.

This Page is Used By

Flowitems



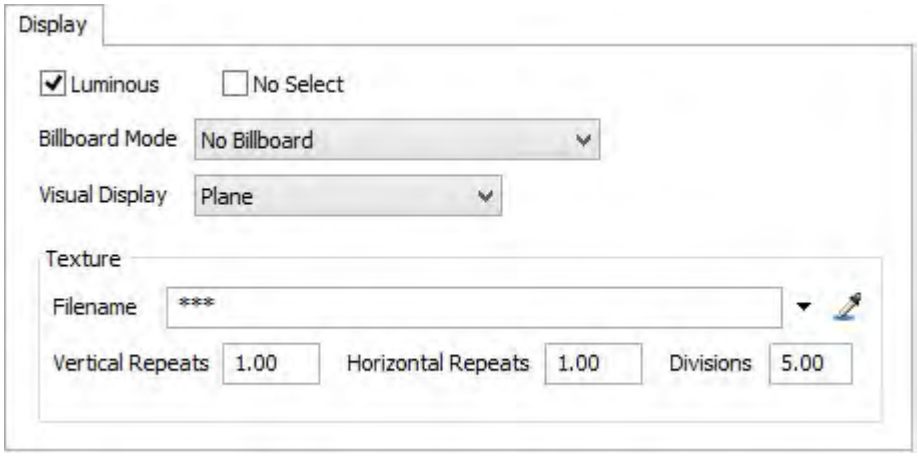
This page will appear in the VisualTool's Properties window if the VisualTool has objects inside of it.

Pass Input Connect Actions To - Set this pull-down list to the object that should be connected to the input port of the Visual Tool. Any objects you subsequently connect to the Visual Tool will automatically connect from the Visual Tool to this object. For more information, refer to the Using the VisualTool as a Container section.

Pass Output Connect Actions - Set this pull-down list to the object that should be connected to the output port of the Visual Tool. Subsequently, when you connect the Visual Tool to another object, it will automatically connect from this object to the Visual Tool. For more information, refer to the Using the VisualTool as a Container section.

This Page is Used By

VisualTool



Luminous - If this box is checked, the object will appear to give off its own light and will have gradual shadows caused by the defined light sources.

No select - If this box is checked, the object cannot be selected using the mouse in the Orthographic / Perspective views.

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.

Billboard Mode - This option is usually only used when "Visual Display" is set to Plane. Options include:

- No Billboard - The visual tool will display based on its set rotation.
- Vertical Billboard - This will cause the plane to stand vertically in the model and always face toward the camera along the same viewing plane.
- All Around Billboard - This will cause the plane always present its face to you no matter what the viewing angle is.
- Screen Locked Billboard - Locks the Visual Tool in front of the view.

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. The various options in this drop-down menu and the subsequent changes to this tab page are described in more detail in the VisualTool section.

Texture

These properties are used to define how a texture is drawn on the 3D object.

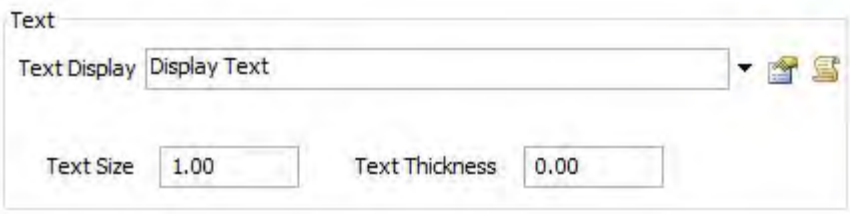
Filename - This file is the bitmap image that will be textured on the object.

Vertical Repeats - This number defines how many times the texture image will be repeated vertically across the image.

Horizontal Repeats - This number defines how many times the texture image will be repeated horizontally across the image.

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).

Text



These options are available when Text is selected in Visual Display.

Text Display - This picklist defines what text will be displayed. You may choose from the picklist options, or click the to edit the code directly.

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.

Model

These options are available when Imported Shape is selected in Visual Display.

Filename - Specifies which 3D shape to display. You can use a 3D shape from the model, or import a new shape by clicking the down arrow and selecting Browse...

Presentation Slide

These options are available when Presentation Slide is selected in Visual Display.

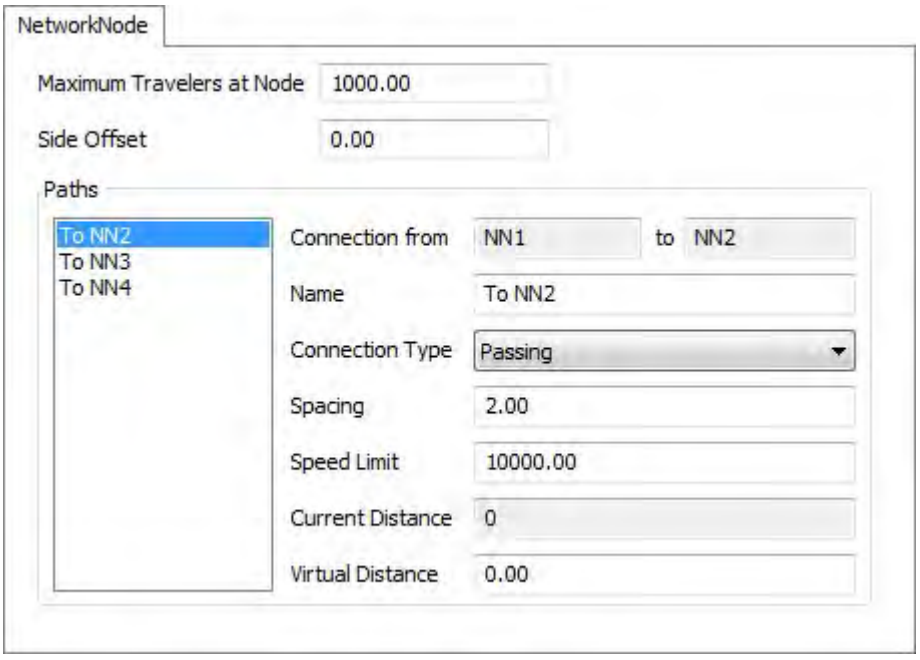
Choose Text - Presentation Slides may have multiple text displays. Click the to add a new text object. The options below will enable to specify the text's properties.

Text Display - This picklist defines what text will be displayed. You may choose from the picklist options, or click the to edit the code directly.

Size - The size of the displayed text.

Color - The color of the displayed text.

This Page is Used By



For more detailed information on these fields, see theNetworkNode documentation.

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.

Paths

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 Properties 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.

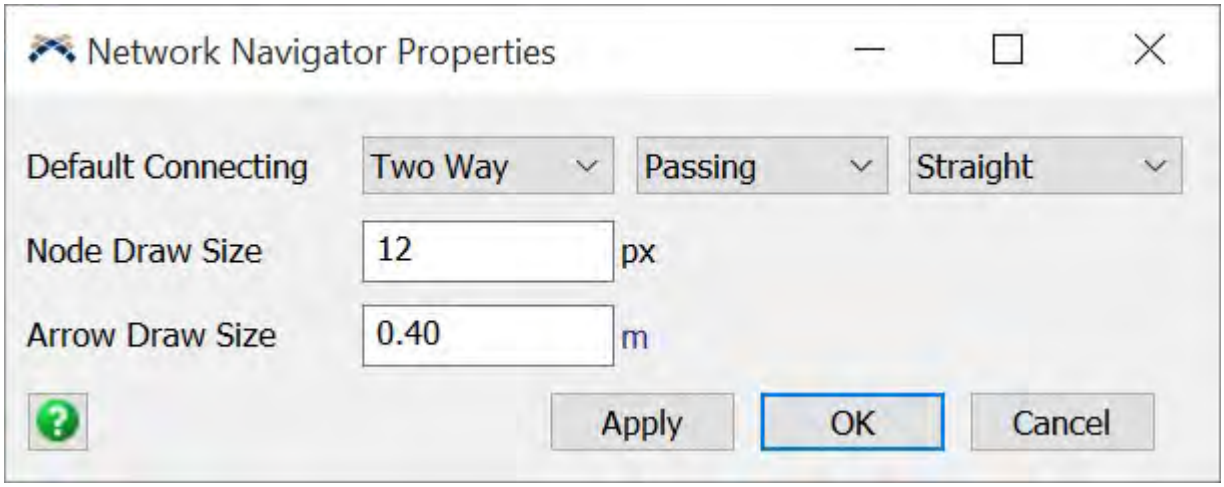
Speed Limit - This number determines the maximum speed that a traveler can travel along this connection.

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.

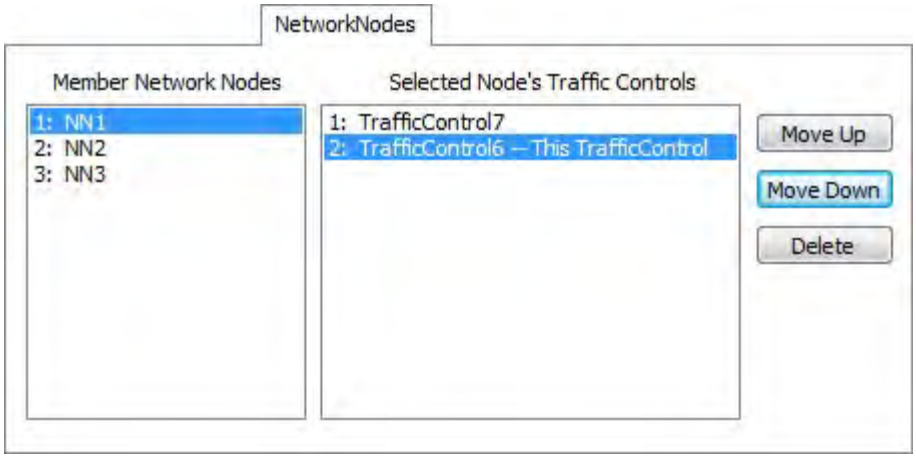
This Page is Used By

NetworkNode



To access this window, right-click on a network node in the model and select Network Properties from the context menu.

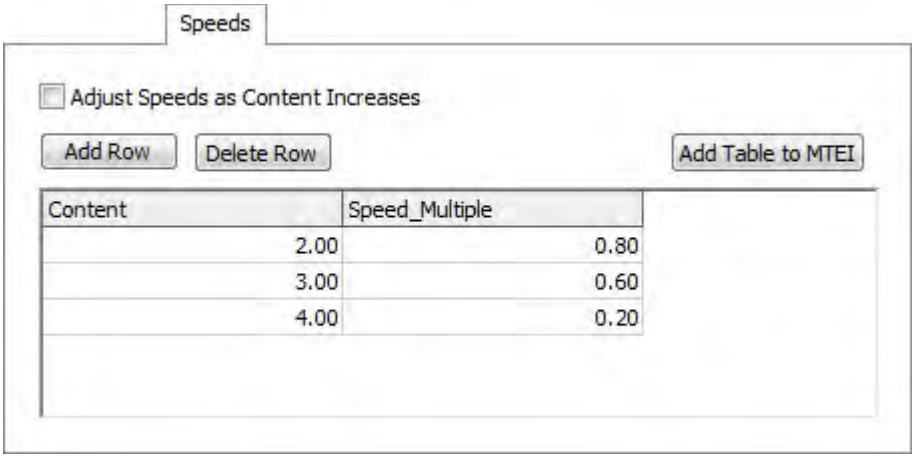
- Default Connecting Defines default path settings when creating network paths. You can always change this after a path has been created. This simply defines the initial settings for the created path when you connect two network nodes.
- Node Draw Size Defines the size, in pixels, at which network nodes will be drawn.
- Arrow Draw Size Defines the size at which path arrows will be drawn.



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



Adjust Speeds as Content Increase - If this option is checked, then the table will be used to adjust speeds of travelers within the TrafficControl's area.

Add Row - Click this button to duplicate the selected row. If no row is selected, it will add a row to the end of the table

Delete Row - Click this button to delete the selected row from the table. If no row is selected, it will delete the last row of the table.

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.

This Page is Used By

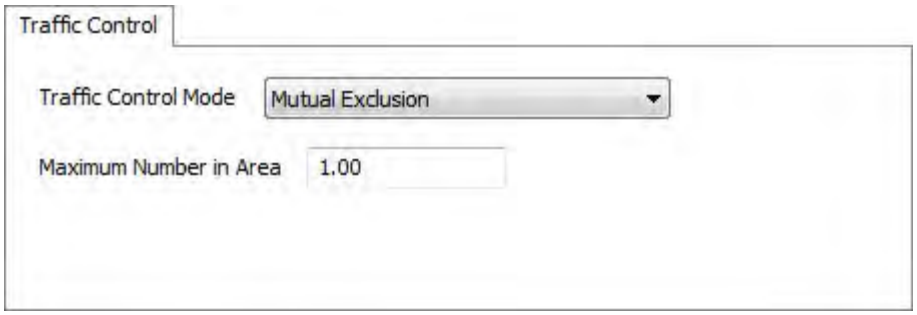
TrafficControl

For more detailed information on the TrafficControl object, refer the TrafficControl object.

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

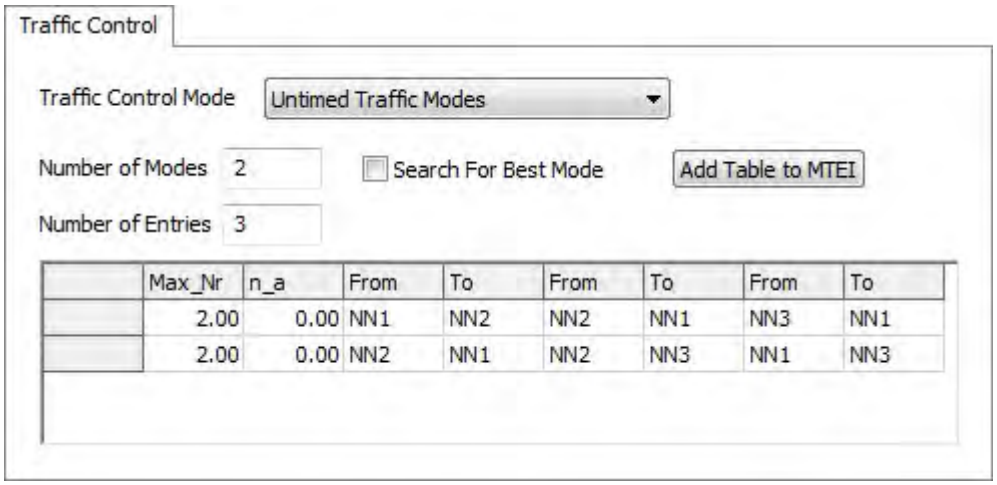
Mutual exclusion is used to only allow a given number of travelers into the Traffic Control's area, regardless of which paths they are on. Here you simply specify the maximum number value.



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.



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

TrafficControl

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.

Fluid Objects

There are 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 in the Fluid Objects Overview.

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

Combiner Separator
MultiProcessor
Rack
BasicFR

TaskExecuters

Dispatcher
TaskExecuter
Operator
Transporter
Elevator
Robot
Crane
ASRSvehicle BasicTE

Travel Networks

NetworkNode
TrafficControl

Visual

VisualTool

Fluid Objects

Ticker
FluidTank
FluidGenerator
FluidTerminator
FluidMixer
FluidBlender
FluidSplitter
FluidPipe
FluidProcessor
ItemToFluid
FluidToItem

The FixedResource is a superclass of the Source, Queue, Processor, Sink, Combiner, Separator, Rack, and MultiProcessor 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.

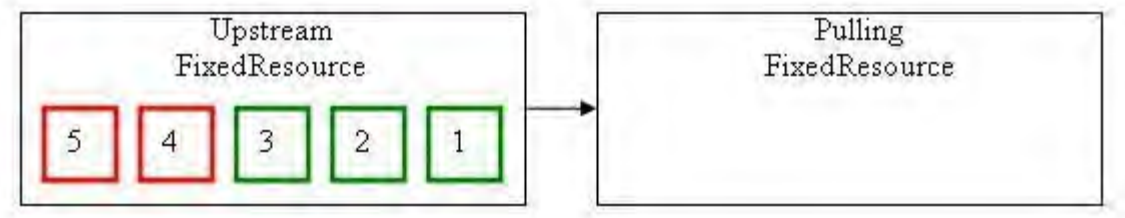
Details

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 Properties window. The following diagram shows the steps that a FixedResource object goes through for each flowitem that it receives and subsequently releases.

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 Pull Strategy 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).

After a flowitem is selected, received, and released the Pulling FixedResource is again ready to recieve one of the Upstream FixedResource's flowitems. The same process is repeated such that all the flowitems are reevaluated, including the flowitems that returned a no (0) before.

Note on the Send To Port of upstream objects that an object is pulling from: In FlexSim versions before 6: if an object was configured to pull from upstream objects, the Send To Port of those objects was nullified. In version 6, the Send To Port of upstream objects is evaluated in conjunction with the pull strategy. Both the send-to-port and the pull strategy must be true for the item to be pulled.

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.

3. Release the flowitem and determine which output ports to open

When the flowitem is released, the FixedResource calls the Send To Port 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 reevaluate its Send To Port, then each time a downstream FixedResource becomes available to receive a new flowitem, the upstream FixedResource will reevaluate the Send To Port 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 Send To Port 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 Send To Port 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 TaskExecuter or Dispatcher. If a valid reference is returned, then a default task sequence will automatically be created, and a TaskExecuter 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 creates a task sequence for a TaskExecuter 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. Send To Port 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 nroftransportsin and nroftransportsout. Once the object has called the Request Transport From field, it increments its own nroftransportsout variable, and notifies the downstream object, which subsequently increments its own nroftransportsin 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 TaskExecuter moves the flowitem, it notifies the FixedResource of the operation being done, so that the FixedResource can appropriately decrement its nroftransportsin/nroftransportsout variable. In summary, the nroftransportsout of the upstream object and the nroftransportsin 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 nroftransportsout variable is then decremented just before the TaskExecuter finishes the frload tasks and moves the flowitem out of the upstream object. The nroftransportsin variable is decremented just before the TaskExecuter finishes an frunload task and moves the flowitem into the downstream object.

The integrity of the nroftransportsin and nroftransportsout 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 nroftransportsin 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 nroftransportsin/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.

Alternatives to Using Port Connections

You can control the flow of items between fixed resources using a few different methods:

- Use standard port connections and standard send-to/pull logic as explained in the previous sections.
- Push and pull from a list, instead of connecting output/input ports.
- Initiate transport manually, bypassing port hand-shaking logic.

The following sections describe these scenarios.

Using Standard Port Connections

It's important to first explain what the standard port connection mechanism is doing.

1. Ports define the objects a FixedResource can send to and pull from

   When you connect FixedResource input and output ports together, you are defining the search patterns by which upstream FixedResources can find downstream FixedResources to send their items to, and/or the search patterns by which downstream FixedResources can find items to pull from upstream FixedResources. For example, when a downstream FixedResource is ready to receive its next item, it is limited in its search for receivable items to the objects that are connected to its input ports. Also it will search them in the order that the ports are connected (input port 1 first, then port 2, and so forth).

2. Ports contain open/closed state to determine availability

   Input and output ports can be open or closed. FixedResources use this open/closed state in deciding where items can go. When an item is released to port 1, for example, the object's output port 1 is made open (green). If the corresponding input port of the downstream object is also open (green), then that is a signal that the downstream object is ready to receive an item, so the item will be sent to that downstream object.

3. Port rankings can be important

   If you are use routing rules based on defined values, i.e. labels, port rankings can be important. For example, if you're sending by item type directly, then items with type 1 will be sent to port 1. This means you must make sure your port rankings are ordered correctly.

Situations That Work Well Using Port Connections

- One-to-One Routing - Port connections work well when you're just sending an item from one station to a single next station. Just connect the the upstream station to the downstream station with an 'A' connection.
- One-to-Many Routing - Using port connections works well if you're sending an item from one station to one of many stations using a basic routing rule like round-robin, first available, random, or by some defined value like a label. Just connect the station to the set of downstream stations, then define the routing rule through the Send To Port field in the object's Flow tab.
- Many-to-One Routing - Using port connections can work well if you're send an item from one of many stations to a single downstream station using a basic pulling rule like round-robin, first available, longest waiting, random, or by some defined value like item type. Just connect each upstream station to the downstream station, then define the pulling rule through the Pull Strategy field in the object's Flow tab.

Pushing and Pulling from a List

Some scenarios can make the management of port connections difficult, such as many-to-many routing, or when prioritization is complex. In these cases, using Lists to define the search pattern for sending and receiving items can be easier than using ports. The connectionless routing example shows how to do this.

In this example you use pick list options that push and pull from an item list. These pick list options override the default port connection mechanism. Here the upstream objects' Send To Port logic pushes items onto the list, while downstream objects' Pull Strategy pulls items from the list. The search pattern is thus defined by

the items that are on the list at any given time, i.e. the items that have been released by upstream objects. Since the searching relies solely on the list, connecting ports is no longer necessary, so you can leave objects that use this mechanism unconnected.

Initiating Transport Manually

In some cases, you may want to initiate transport of items before the downstream object(s) are ready to receive them. As explained above, when using standard send-to/pull logic, transport defined by the Use Transport checkbox will only be initiated when the downstream object is ready to receive the item. To override this functionality you would need to bypass the send-to/pull mechanism altogether. To do this, leave the upstream and downstream object input and output ports unconnected. Then use triggers on the upstream object(s), such as OnEntry or OnProcessFinish, to manually create a task sequence to transport the items to their destinations. If you are using the Process Flow module, you could use an Event-Triggered Source to start a custom process when the desired event happens. You can use center ports for referencing between objects.

BasicFR

The BasicFR object is a FixedResource that is designed to be customized into a user library object. 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 modeller.

Commands

receiveitem() releaseitem()

Extra Parameters for the Entry/Exit Triggers

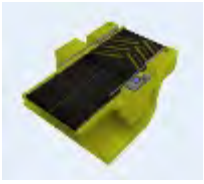
The BasicFR passes 2 extra parameters into its entry and exit triggers:

- parval(3) - the current value of its nroftransportsin variable.
- parval(4) - the current value of its nroftransportsout variable.

Properties pages

BasicFR Advanced
Flow
Triggers

Labels
General



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 accepts 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 Properties 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 Strategy 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 interarrival 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.

Properties pages

ProcessTimes
Breakdowns
Combiner
Flow
Triggers
Labels

General

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, item type 1 needs to go through operations A, B, C, and D, and type 2 needs operations E, F, G, and H, but both types must share one station for their processes. Give the MultiProcessor 8 processes: A - H, and for type 1 have processes E - H have 0 process time, and for type 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.

Properties pages

MultiProcessor
Flow
Triggers
Labels
General

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 properties other than the simple setup, process times and flow page properties, 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 modeller-defined setup time.

Processing - The object is in its modeller-defined process time

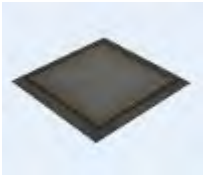
Blocked - The object has released its flowitem(s), but downstream objects are not ready to receive them yet.

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.

Properties pages

Processor
Breakdowns
Flow
Triggers
Labels
General



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 if 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

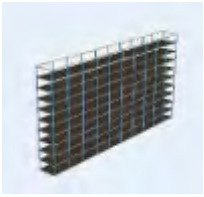
Collecting - The Queue is collecting flowitems for a batch.

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.

Properties pages

Queue
Flow
Triggers
Labels
General



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.

rackgetnrofbays(obj rack)

This command returns the total number of bays the rack has.

rackgetnroflevels(obj rack [,num bay])

This command returns the number of levels in the given bay of the rack.

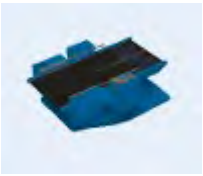
States

The Rack doesn't implement any states. Use the content graph to get statistics.

Properties pages

Rack
SizeTable
Flow
Triggers
Labels
General

Separator



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 modeller-defined setup time.

Processing: The object is in its modeller-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.

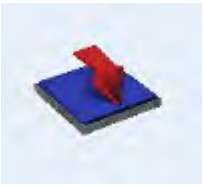
Waiting for Transport: The object has released a flow item, and a downstream object is ready to receive it, but a transport object has not picked it up yet.

Down: The object is broken down.

Properties pages

ProcessTimes
Breakdowns
Separator
Flow
Triggers
Labels
General

Sink



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.

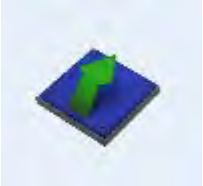
States

The Sink does not implement any states. Refer instead to its input statistics.

Properties pages

Sink
Flow
Triggers
Labels
General

Source



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 labels 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 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 interarrivaltime 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 interarrivaltime 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, 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.

Properties pages

Source
Flow
Triggers
Labels
General

TaskExecuters Concepts

TaskExecuter is the top level class for several objects in the library. Operators, Transporters, ASRSvehicles, Cranes and other mobile resources inherit from the TaskExecuter 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 TaskExecuters, refer to Lesson 2 of the Tutorials.

Details

TaskExecuters and their sub-classes are able to execute task sequences, perform collision detection, and execute offset travel.

The TaskExecuter is also a sub-class of the Dispatcher class, and thus a TaskExecuter 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 TaskExecuter 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 TaskExecuter's task sequence queue, and if the task sequence is still in the queue when the TaskExecuter finishes its active task sequence, the TaskExecuter will then execute the task sequence.

User-Defined Properties

All TaskExecuters 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 TaskExecuter'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 TaskExecuter 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 TaskExecuter'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 TaskExecuter should execute offset travel to the load/unload location when it loads or unloads a flowitem.

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 TaskExecuter will wait before loading the flowitem and moving on to the next task. Note that if the TaskExecuter 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 TaskExecuter will wait before unloading the flowitem and moving on to the next task. Note that if the TaskExecuter 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 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.

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 TaskExecuter. 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 TaskExecuter uses this distance to travel its offset. Also, for a load and unload task, the TaskExecuter 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 TaskExecuter and its sub-classes have the capability of detecting collisions with other objects. Collision detection is performed by adding collision members to a TaskExecuter, then adding collision spheres to it and its collision members, then executing logic when one of the spheres of the TaskExecuter collides with one of the spheres of one of its collision members. Each collision sphere you specify has a location in the TaskExecuter's frame of reference, and a radius. The TaskExecuter repetitively checks for collisions at time intervals that you specify. At each collision check, the TaskExecuter 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 TaskExecuter 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 TaskExecuter 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 TaskExecuters 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.

States

The TaskExecuter's states are purely dependent on the types of tasks that the TaskExecuter performs. Many tasks are associated with a hard-coded state, but with some tasks the modeler can specify an explicit state for the TaskExecuter to be in when executing that task. Here are some of the states that you will see often with a TaskExecuter. 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. Blocked: The object is currently traveling, but is blocked on the network.

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.

Properties pages

ASRSvehicle
Breaks
Collision
Triggers
Labels General

Related Topics

Task Sequences

The BasicTE object is a TaskExecuter that is meant for developers to create user libraries with. It passes almost all inheritable TaskExecuter logic to picklist functions, so that user library developers can specify virtually all functionality for the TaskExecuter.

Details

The BasicTE is a sub-class of the TaskExecuter. 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.

Properties pages

BasicTE Page
TaskExecuter
Breaks
Collision
Triggers
Labels
General

Related Topics

Task Sequences

Crane

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 Properties 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.

Properties pages

Crane
TaskExecuter
Breaks
Collision
Triggers
Labels
General

Related Topics

Task Sequences

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.

Ports

You use input and output ports to associate teams of task executers. When you connect the output ports of a dispatcher to the input ports of a task executer, this makes that task executer a member of the dispatcher's "team". When the dispatcher receives a task sequence, it will dispatch the task sequence to one of the task executers connected to its output ports, depending on logic you define.

Dispatching Logic

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.

Properties pages

Dispatcher
Triggers
Labels
General

Related Topics

Task Sequences

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 Elevator 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.

Properties pages

Breaks
Collision
Triggers
Labels
General

Related Topics

Task Sequences

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 TaskExecuter. 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 TaskExecuter. 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 has animations in order to Walk, Walk Loaded, and Stand. These animations can be viewed in the Animation Creator by right clicking an Opearting and selecting Edit > Animations....

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 TaskExecuter states.

Properties pages

TaskExecuter
Breaks
Collision
Triggers
Labels
General

Related Topics

Task Sequences

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, 6 joints rotate to move the flowitems.

Details

The Robot is a sub-class of the TaskExecuter. It implements offset travel by rotating its joints. Note that by default, the x/y/z location of the Robot does not change at all when it does offset travel. 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. 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.

Properties pages

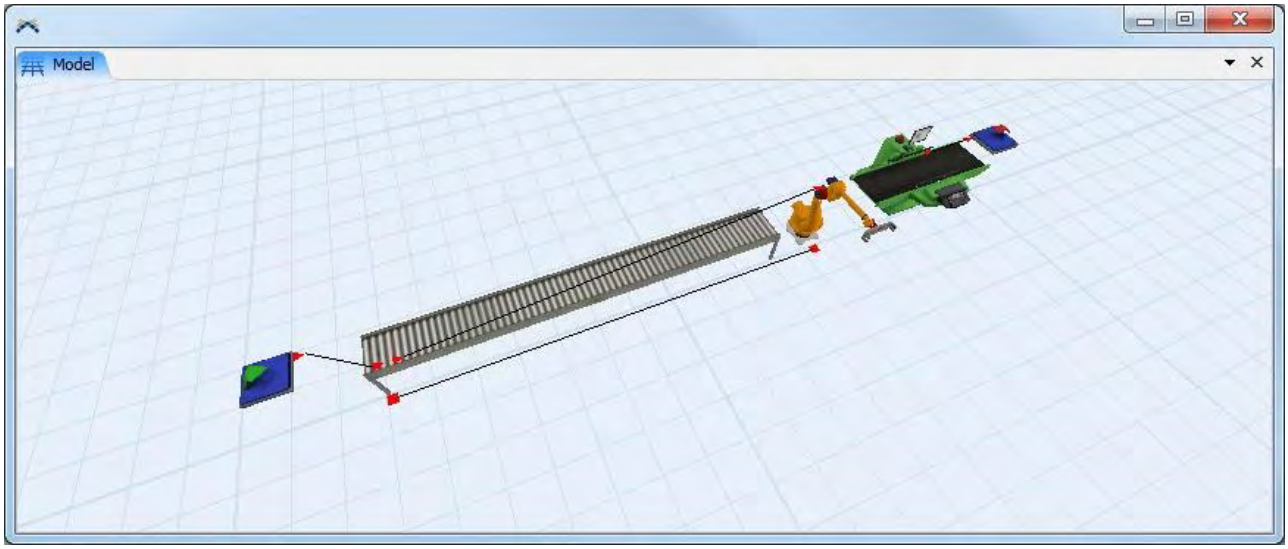
Robot
Breaks
Collision
Triggers
Labels
General

Robot Moving Flowitems

Getting the Robot to move Flowitems

The Robot object is used in your model in the same way that the Operator object is used.

1. Create a model: Source > Conveyor > Processor > Sink connected using the ‘A’ key.
2. Open the Conveyor's Properties window. Under the Flow tab check the Use Transport checkbox.
3. Position a Robot between the Conveyor and Processor objects as shown using Grid Snap.
4. Make an ‘S’ connection between the Conveyor and the Robot.
5. Press Reset. Your model should look like the following screenshot.
6. Press Run!



The Robot responds to task sequences created by fixed resources much like the Crane object, more specifically the Robot responds to FRLOAD, FRUNLOAD, TRAVELTOLOC and TRAVELRELATIVE task types. The Robot has 4 modes of operation, each mode changes the way the Robot responds to a task and more importantly how long it takes to perform the task.

1. Define Move Time - This is the default option. Here you define a move time explicitly with code.
2. Use Simple Motion Method - This option allows you to provide a few basic speed inputs that determine the move time.
3. Define Motion Paths - The most flexible option, explained below.
4. Use Joint Speeds - Here you define a speed for each of the Robot's joints.

Robot Motion Paths

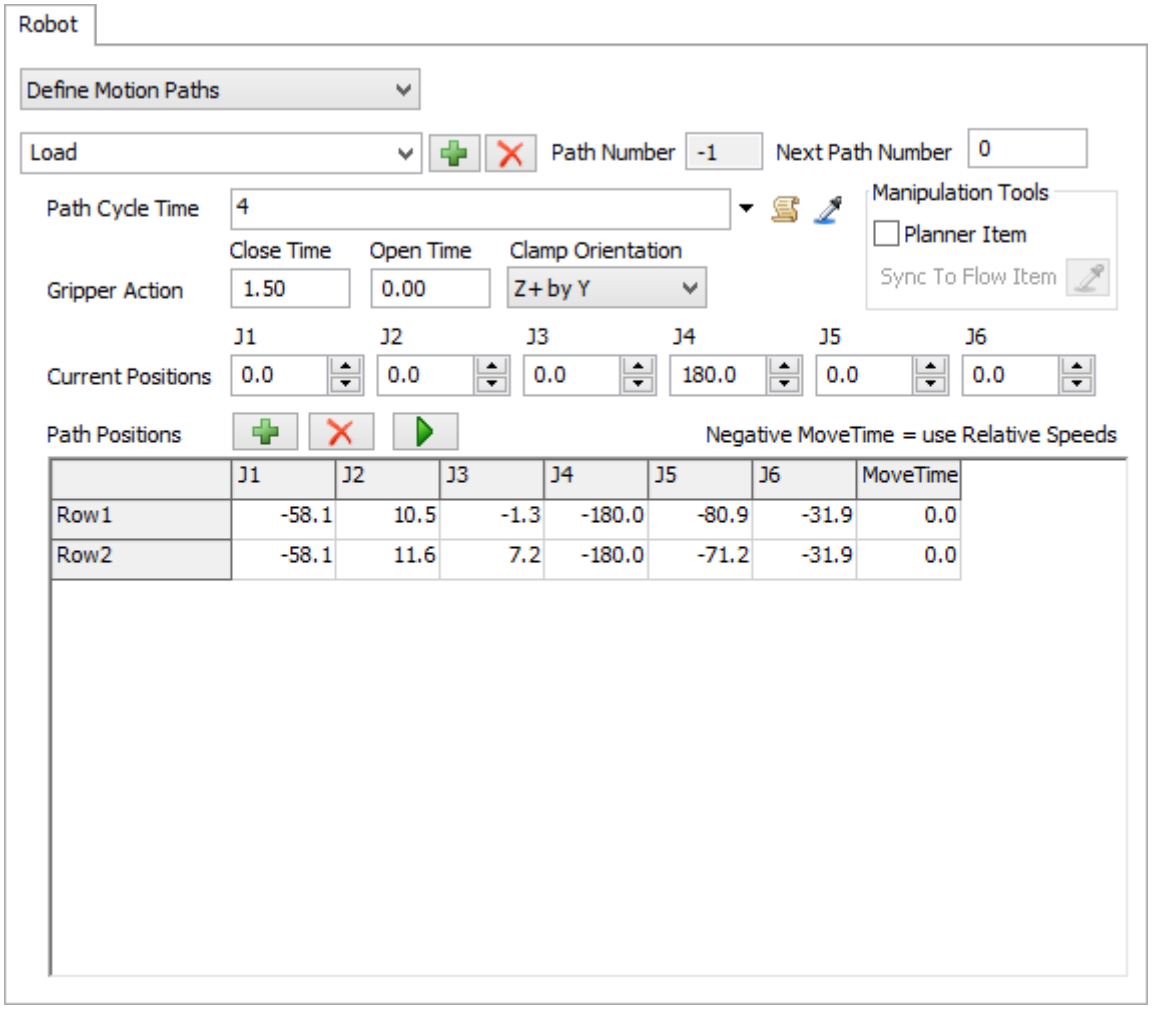
This is the most advanced mode that allows the user to design paths for the robot to follow. By default, the Robot contains 2 paths (called Load and Unload) to demonstrate some of the parameter options.

Topics

- What is a Path?
- What controls the timing of a Path movement?
- How do I create my own paths?
- How are the Gripper Action Times applied?
- How does the Robot choose which path to use?

What is a Path?

A Path is a set of movements as defined by a series of robot positions. The Robot stores robot positions in a tabular format where each row is a record of the 6 joint angles that make-up the given position. The default Load path for example contains 2 positions. When the Robot performs this path, it will move from its current position to Row1 position, followed by Row2 position. You can preview the motion of the robot for this path by pressing the  button. You can also directly view an individual row's position by clicking on the row in the table and pressing the  button to apply that row to the Robot's joints.



What controls the timing of a Path movement?

There are various options for controlling the time a path movement takes in the simulation. Note that the Gripper Action Times are added to the end of a movement.

- If Path Cycle Time > 0 - You can specify the length of time the Robot takes to complete the series of path movements. In this case, the MoveTime column in the Position Table affects the distribution of the Path Cycle Time delay time as follows:

  Without Weighting - Leave the MoveTime column in the Position Table all zeros or equal values and the Path Cycle Time will be divided evenly amongst the number of positions in the path.

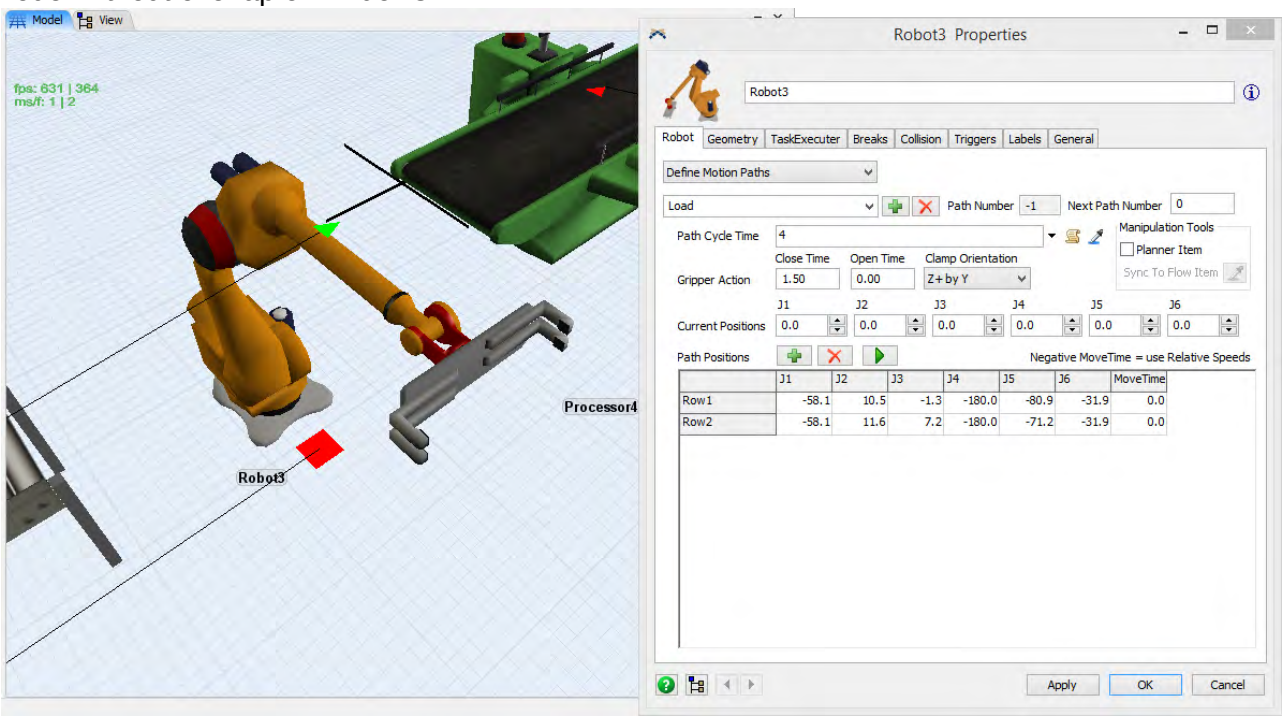
  With Weighting - Adjust the values in the MoveTime column to represent the ratio of the Path Cycle Time you wish to assign to each position row.

  Example - A path contains 3 position rows with 1, 1, 2 respectively in the MoveTime column and the Path Cycle Time was 5.2s. The AR would take 1/(1+1+2) * 5.2s to move to the first position, an equal time to move from the first position to the second position, and 2/(1+1+2) * 5.2s to move from the second position to the third position.

- If Path Cycle Time = 0 and MoveTime column fields > 0 - When the Path Cycle Time field returns a 0, the Robot will use the sum of the values in the MoveTime column as the cycle time for the path. In this way you have precise control of the move time from each position to the next. This method can also be used to insert a stop delay in the movement; set 2 rows in sequence to the same position and use the MoveTime as the delay time.
- Negative Move Time affects animation only (Relative Speeds) - For a given movement within a path, if the MoveTime value is a negative value the Robot will still treat it as if it were positive, however the joint speeds will be adjusted such that they are proportional to the Relative Speeds entered in the Robot Geometry tab. Note that this is purely a change in the animation and does not affect the movement cycle time. Usually, entering the actual max speed of each joint (from manufacturer) into the Relative Speeds will give best results.

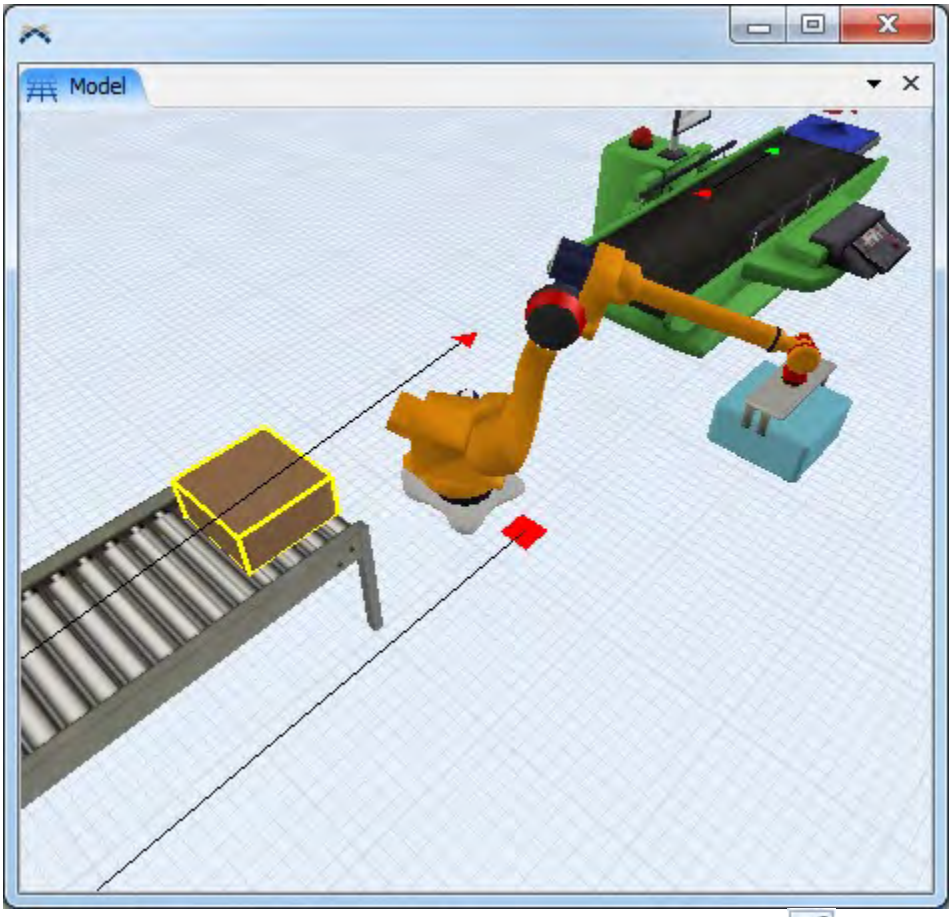
How do I create my own Paths?

1. Position and size the model view such that you can see the Robot Properties GUI and the robot in your model without overlap of windows.

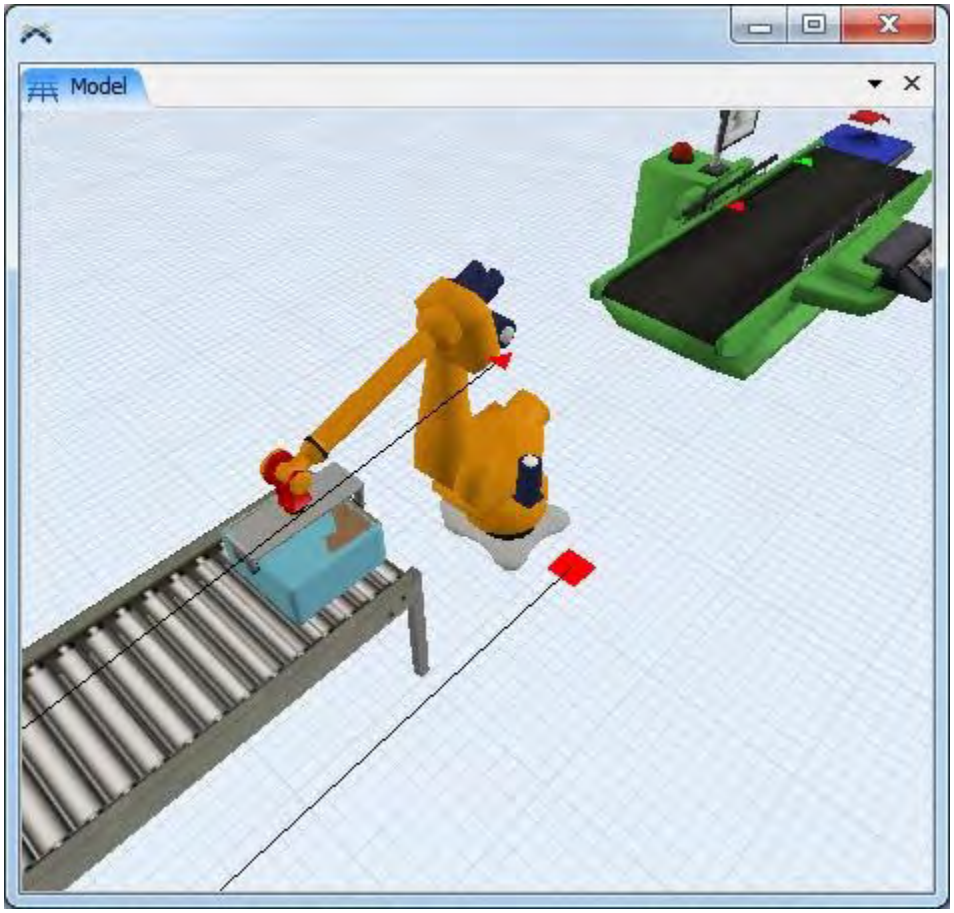


2. Click Add Path. This will create a duplicate of the currently selected path.
3. In the path's name field type a descriptive name for the path. The name has no functional effect on the behavior of the Robot.
4. Clear the Position Table by pressing  until there is only one row remaining.
5. Turn ON the Path Planner by clicking the Planner Item checkbox. A box named PlanningItem will appear in the 3D view in the Robot's gripper. You can move and rotate the PlanningItem with your mouse and the robot will move accordingly. You should have the Resize and Rotate Objects feature checked (found under Flexsim's Edit menu). Also, you might consider disabling Snap To Grid before proceeding further.
6. Begin by recording the end position of your path as it is the most important position. In this case, I wish to record a second path for the Robot to follow before loading the FlowItem from the conveyor. Thus, the end position would have the robot gripped onto the FlowItem as it sits on the end of the conveyor.

a. Press the model Step button (in the simulation run panel) to advance the model run forward by event until a FlowItem appears at the end of the conveyor ready for pickup. You may need to recheck the Planner Item checkbox if you reset the model since the robot hides the item when the model is reset.



b. In the Robot properties window, click the Sync to Flow Item button next to the Planner Item check box. The cursor will change to a "sampling" cursor. Now click on the flow item in the model view.



c. Choose an option from the Clamp Orientation drop-down to determine the orientation from which the clamp will grab the flow item.
d. Make sure to choose the proper Close Width drop-down option for determining closed grip width associated with this path. (The open gripper width parameter is found in the Geometry tab)
e. Now that the Robot is gripping the item as desired, save the Robot's joint positions to the row in the table. Click on the first row in the table, and press the button to save the Robot's current joint positions to that row.
f. The recorded position is actually the end position the Robot must have at the end of a LOAD task (call it the pickup position). Technically this is all you need, however often you will want to insert position rows above this row that control how the Robot gets to the pickup position. For example you may want to have the Robot first go to a position above the flow item, then drop down to clamp it. To do this, add a new row by clicking on the first row and clicking the button. This will add a row before the selected row. Then click on the PlanningItem and use the scroll wheel on your mouse to change the Z location of the PlanningItem while staying directly above the pickup position. Then click on the newly created row in the table and press the button to update that row with the new position.
g. If you were to reset and run the model, the Automatic Path Selection would likely not choose your new path to load the FlowItem because the original Load path still exists and is positioned higher in the Path Selector list. Choose the original Load path and delete it.
h. Reset and Run the model.

How are the Gripper Action Times applied?

The Gripper Action Times are added to the Path movement time since a close or open clamp action is performed at the end of a LOAD or UNLOAD task respectively (after the path movement is complete).

1. If you enter a positive value into the Close Time field, the Robot will close its clamp at the end of the Path movement and the value entered will be the delay time associated with the action. Note that if the Robot is responding to a task that does not involve a FlowItem, the closed clamp width will be 0.
2. If you enter a positive value into the Open Time field, the Robot will open its clamp at the end of the Path movement and the value entered will be the delay time associated with the action. The clamp will open to the Open Gripper Width specified in the Geometry tab.
3. If both the Close Time and Open Time fields are 0, the clamp will not move.
4. If you enter a negative value into either the Close Time or the Open Time fields, the Robot will perform the action in 0 time. (as of FlexSim v5)

How does the Robot choose which Path to use?

By default, the robot searches for the path with an end position (last position row) which results in the smallest distance between the Robot's end effector and x,y,z location as requested by the task it is responding to.

Controlling the Path that the robot uses for a given task

There are two ways to override the Robot's automatic path selection. It is helpful to understand that the default task sequence that is generated when the Use Transport is checked on a Fixed Resource contains 2 tasks that the Robot responds to (FRLOAD and FRUNLOAD tasks).

1. Specify directly on a custom Task Sequence - Knowledge of writing your own custom task sequences is assumed in this section. Specifying a negative end speed variable when writing the task sequence will tell the Robot to use the path number that matches this end speed (Path Number = end speed). The task types that the Robot responds to are listed below with the location of the end speed variable. See the Task Type Quick Reference found in the User Manual under Task Sequences.

TASKTYPE_	Location of end speed variable
LOAD and FRLOAD UNLOAD and FRUNLOAD	var2
TRAVELRELATIVE TRAVELTOLOC var4 (for movements that don't involve FlowItems)	

2. (As of FlexSim v5) When using a TRAVELRELATIVE task type with a negative endspeed to choose the path you can now use var1 (the x parameter) to tell the Robot which position row to stop at if you don't want it to perform the entire path.
3. Next Path Number (great for known path sequences such as palletizing) - Palletizing is simply a sequence of LoadPath1-UnloadPath1-LoadPath2-UnloadPath2… once the Robot unloads the last FlowItem on the Pallet, the sequence repeats. Every Path has a "Next Path Number" field in which you

can specify the path number you want to be used following the currently selected path. When the Robot executes a path that has a non-zero Next Path Number specified, it will store this path number into memory until the next time in the model run it performs a task that requires automatic path selection (not specified on the task itself). Instead of searching for a suitable path, the Robot will choose the path number stored in memory.

4. If during the model run you wish to clear the Robot's Next Path Number memory, simply set its "nextpathselected" variable to zero and the Robot will return to conducting automatic path selection for a best-fit path (if not specified on the task).
5. A combination of a custom task sequence and Next Path Number can save a lot of time!
6. Take the example of a palletizing operation with 4 FlowItems per pallet.
    a. Make 1 load path and 4 unique unload paths to show the positioning of FlowItem on the pallet. The load path will be Path Number 1, and the unload paths will occupy Path Numbers 2 through 5. Don't forget about the trick using the model "Step" and the "Match Highlighted Item" buttons we learned in "How do I create my own Paths?".
    b. The path sequence we want is LoadPath1-UnloadPath1-LoadPath1-UnloadPath2-LoadPath1UnloadPath3- LoadPath1-UnloadPath4 or Path Numbers 1-2-1-3-1-4-1-5. Since the LoadPath1 is always used for the FRLOAD task, we can specify var2 as -1 on this task.
    c. In the Path Planner, go to the load path (Path Number 1 in this example) and specify a Next Path Number = 2, as this is the first unload path of the cycle.
    d. Go to the first unload path (Path Number 2 in this example) and specify a Next Path Number = 3 (-3 will work as well). Path Number 3 should have a Next Path Number = 4, and Path Number 4 should have a Next Path Number = 5. Path Number 5 should have a Next Path Number = 2 to complete the cycle.

Transporter

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.

Properties pages

Transporter
Breaks
Collision
Triggers
Labels
General

Related Topics

Task Sequences

NetworkNode

Topic List

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 travelling 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 tutorials.

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 TaskExecuters to the NetworkNodes they will be stationed at for the beginning of the simulation.

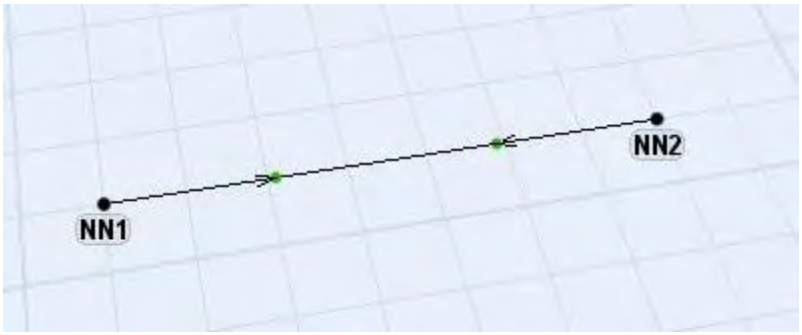
Connecting NetworkNodes to Each Other

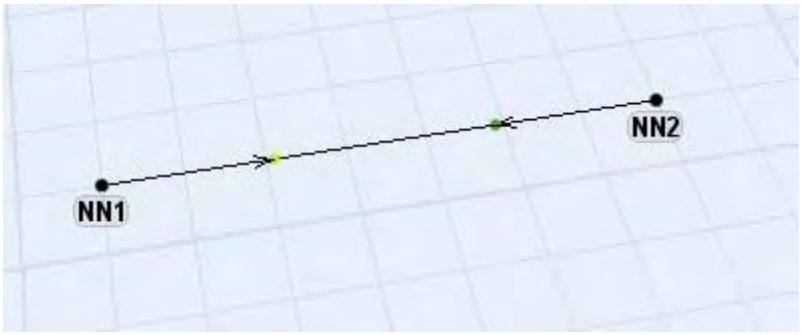
Each NetworkNode can have several paths connecting it to other NetworkNodes. Each path represents two single-direction travel paths. Each path direction can be configured independently.

Configuring Paths

To create a path between two network nodes, hold down the 'A' key, click on one network node, and drag to the other node.

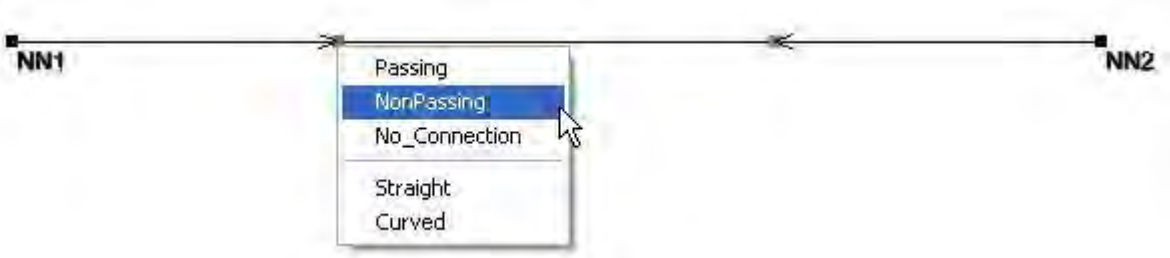
This will create two one-way, passing connections between the nodes. The path is drawn as a green tape between the two nodes. The tape is divided into two sides. Each side describes one direction of the path. Subsequent 'A' drag connections will toggle one direction of the path between passing and non-passing (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.



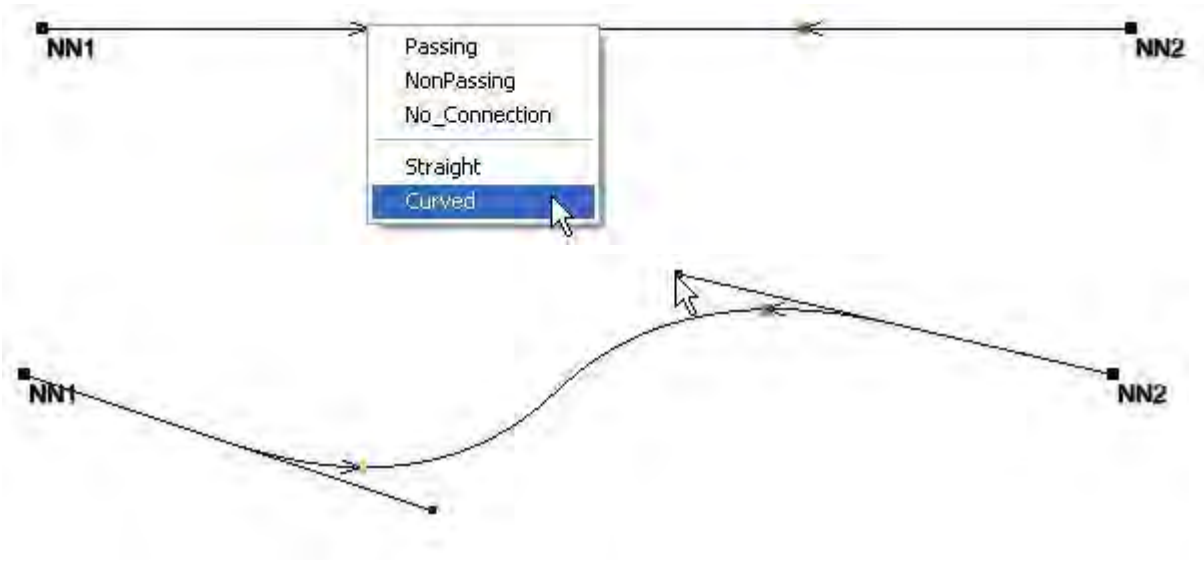


A 'Q' drag connection will toggle one direction of the path as "No Connection", which means travellers aren't allowed to travel in that direction. This type of connection is drawn in red. The diagram below shows a non-passing connection going left-to-right and a no connection going right-to-left. If 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 modeling utility.

Configuring Paths Through the Properties Window

When you open a NetworkNode's Properties window, the Paths page allows you to configure all onedirectional paths that extend out of that node. If you want to configure paths going into the node, then edit the properties 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:

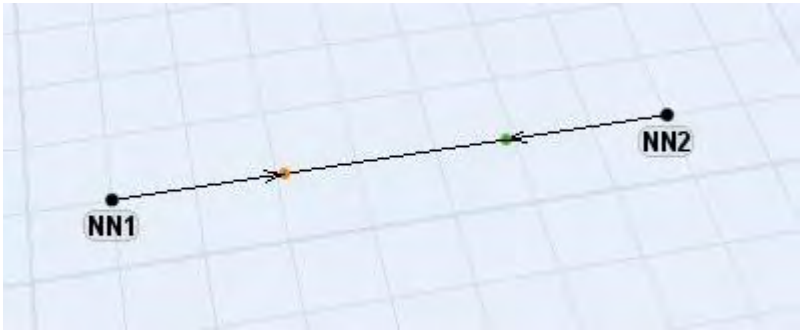
- Name - The name of a connection is simply for semantic purposes, and has no effect on model logic.
- Connection Type - There are three connection types: No Connection, Passing, and Non-passing. No connection means that no traveller should travel along this path, in the given direction. If this is selected, 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 travellers 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 traveller and the front of another traveller on the path.
- Speed Limit - This is a speed limit for the path. Travellers will travel the minimum of their own speed and the speed limit of the path. If the path is a Passing connection, then travellers will accelerate or decelerate to the appropriate speed once the get on the path. If it is non-passing, however, then travellers 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 travellers onto the edge until it has been opened again. However, travellers 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 TaskExecuter 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 TaskExecuter 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 TaskExecuters

To connect a NetworkNode to a TaskExecuter that you want to travel on the network, make an 'A' drag connection between the NetworkNode and the TaskExecuter. 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 TaskExecuter given a Travel task will use the network to get to the destination. It also means that the node that you connect the TaskExecuter to is the first node that it will travel to when it needs to travel through the network. Whenever a TaskExecuter finishes a travel operation, arriving at the node connected to the travel operation's destination object, the TaskExecuter will become inactive at that node, and the red line will be drawn to the TaskExecuter while he is doing other operations in that node's area. What this means is, the next time the TaskExecuter gets a travel task, he must return to the NetworkNode he was inactive at in order to get back onto the network.

(By "inactive" we mean that the TaskExecuter 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.)

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.

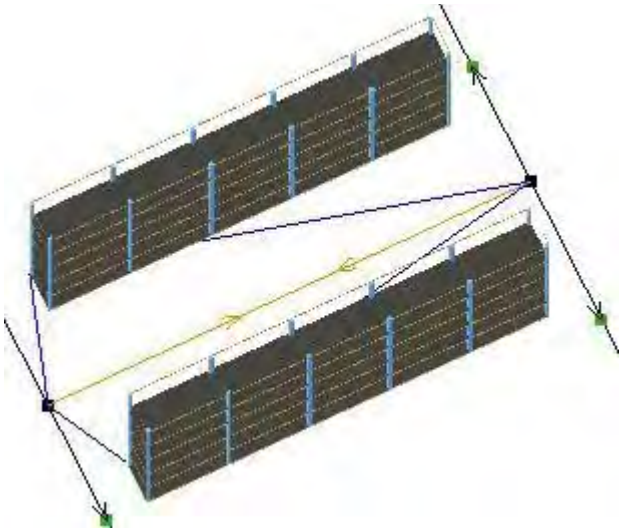
Maximum Number of Travelers

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 TaskExecuter 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 TaskExecuter 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 TaskExecuter 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 TaskExecuter 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 TaskExecuter 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.

States

The NetworkNode does not implement any states.

Properties Pages

NetworkNode
Triggers
Labels
General

Related Topics

Tutorial: Lesson 3
TrafficControl

TrafficControl

Topic List

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		
	Coming From	Continuing To
Area Entry	Non-member path	1. Member path OR 2. Member final destination node
Area Exit	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 r 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.

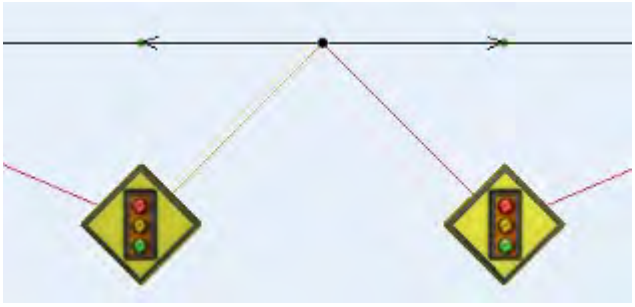
The traffic control selects modes based on entry requests. If there are no travelers in the area, then the traffic control simply waits. When the first traveler requests to enter a certain member path of the area, the traffic control searches its table to find a mode that includes that path. If found, 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.

Interaction

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

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 TaskExecuters to that NetworkNode, and then having them enter the model at the beginning of each simulation run.

Manipulating Speeds

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.

Customizing Area Entry Rules

You can also customize the rules by which a traffic control allows/disallows traveler entry into the traffic control area. For more information on this, refer to the command documentation for trafficcontrolinfo().

States

The traffic control does not implement states at this time.

Properties Pages

Traffic Control
Speeds
NetworkNodes
Triggers
Labels
General

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 or 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

Since the VisualTool works differently than other FlexSim objects, a detailed description of how it is used will now be explained. 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

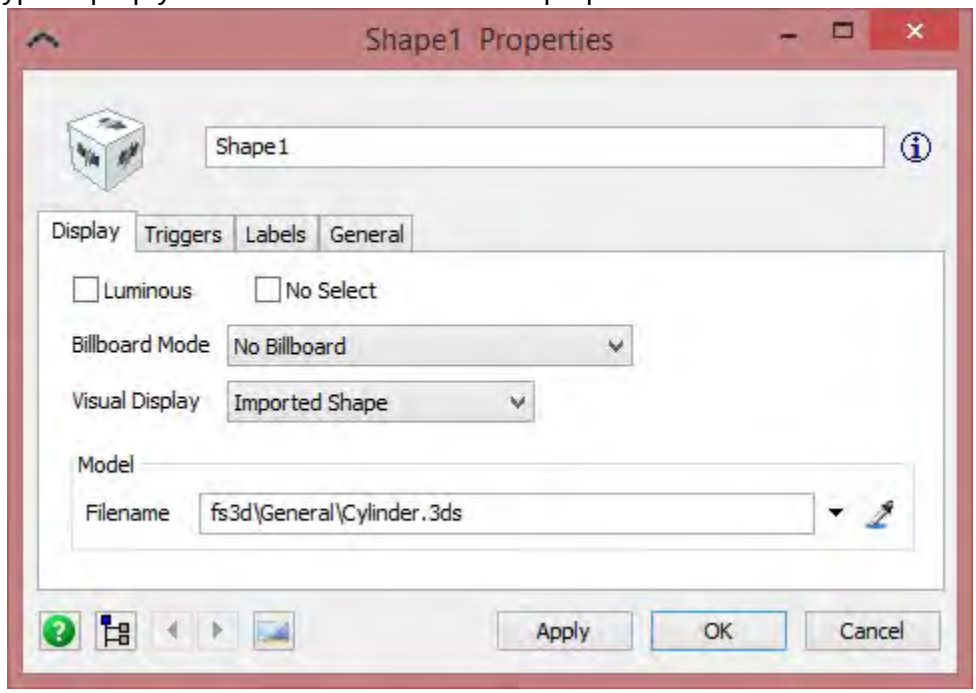
Using the VisualTool as a Container

The default setting for the VisualTool is a plane. When placed in the model the VisualTool will display a gray plane. The size and location of the plane can be set graphically in the 3D view window, or by using the VisualTool's Display tab (the use of the Display tab is explained in the section Using the Visualtool as a Plane, Cube, Column, or Sphere). A VisualTool can be used as a container by dragging objects from the 3D library into the VisualTool. This can be useful if you want to compartmentalize your model or you want to add custom objects to a user library from use in multiple models. This allows for easy updating when functionality changes. See the Example page for more information on containers.

If you want to collect statistics on the container use Tracked Variables or Process Flow.

Using the VisualTool as a Plane, Cube, Column, or Sphere

To use the VisualTool as a visual prop in you model is a simple process. On the "Display" tab, choose the type of prop you would like and define the properties.



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

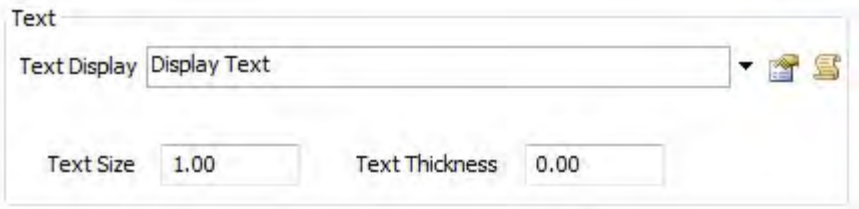
The cube, column, or spheres are simple shapes that can be assigned a texture much like the plane.

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), Google SketchUp (.skp), VRML 1.0 (.wrl), AutoCAD DXF (.dxf), and Stereo Lithography (.stl).

Using the VisualTool as Visual Text

3D visual text can be added to you model to display labels, statistics, or other information in your model. When the visual display is set to "Text" you are presented with a picklist of options for setting the text.

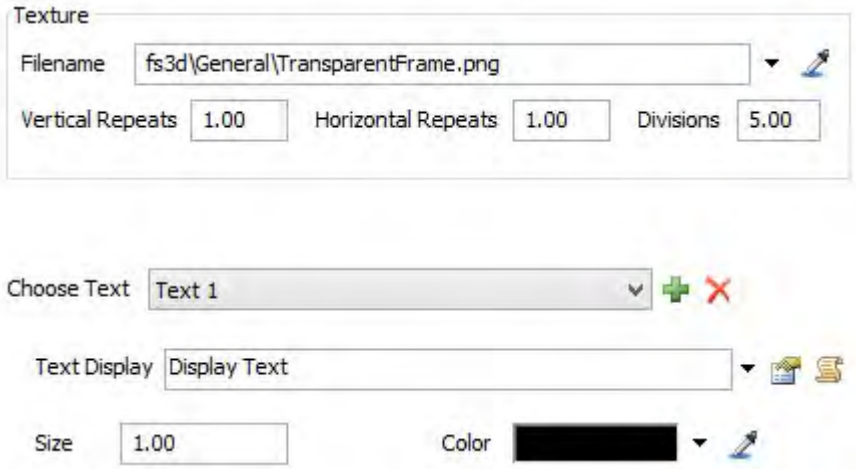


Pick options include simulation time, content, state, outputs, inputs, and many others. If any statistics are selected in the picklist the user must connect the VisualTool center port to the object that you want to display the information about. The text can be edited by selecting the code template button .

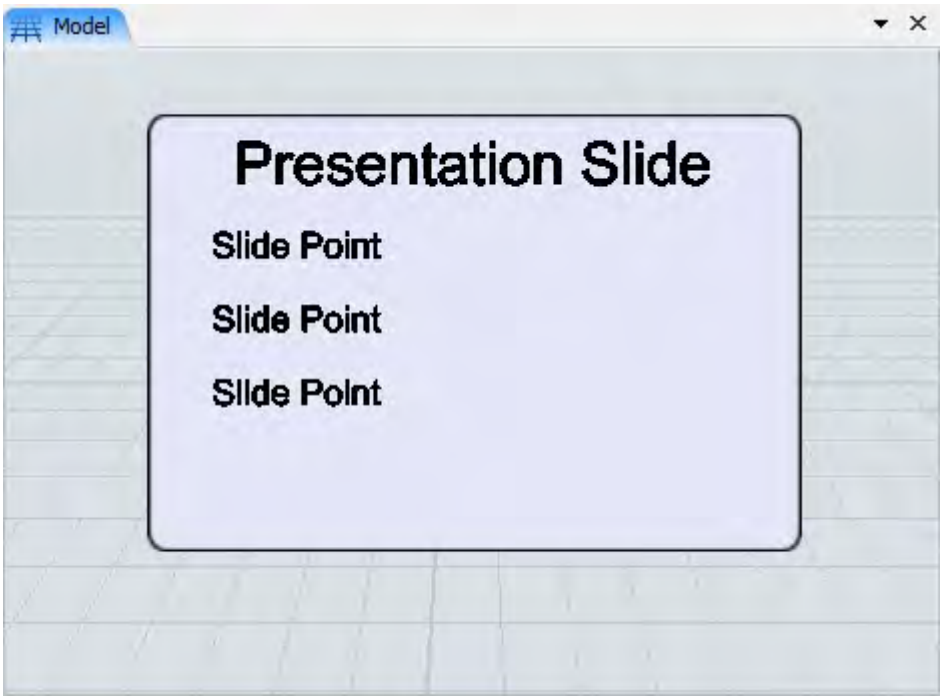
Using the VisualTool as a Presentation Slide

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 menu option Tools > Presentation > Presentation Builder.

When the visual display is set to "Presentation Slide" additional options will appear on the Display tab.



Press the Add button to add a new line of text to the slide. The first line of text is labeled "Text 1" and is the slide header. Additional text is added to the slide as line items. For example, if you were to add four text items to the presentation slide you would see the following:



Each text is given the default location on the slide as shown. By selecting the desired text in the drop down box, you can also change the text that is displayed using the Text Display list, the size of the text and the text color.

You can apply any background by selecting a texture on the Display tab or change the background color by selecting the color on the General tab.

Properties pages

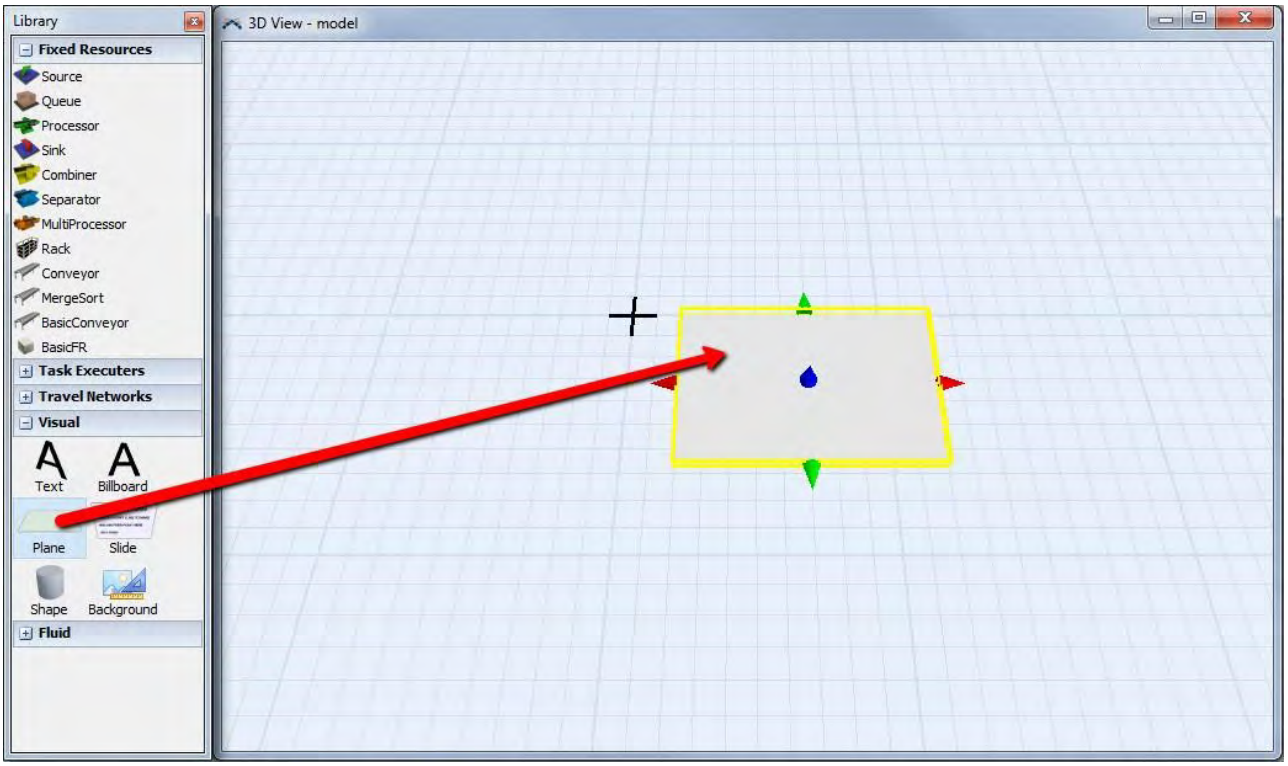
Display
Container Functionality
Triggers
Labels
General

VisualTool Example

Using the VisualTool as a Container

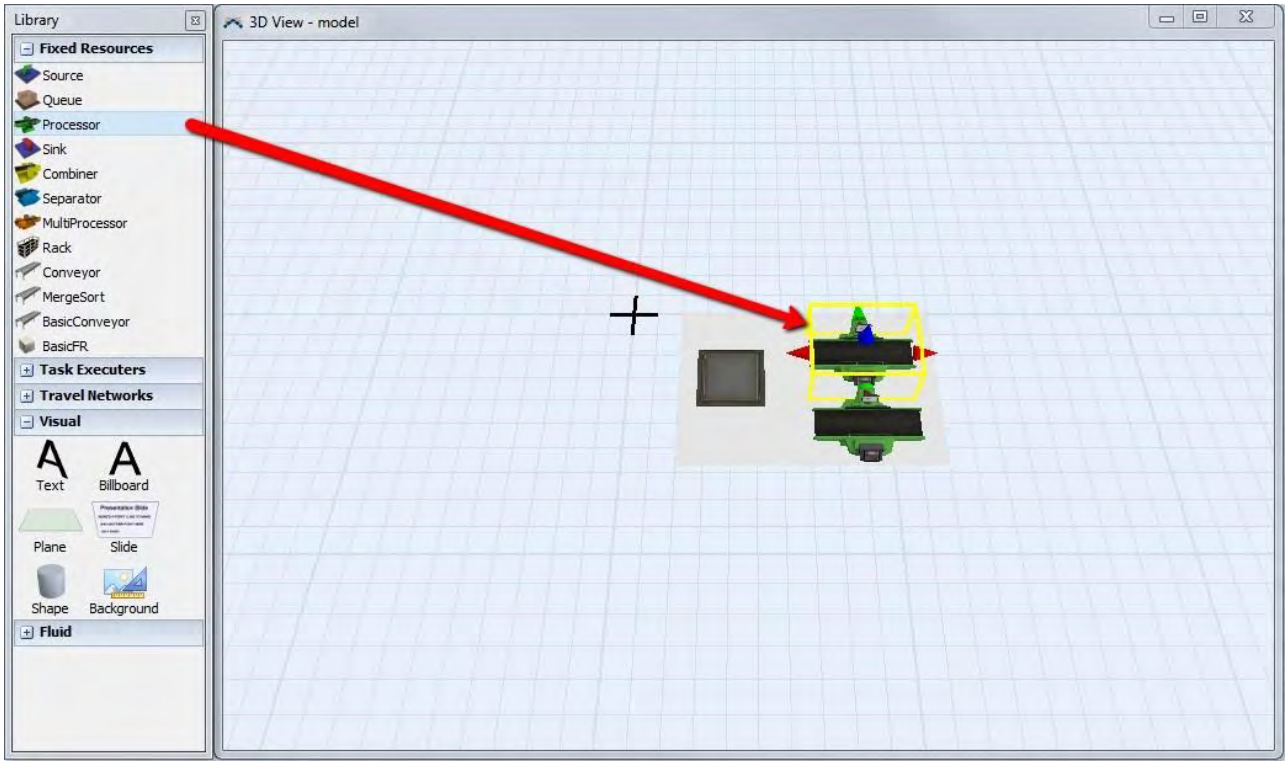
Step 1: Place a VisualTool into the model view

To add a VisualTool to the model, simply drag it from the library onto the model view.



Step 2: Drag-and-drop 1 Queue and 2 Processors into the VisualTool

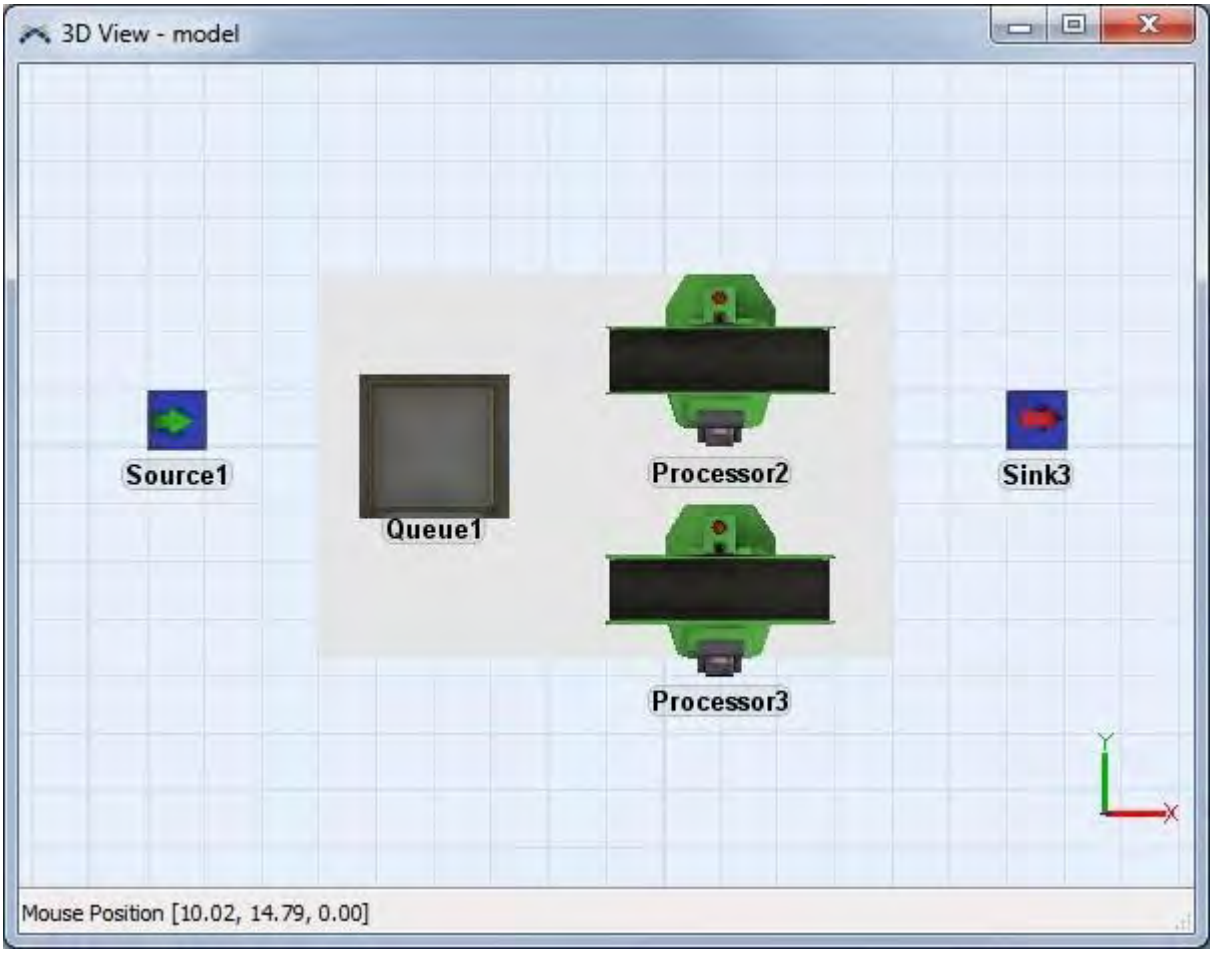
To add objects inside the container simply drag them from the library and place them on the VisualTool object.



When you place an object on the VisualTool it will automatically be place inside the VisualTool object. You can test this by selecting the VisualTool and moving it's location with your mouse. When you move the VisualTool the objects inside will move as well.

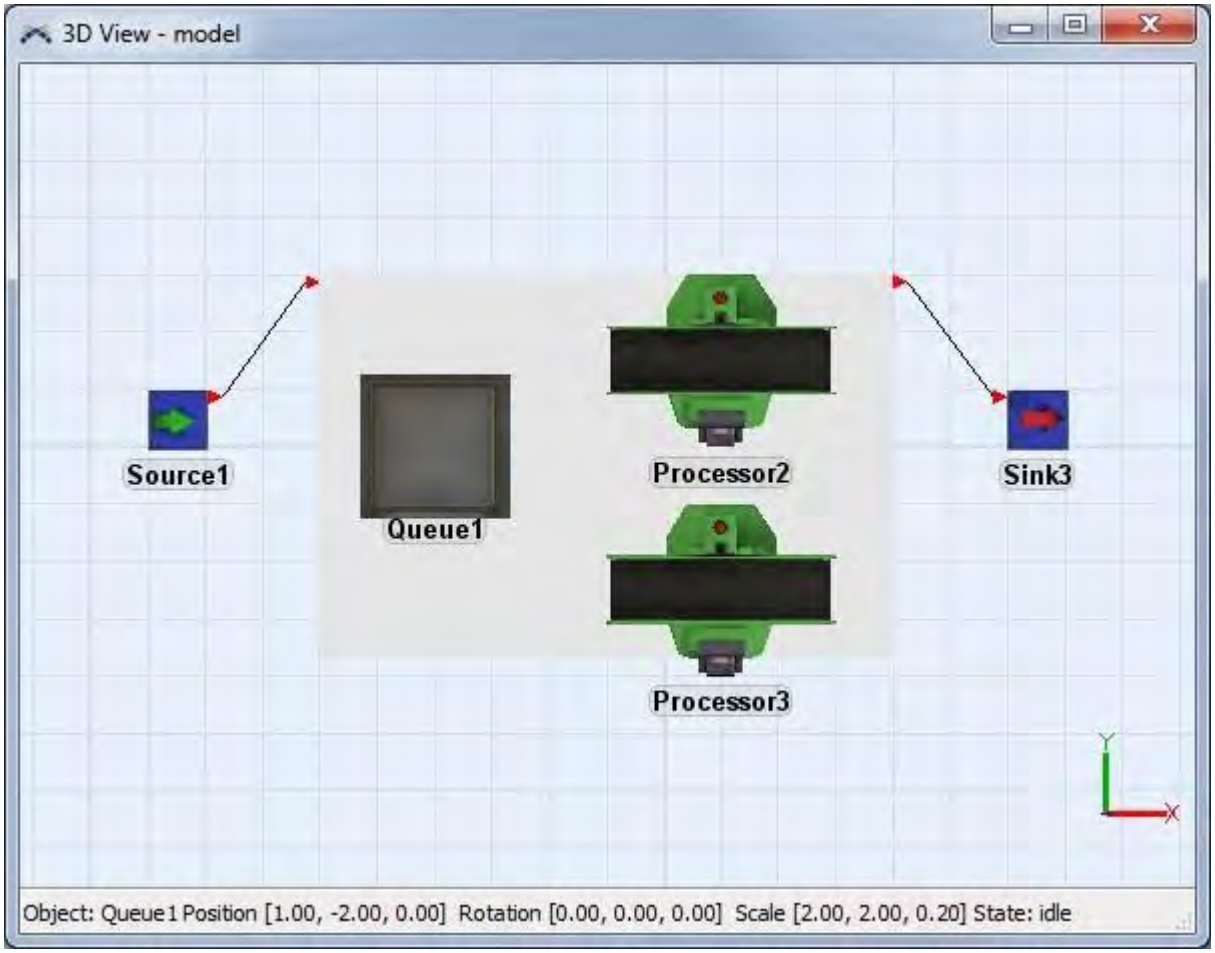
Step 3: Drag out 1 Source and 1 Sink into the model view

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.



Step 4: Connect to Source to the VisualTool, and the VisualTool to the Sink

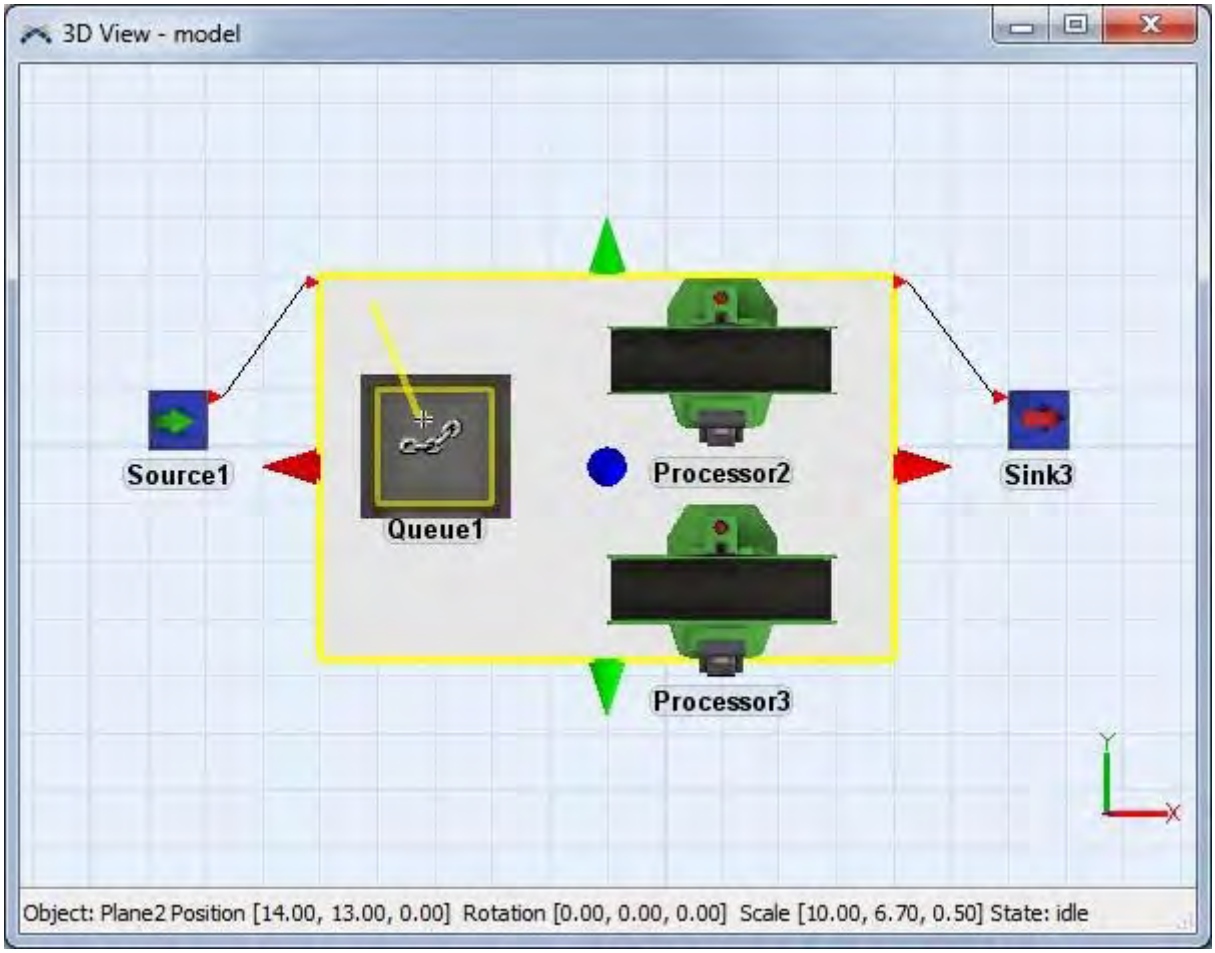
While pressing the "A" key on the keyboard, click-and-drag a connection from the Source to the VisualTool (not the Queue). When you release the left mouse button you will see a connection made between the Source and the VisualTool. Do the same thing to make a connection between the VisualTool and the Sink.



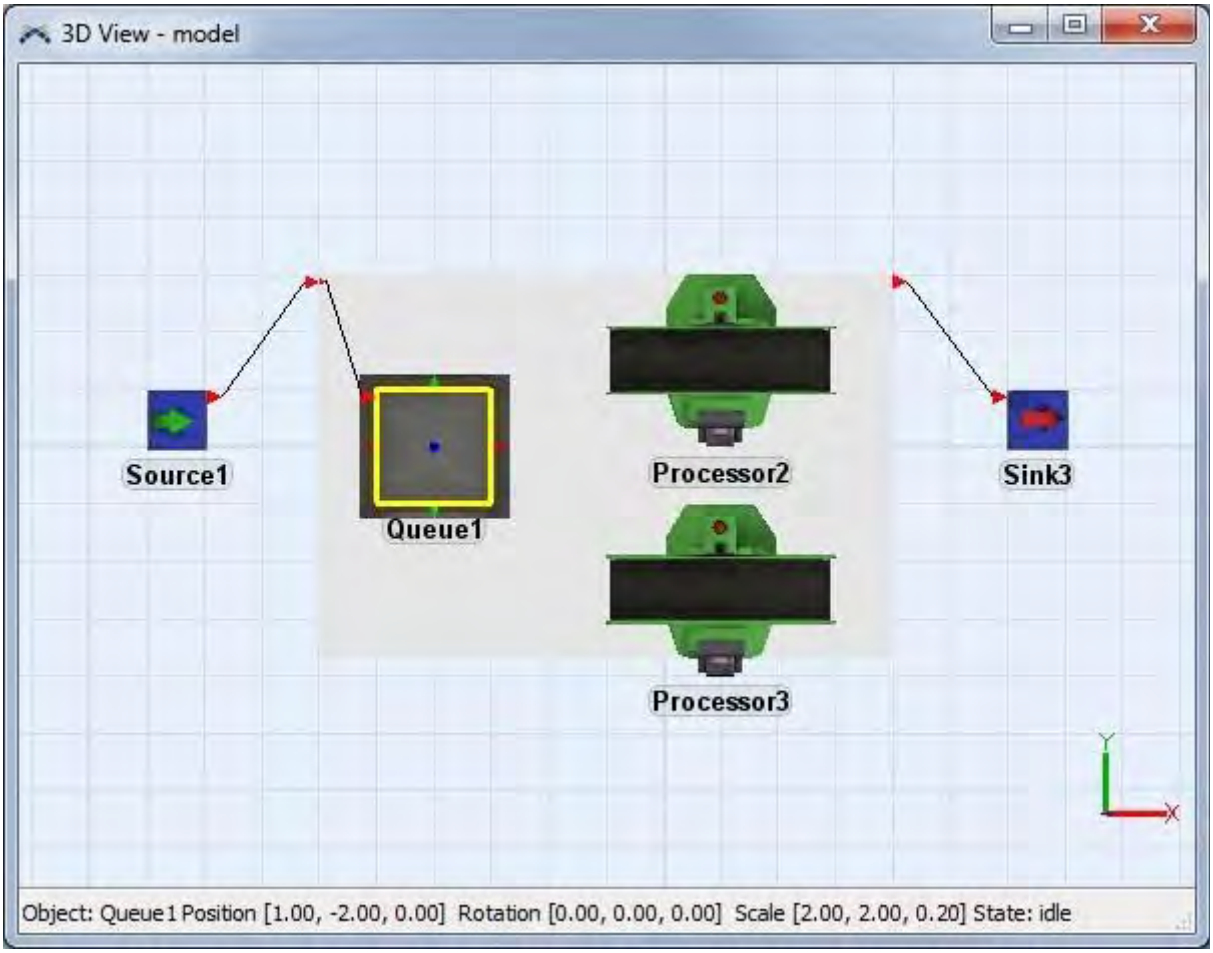
At this point the Source and the Sink are connected to the container (VisualTool). Now we will connect the container to the model inside.

Step 5: Connect the Container to the Queue

Drag a connection from the container to the queue.

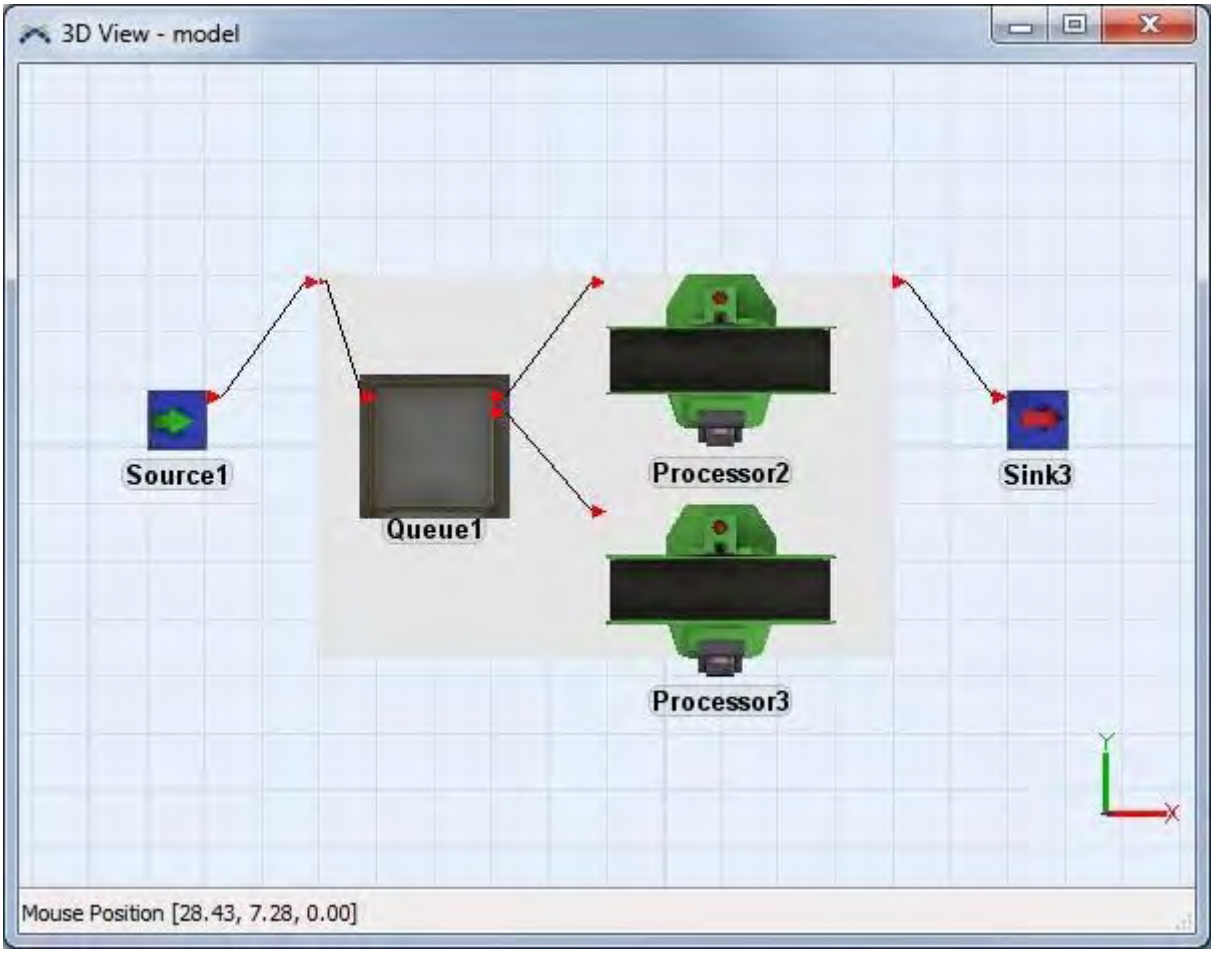


When you release the left mouse button you will see a connection from the internal port (a smaller arrow) to the queue.



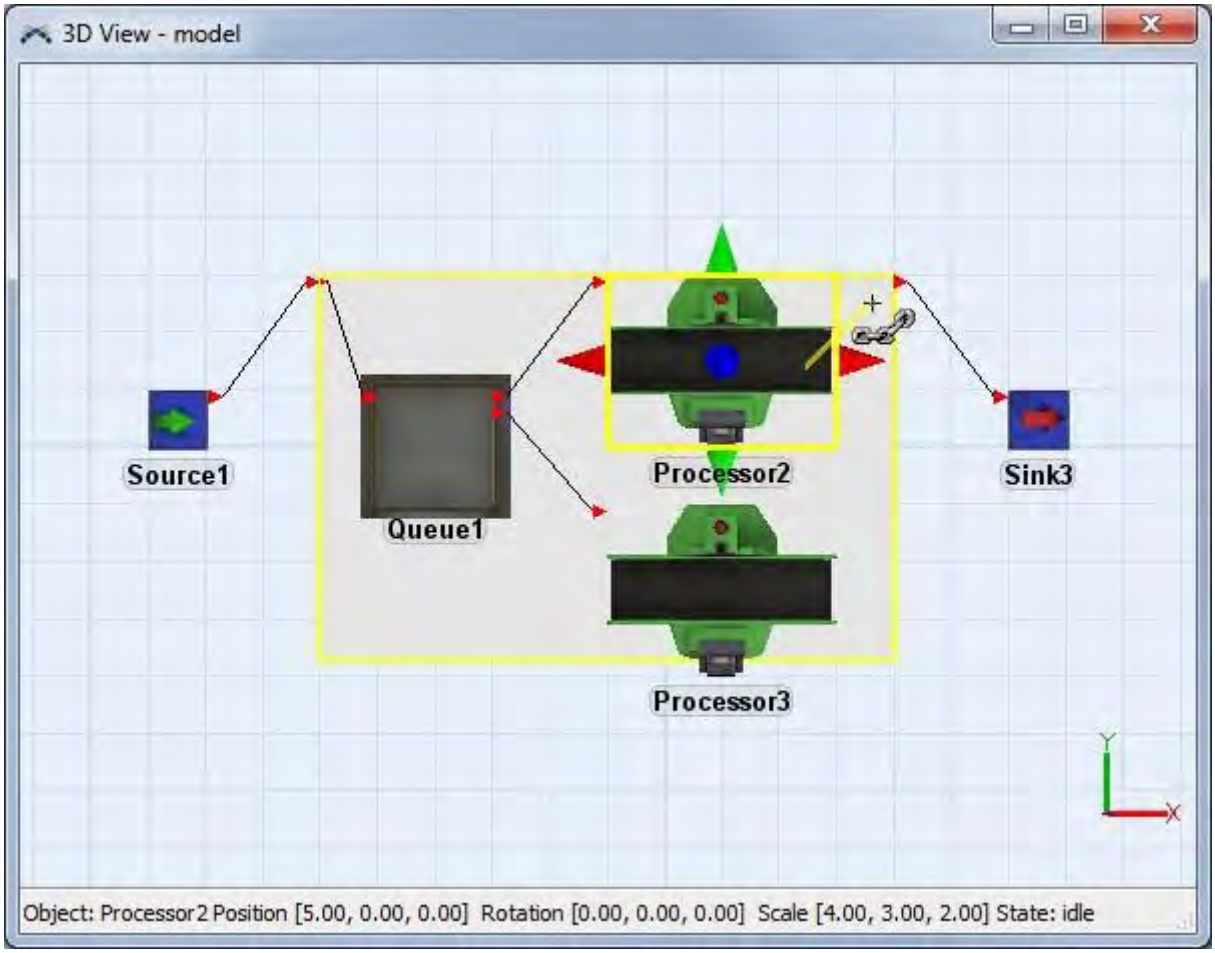
Step 6: Connect the Queue to the Processors

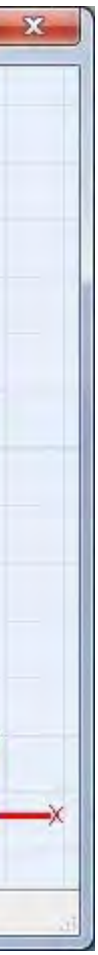
Following the same procedure connect the queue to the 2 processors.

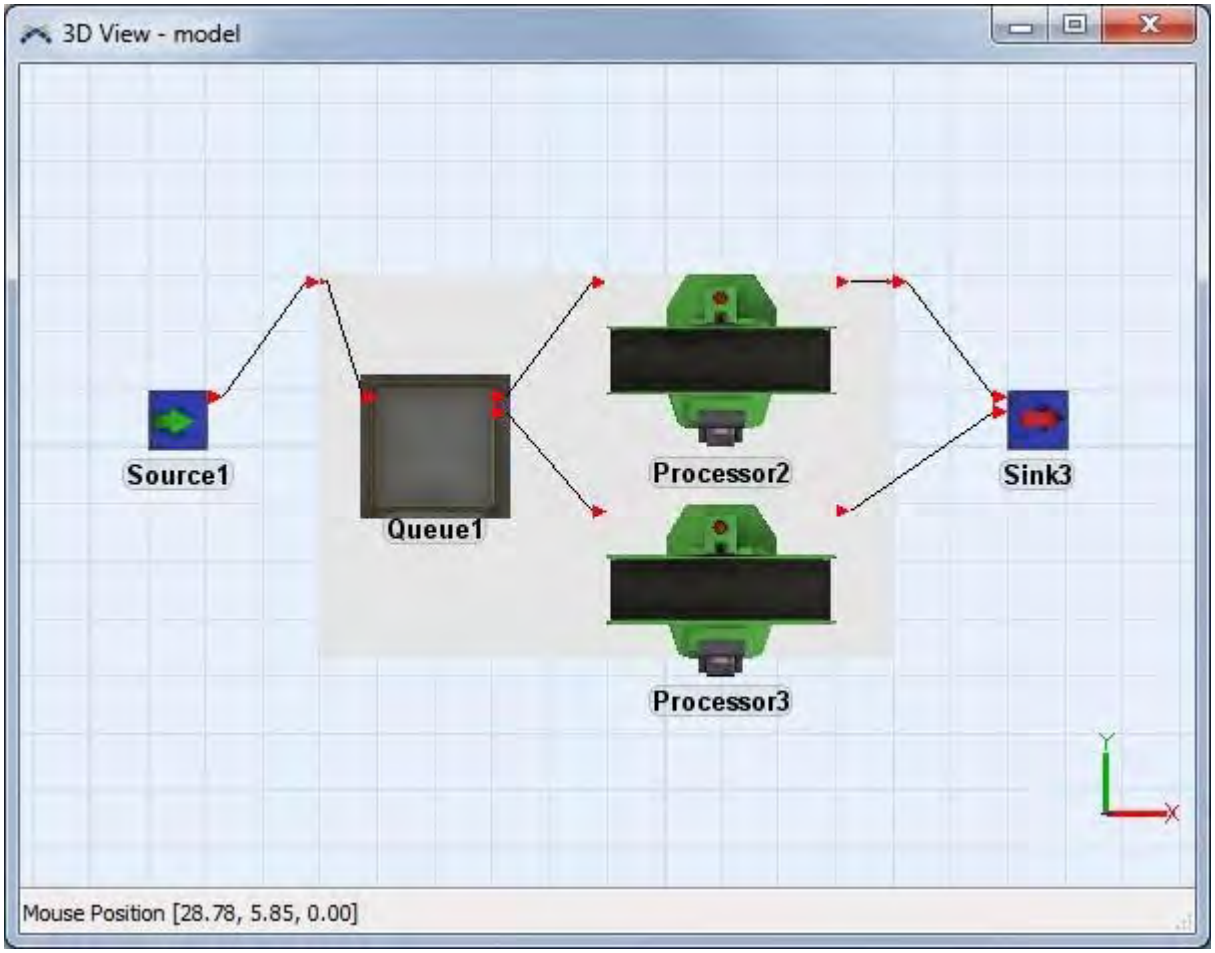


Step 7: Connect the Processors to the container or directly to the Sink

There are 2 ways to connect "into" or "out" of a container. The first way was shown in step 4 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.


3D View - model
Source1
Queue1
Processor2
Processor3
Sink3
Y
X
Object: Processor2 Position [5.00, 0.00, 0.00] Rotation [0.00, 0.00, 0.00] Scale [4.00, 3.00, 2.00] State: idle

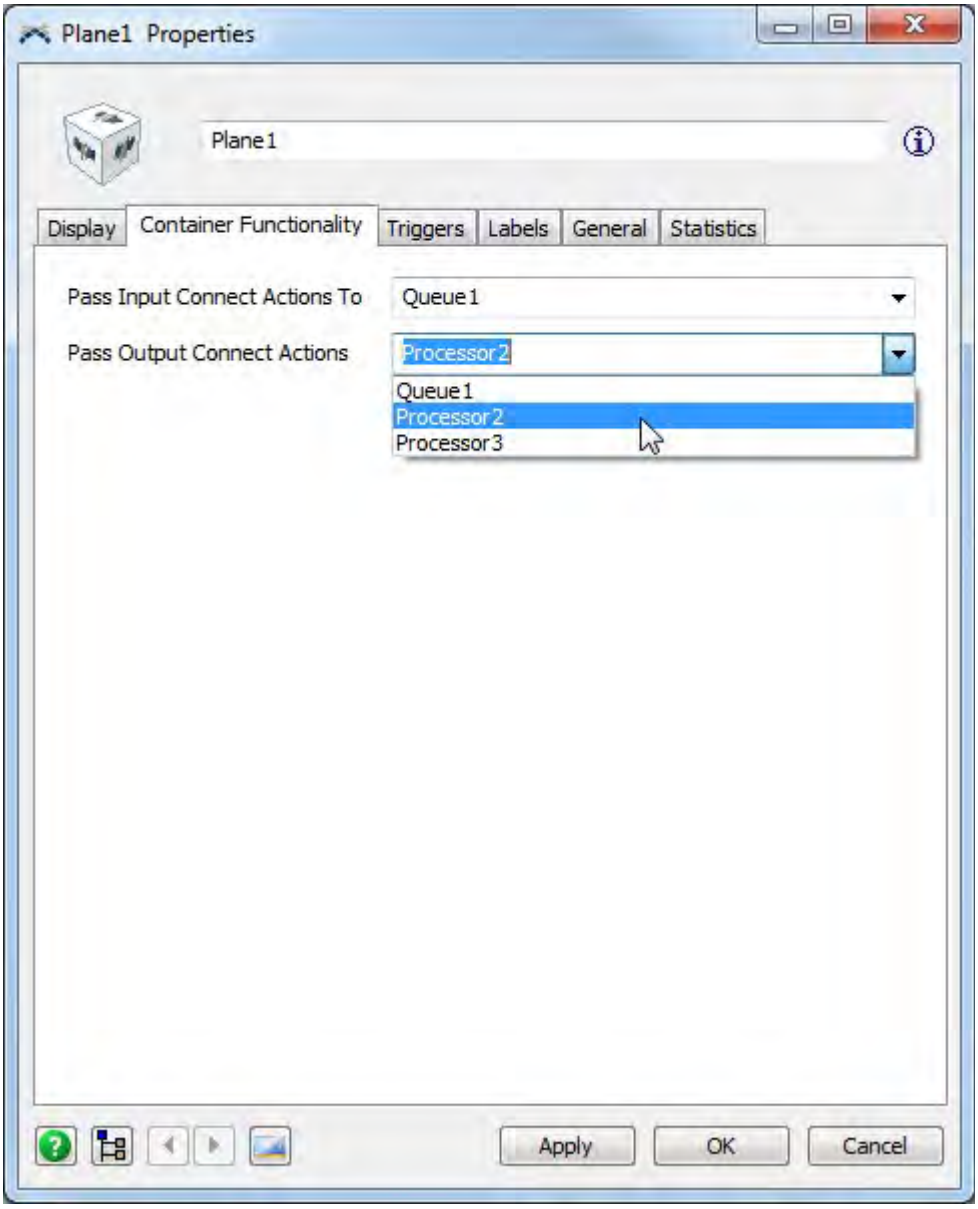




Step 8: Set the Container Functionality

At this point you now have a functioning container that holds a sub-model. You can leave the container as it is so that the contents are visible and editable, or you can hide the contents so that the container becomes a completely encapsulated black box. To hide the contents of the container, uncheck the Show Contents box on the General tab of the VisualTool properties page.

Settings for black box functionality can be found on the Container Functionality tab of the VisualTool properties page. Set the "Pass Input Connect Actions To" to the object that should be connected to the input port of the VisualTool. Set the "Pass Output Connect Actions" to the object that should be connected to the output port of the VisualTool. The internal input and output ports will be connected when an external object is connected to the container's external input or output ports.



You can also use any of the options on the Display tab to present your view of the container as a box, 3D shape, 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 > View Contents.

Fluid Library Concepts

The FlexSim Fluid Library consists of twelve 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 modeller 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 modeller'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 modeller 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 modeller 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 modeller 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 modeller to position the level indicator bar in such a way that it appears to be part of the object's 3D shape.

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 modeller defines. Once the material has been pulled into the Blender it is ready to be pulled out immediately by downstream objects. The modeller 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 Properties window 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 modeller uses to identify the material that is being pulled in. It is for the modeller'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. If there is not enough empty space in the Blender to pull at the full input rate, the Blender will lower the input rate (while still maintaining the defined percentages) to fill itself. It recommended, therefore, that the maximum content of the Blender be at least twice the input rate. This will allow the Blender to receive at the maximum input rate, as long as material is flowing out at least as fast.

Because the Blender controls how much it pulls from each port at any point in time, the modeller 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.

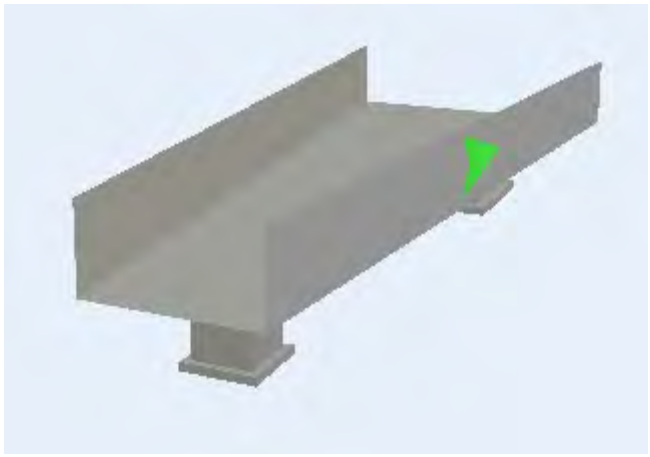
Mixing - The Blender has received material that it mixed together.

Blocked - The Blender has received material, but it cannot send it downstream.

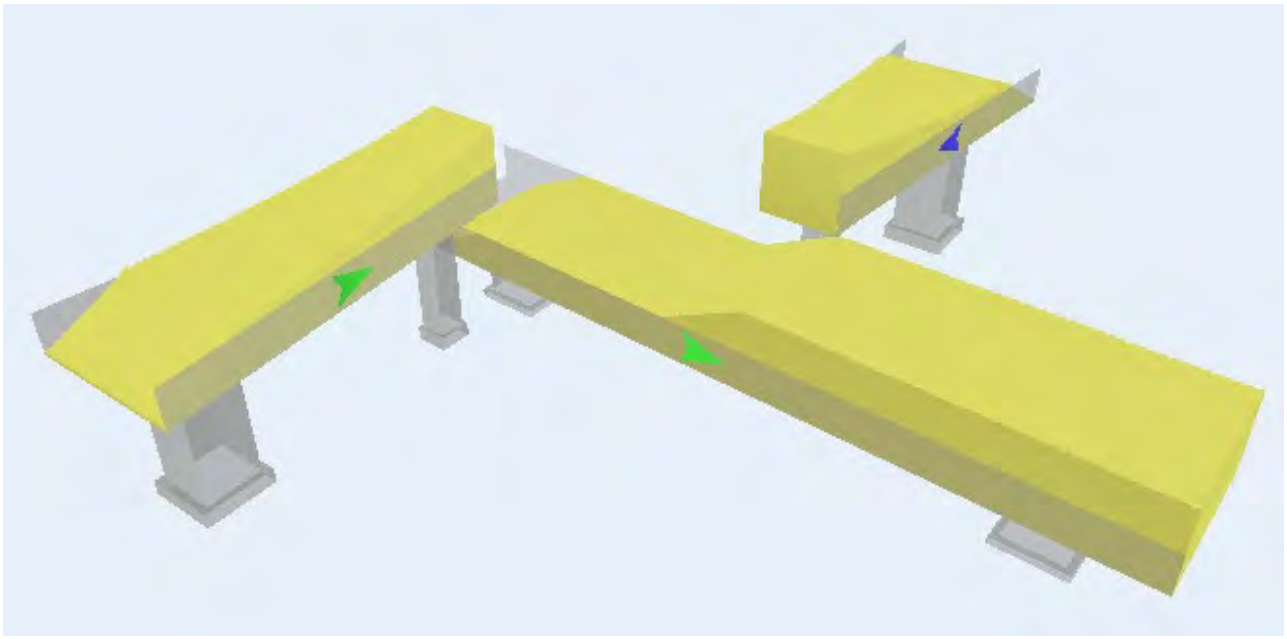
Properties pages

- Blender
- Recipe

Triggers
FluidLevelDisplay
Labels
General



The Fluid Conveyor is used to control fluid flow through the use of multiple input and output ports. Inputs and outputs may be placed anywhere along the length of the conveyor. The direction, speed, acceleration and angle of repose along with the length and width of the conveyor are all factors that will affect where material rests inside the conveyor and when and to which output port the material will be sent.



Details

Speed and Direction: The Fluid Conveyor cannot have a negative speed. If the target direction is changed, the Fluid Conveyor will decelerate to 0 speed and then accelerate to the target speed. Slices: In order to simulate a moving fluid material, the Fluid Conveyor breaks all of its content up into a series of slices defined by the user. The more slices in the conveyor, the higher the resolution. This also means there is more processing required to move and repose the material through all the slices.

Inputs: Inputs are defined over a range of slices. For every tick where material moved into the Fluid Conveyor from one of the input ports, the incoming material will be spread evenly through all of the slices that fall inside the input range.
Outputs: Outputs are defined at one point along the conveyor. Each output has a forward and reverse output percentape as well as a stopped rate. The forward and reverse percentages specify the total percentage of material that should exit through that output based on the current direction. If the output cannot handle all of the material, it is either spilled onto the floor (see Allow Spillage) or it will pass the output. The stopped rate will cause material to flow through the output when the conveyor is stopped. For example, if the stopped rate is 0.1 and the material sitting on top of an output is 0.5, the output will take 0.1 fluid units per tick, until there is no more fluid available to take.

Angle of Repose: The angle of repose defines the material's steepest angle of descent of the slope relative to the horizontal plane. This angle ranges between 0 and 90 degrees. For example, dirt has an angle of repose of 30-45°, dry sand 34°, wet sand 15-30°, and water is 0°. The fluid conveyor also has a repose rate. A value of 0 will cause the Angle of Repose to be ignored.

Repose Rate: The repose rate defines how quickly a reposing pile of material will reach its natural resting state (based on the angle of repose). The repose subroutine will be run the number of times specified in this field (the larger the number, the more processing time it mell take each tick to repose). A value between 1 and 10 is usually sufficient.

Allow Spillage: If allow spillage is checked on the Inputs/Outputs page, the fluid conveyor will try to send the full percentage of material to each output. Any material that cannot be handled by the downstream object will be spilled onto the floor. Any material that is unable to leave the conveyor through on output port that reaches the end of the fluid conveyor will also spill onto the floor. The spillage amount is recorded an a total amount of material spilled.

Sensors: Similar to the Fluid Tank's Marks, sensors can be added anywhere along the length of the Fluid Conveyor. These sensors can be set up to be triggered based on the Peak Height of a section, or on the total Volume of the section. Each sensor has a Start and an End position. This marks the range over which the sensor will look for the specific values. You can also specify the low, mid, and high values for each sensor. When fluid passes one of these values, a Sensor Trigger is fired.

States

Flowing - The Fluid Conveyor has material but is stopped. Material is settling and flowing through outputs.

Releasing - The Fluid Conveyor is sending material downstream and no material is entering.

Collecting - The Fluid Conveyor is collecting material and no material is leaving.

Not Empty - The Fluid Conveyor is stopped and no material is entering or leaving.

Empty - The Fluid Conveyor is conveying but has no material in it.

Idle - The Fluid Conveyor is stopped and there is no material in it.

Conveying - The Fluid Conveyor has material that is currently being moved along the length of the trough at a speed > 0.

Properties pages

Fluid Conveyor
Inputs/Outputs
Sensors
Triggers
Labels
General

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 modeller 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 modeller 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 modeller 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 modeller 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 modeller to change the output rates during a model run. The modeller 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.

Properties pages

Generator
Triggers
FluidLevelDisplay
Labels
General

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 modeller 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 modeller 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 modeller 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 modeller 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 modeller 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 modeller'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 modeller 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 modeller 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.

Properties pages

Mixer
Steps
Triggers
FluidLevelDisplay
Labels
General

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 modeller has to take into account the time required to move material from one point to another. It is also used if the modeller 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 modeller 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 modeller 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 modeller to edit the maximum port input and output rates, as well as the port scale factors. The modeller also has access to the AdjustInputRates and AdjustOutputRates functions. Unlike the other fluid objects, the modeller 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 modeller 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 modeller 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 modeller to state whether or not the joint between this section and the next one will be drawn. By editing this table, the modeller can make the Pipe look however they need. The modeller 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".

Properties pages

Pipe
Layout
Triggers
Labels
General

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 modeller specifies. The modeller 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 modeller 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 modeller 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.

Properties pages

FluidProcessor
Triggers

FluidLevelDisplay
Labels
General

The Splitter is used to send material to multiple output ports in percentages that the modeller 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 modeller to enter a description of each port (for the modeller'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 modeller 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.

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 Properties window 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 modeller'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 modeller 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.

Properties pages

Splitter
Percents
Triggers
FluidLevelDisplay
Labels
General

The FluidTank is a simple Fluid Object that can receive and send material at the same time. The modeller 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 modeller 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 modeller wants a constant or infinite stream of incoming material, they should use a FluidGenerator instead. The Tank has a maximum capacity that the modeller 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 modeller 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.

Properties pages

Tank
Marks

Triggers
FluidLevelDisplay
Labels
General

The FluidTerminator is used to destroy fluid material once the model is done with it.

Details

The FluidTerminator is the object that modellers 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 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 modeller 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.

Properties pages

Terminator
Triggers
FluidLevelDisplay
Labels
General

The FluidToItem 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 FluidToItem is used to convert fluid to flowitems that can be sent to any Fixed Resource. The modeller chooses the flowitem class, the default labels and name for the flowitems that are created. The modeller 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 FluidToItem 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 FluidToItem can send flowitems to any of the discrete objects and can call operators to do the transporting. The modeller can define how the input to the FluidToItem 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

Empty - The FluidToItem has no material in it.

Blocked - The FluidToItem has material in it that it is unable to send downstream. Collecting - The FluidToItem is collecting material.

Properties pages

FluidToItem
Flow
Triggers
FluidLevelDisplay
Labels

General

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 modeller 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 modeller 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 modeller 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 modeller 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 modeller. 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.

Properties pages

ItemToFluid
Flow
Triggers
FluidLevelDisplay
Labels
General

The Ticker is responsible for breaking time into small, evenly spaced units called "ticks". The modeller 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 modeller 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 modeller 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 modeller drags any Fluid Object into a model, a Ticker is created if there is not one already in the model. If the modeller 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 modeller 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. This 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 modeller 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 modeller to find the appropriate balance of speed and accuracy for their model.

The modeller 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 modeller understand what is happening in the model, they have no affect 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.

Properties pages

Ticker
Triggers
Labels
General

You can use the animation creator to create and save custom animations for FlexSim objects and Flowitems in your model. Some of the objects in the FlexSim Library come pre-programmed with some basic animations. While those animations are generally enough for most simulations, you can use the animation creator to change the animations or add new ones. Using the animation creator you can:

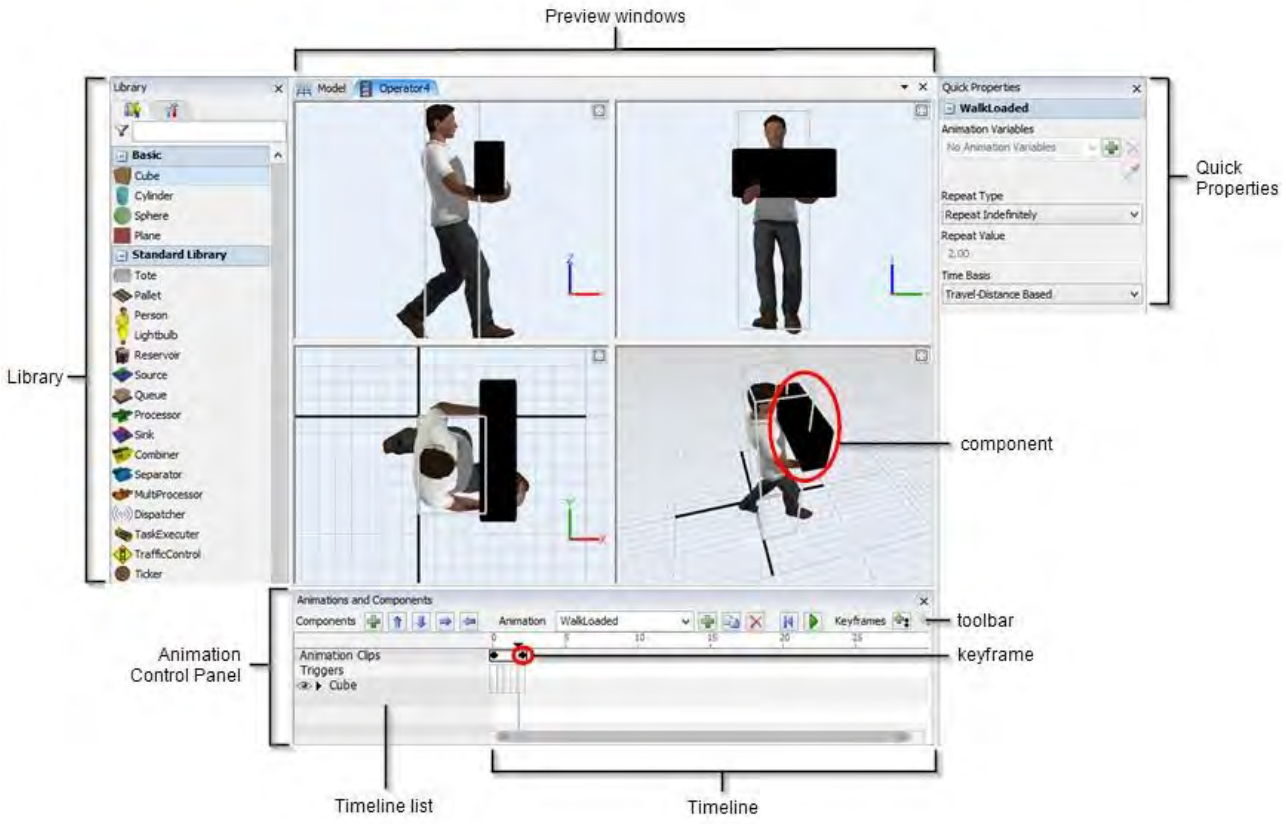
- Import and edit animation files created in other animation software applications. You can import custom objects with bone animation files from third-party software applications such as Blender, Poser, Mixamo, or 3DMax.

- Export animations created in FlexSim to other FlexSim users. Any animation files you create using the animation creator can be easily shared with other FlexSim users if they want to use that animation in a different simulation model.

- Create variables and triggers that can dynamically change how an object is animated during a simulation run or get information from an animation-related object. Using the animation creator, you can create variables that change an object's animation under certain conditions during a simulation run. For example, you could create a variable that slows down or speeds up an animation based on certain conditions. You could also use variables to change the position, rotation, or color of an object's animated components under certain conditions.

- Customize how flowitems are animated. You can use the animation creator to create surrogate objects. Surrogate objects are objects that can control the way flowitems will be animated when they are being handled by a task executer or a fixed resource. For example, you could animate a surrogate object so that it will rotate when it is inside a processor.

The following image shows the basic animation creator interface:



See The Animation Creator Interface for an explanation of each element of the interface. The remainder of this topic will explain key concepts about the animation creator.

Topics

- Opening the Animation Creator
- Key Terms and Definitions
- Creating and Importing Custom Animations
- Creating Animation Clips
- Animation Commands

Opening the Animation Creator

To open the animation creator, you first need to add the object you want to animate to the 3D simulation model. Then, right-click the object and select Edit, then Animations from the menu. The animation creator for this object will open in a separate window (tab) in the center pane.

Alternatively, you can access the animation creator by double-clicking the object. Under the General tab, in the Appearance group, click the Edit button next to the Visuals/Animations property.

Key Terms and Definitions

The following are some important terms you'll need to know in order to use the animation creator:

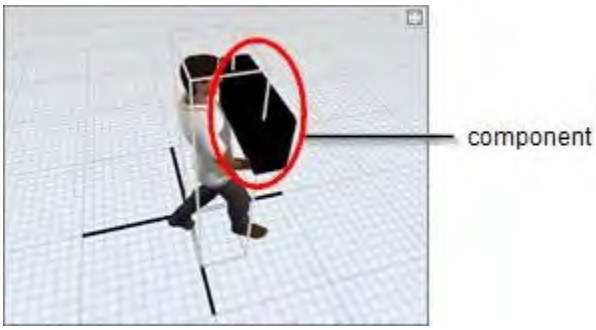
Animations

An animation is a sequence of object movements that are triggered at various points in a simulation run. You can animate any object that is capable of moving throughout the model or interacting with flowitems, such as task executers or fixed resources. In FlexSim, animations are made up of: 1) components and 2) keyframes, which are defined below.

Components

Components are 3D shapes that comprise an object. These shapes can be combined together and be positioned in such a way that looks like movement when an animation is run. The standard FlexSim objects generally come pre-programmed with at least one component, but some might have more. You can also add a custom component to a standard FlexSim object.

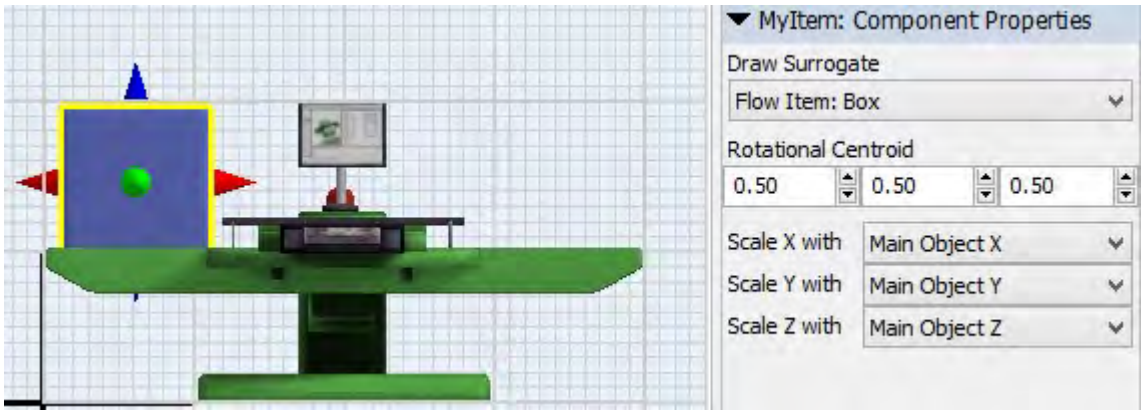
The following image shows a custom component that has been added to the Operator's Walk Loaded animation:



Surrogates

A surrogate is a special kind of component that will be replaced visually by another object (such as a flowitem) when an animation is running in the simulation model. It acts like a placeholder for another object in an animation. Whereas components are part of an animation visually (meaning it will show up in animation), a surrogate can be visually replaced by another object.

The most common reason for turning a component into a surrogate is to change the animation of a flowitem while it is being handled by one of the fixed resources or task executers during a simulation run. Once a component has been converted to a surrogate in the animation creator, it can be used to change a flowitem's position, size, or rotation in an animation while a fixed resource or task executer is handling it. When the fixed resource or task executer begins handling the flowitem, the flowitem's shape will be replaced by the shape indicated in the Draw Surrogate menu in the surrogate's Quick Properties. For example, in the following image, you'll see that a surrogate (a cube) has been added to a processor's animation:



In this example, the cube will be replaced by the flowitem box shape when it enters the processor because that is what was selected in the Draw Surrogate menu. If the Draw Surrogate menu instead had been set to None, the flowitem will change to the shape of the surrogate object (a cube) when it enters the processor. The flowitem will then adopt the same position, size, or rotation coordinates that were used in the surrogate's animation.

Surrogate shapes

The shape that is being drawn for the surrogate is not important and will not show up in the model. It is purely for your convenience while you are animating the surrogate.

Keyframes

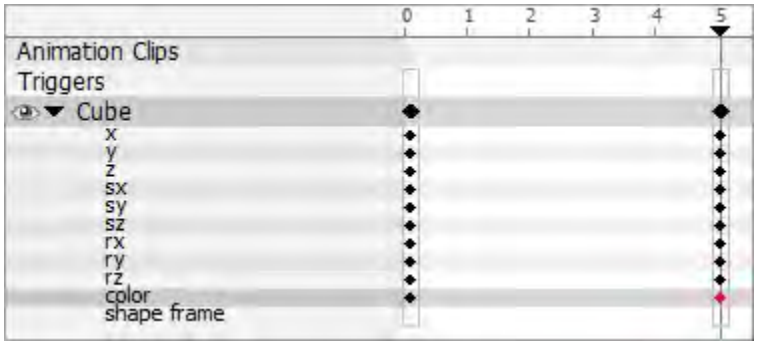
A keyframe is a drawing of an object that defines the starting and ending point of an animation segment. A component keyframe consists of a set of values that define the component's properties such as its position, size, rotation, shape, color, etc. In other words, keyframes are the saved values for different properties of a given set of components.

Usually, you will put two or more keyframes on a timeline to define the beginning and end of an animation segment. Although keyframes typically define the beginning and end of an animation, you should also put keyframes any place that marks a major movement between components. See the example in the next section (Tweening) for more information.

Tweening

Tweening is when FlexSim automatically creates the animation transitions between two keyframes. FlexSim has the capability to tween two components based on many different factors such as an object's position, rotation, size, color, etc. For example, imagine you want to animate a cube component that will change from blue to red in about 5 seconds, as illustrated in the following image:

The timeline for this kind of animation would look approximately like the following image:



Notice that in this timeline, the cube had a keyframe at time 0. On this keyframe, the cube's color component is set to blue. There is a second keyframe at time 5 where the color is set to red. While the animation is running, FlexSim will automatically tween the colors, which means that it will create all the transitional colors between time 0 and 5. In other words, you don't have to manually create a purple color halfway through the animation because FlexSim will do it for you when it tweens the two keyframes.

As a general rule, you should try to put keyframes in places that mark the begining and end of a major movement. For example, let's say you wanted to animate an Operator walking.The first keyframe should be the position the Operator starts from (perhaps a standing position) and the next keyframe might be the right leg extended and so forth. FlexSim will then automatically generate a smooth transition between the keyframes to create the illusion of leg movement during a simulation run.

Timing Animations

The amount of time between the keyframes on the timeline will determine how quick or slow the transition will occur.

Animation Variables

Animation variables can possibly change the way objects are animated based on conditions that might change during a simulation run. For example, a variable on a component keyframe can change a component's position, rotation, size, color, shape, etc. based on dynamic conditions in the simulation model. You could also use a variable to determine how fast a particular animation should run based on certain conditions.

Keep in mind that an animation variable is basically a reference point to an animation-related object, component, or keyframe. Animation variables can point to:

- Components
- Surrogates
- Keyframe times
- Time gaps
- Component keyframes (such as the component's position, size, rotation, color, or shape frame)

Once you've created an animation variable, you can reference that variable in a Flexscript command. (See Animation Commands for more information.) Flexscript commands can either get information about a variable or it can set the value of a variable (thereby possibly changing the variable). For example, if you want to dynamically change the length of your animation based on parameters in your model at run time, you can create an animation variable that points to a time gap, and then set that animation variable through a keyframe trigger or from a trigger in your model.

Component variables are read-only

Animation variables pointing to a component are static variables. They are merely available as a reference point to a component through the getanimationvar command. Calling setanimationvar to a variable pointing to a component will have no effect. If you want to dynamically set a component's property, use a variable on a component keyframe instead.

Keyframe Triggers

Triggers can be added to animations that will fire when the animation gets to a specific point in the animation's timeline. They are added through the timeline. Triggers allow you to dynamically change or update animation variables, stop the animation, change some parameter in your model, or execute any other Flexscript code.

Creating and Importing Custom Animations

You can create your own custom 3D objects and animations using third-party software applications such as Blender, Poser, Mixamo, or 3DMax. To import a custom object and animation:

1. First you need to create a custom object to act as your basic template. Drag either the Basic FR object or the Basic TE from the Library into your simulation model. (Use the Basic FR object if you want to create a fixed resource or the Basic TE if you want to create a task executer.)
2. Click the Basic FR object or the Basic TE object to select it.
3. In Quick Properties under General Properties, click the arrow next to the Shape box to open a menu.
4. Select Browse from the menu and navigate to the location of your custom 3D object's file. Select the file that you want to import. The Basic FR object or the Basic TE will be updated to the new shape that you imported.

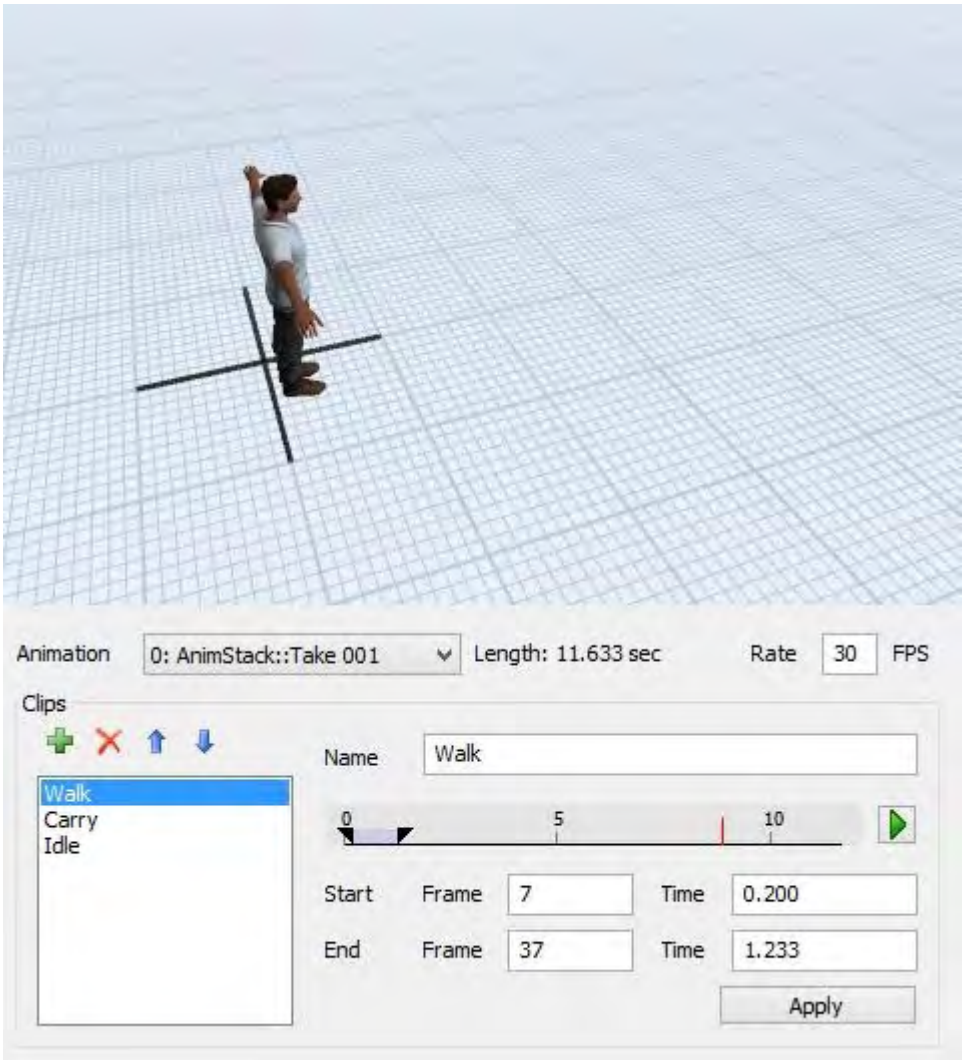
When you import an object, it will also import any animations that are associated with it. In order to create custom animations, you might need to create animation clips for the object. See the next section about Creating Animation Clips for more information.

Creating Animation Clips

When you create a custom object using third party software, you might typically want to create more than one animation for it. For example, the default Operator object in FlexSim has three different animations: walking, walking while carrying a load, and standing. So, if you wanted to make your own custom Operator object, it should probably have at least these three same animations.

However, some animation software programs only allow you to export all these animations as a single animation file. Consequently, when you import these files into FlexSim, you'll need to cut the animation file into individual clips using the clip editor in the animation creator. To create animation clips:

1. With the animation editor open for the object you want to animated, click the Add Clip button on the animation toolbar and select Add Animation Clip from the menu. This will open the clip editor, as shown in the following image:



2. Click the Add button to add a new clip to the list of clips.
3. Give the clip an appropriate name by typing a new name in the Name box.
4. If needed, press the Play button to preview the full animation.
5. Now you'll need to choose which section of the animation should be included in this clip. Drag the sliders on the timeline to the beginning and end of the section you want to include in this clip. Alternatively, you could manually set the exact frames and times that should be included using the Frame and Time boxes below the timeline.
6. If needed, you can also change the rate of frames per second using the Rate box.
7. Click the Apply button to save the changes to the clip.
8. Make any other necessary changes and then close the window.
9. Now when you click the Animation menu in the animation control panel, the new clip you created will appear in the list of available animations.

Use the animation creator's timeline if you want to combine one or more clips together.

Animation Commands

The following commands deal with animations:

startanimation(object, animation) - Starts an animation on the object. Animation can be either a string value that is the name of the animation, or it can be a number which is the animation rank.

stopanimation(object, animation) - Stops an animation on the object. Animation can be either a string value that is the name of the animation, or it can be a number which is the animation rank.

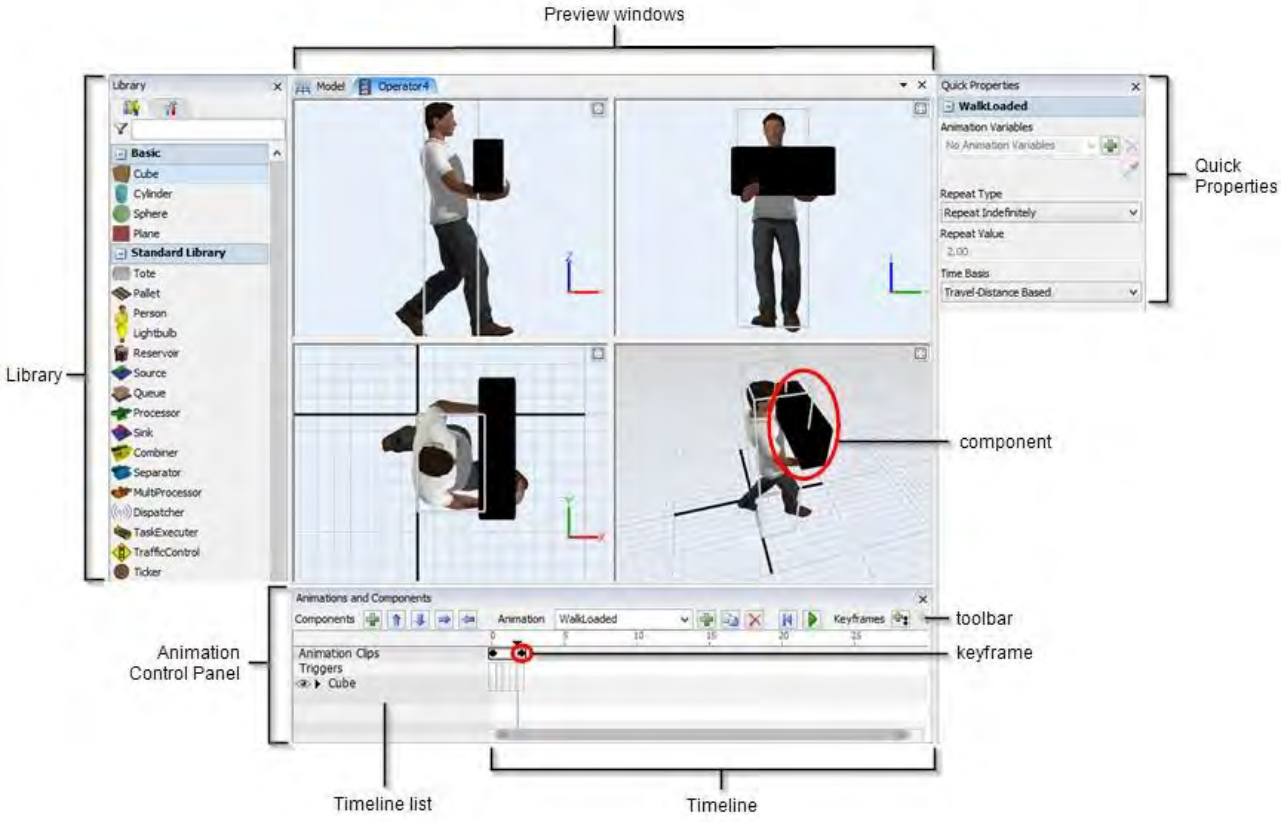
resumeanimation(object, animation) - Resumes an animation on the object that was previously stopped. Animation can be either a string value that is the name of the animation, or it can be a number which is the animation rank.

getanimationvar(object, animation, "varname") - Returns the value of an animation variable.

setanimationvar(object, animation, "varname", value) - Sets the value of an animation variable. Value can be a number or object and is based upon what the animation variable is linked to.

The Animation Creator Interface

This topic will provide an in-depth overview of the Animation Creator UI. The most important elements of the animation creator are labeled in the following image:



Note that in the preceding image, the toolbar is slightly truncated. See Toolbar for an explanation of the full toolbar accompanied by complete images.

Each element will be described in more detail in the following sections.

Preview Windows

Located in the upper portion of the animation creator, the preview windows allow you to see how your objects will appear visually during the animation. There are four different panes in this window that allow you to see the animation from various angles.

You'll be able to see the object's animation in action when you run an animation using the animation control bar. You can also scrub the timeline (which means to put the cursor at a particular place in the animation timeline) to see how the object will look at specific points in the animation.

If you add any components to the animation, you can also use the preview windows to manually resize, reposition, or rotate the component object. You can also zoom in and out or change the angle the same way you would in the 3D model.

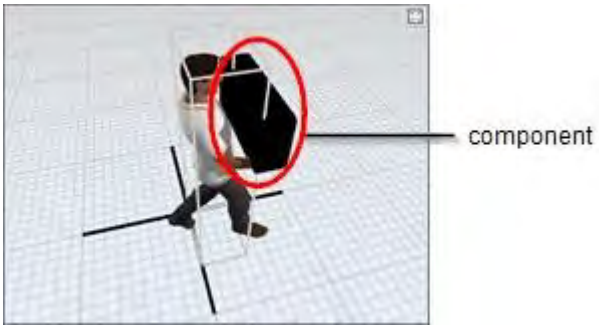
In the top right corner of each preview pane, you can click the Maximize button to hide the other 3D windows and expand the selected 3D window to take up the entire preview window. Click the button a second time to return to the default view.

If you right-click anywhere inside a blank spot of the preview window, a menu will pop up with one option: Flip Axis. This option flips the axis of the view. For example, if you flip the axis that is currently displaying the top view, it will display the bottom view. This option is only available in the side, front, and top 3D views.

Components

Components are 3D shapes that comprise an object. These shapes can be combined together and be positioned in such a way that looks like movement during a simulation run. Any object can possibly have one or more components or you can add a component to a standard FlexSim object.

The following image shows a custom component that has been added to the Operator's Walk Loaded animation:



Best practices when editing components

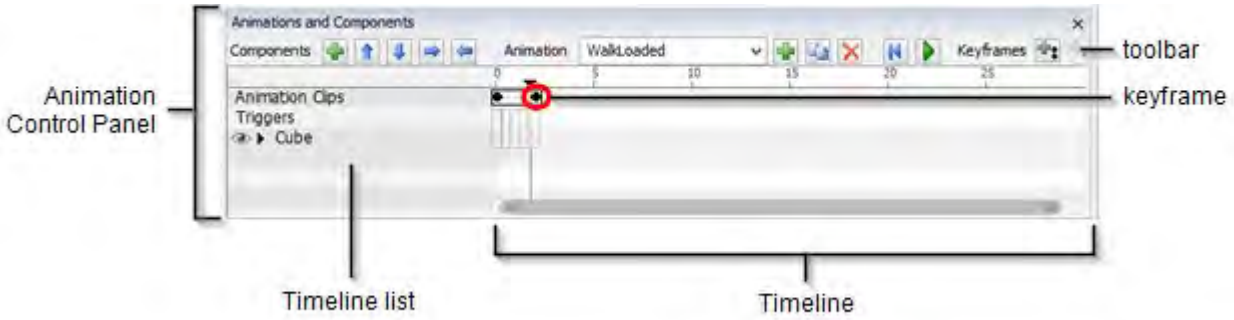
When making changes to components, it is usually wise to start from the outside (parent components), and work your way through all of the child components. If you ever make a mistake, click the Base Postions button to retrn the animation to its default position.

Library

When the animation creator is open, the Library will change to display some basic objects that can be added as components in the animation.

Animation Control Panel

Located in the lower portion of the animation creator, the animation control panel is the primary workspace you'll use while working inside the animation creator, as shown in the following image:



Note that in the preceding image, the toolbar is slightly truncated. See Toolbar for an explanation of the full toolbar accompanied by complete images.

The following sections will explain the different elements of the animation control panel.

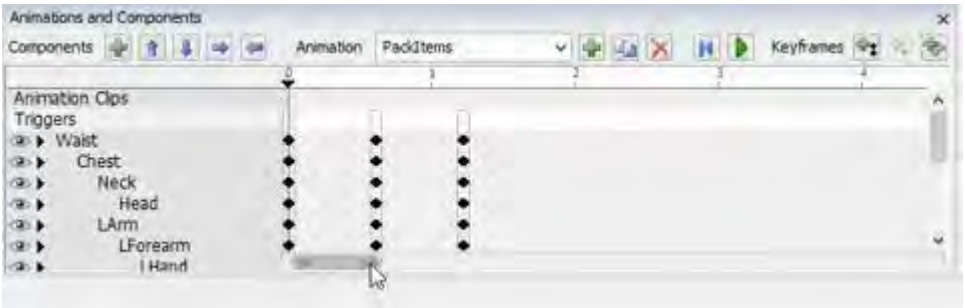
Timeline

The timeline is a chronological display of the animation over time. You can also scrub the timeline (which means to put the cursor at a particular place in the animation timeline) to see how the object will look at specific points in the animation.

You can also drag keyframes along the timeline to make them longer. As the space increases between keyframes on the timeline, the tweening animation will take longer.

If you want to zoom in or out of the timeline, you can use either one of the following methods:

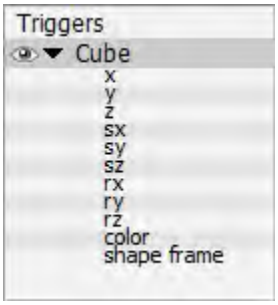
- Click the Zoom button on the toolbar to zoom in or out on the timeline to make all the keyframes fit on the screen.
- Position the mouse above the timeline and press the Ctrl key while scrolling the mouse wheel.
- Click the right end of the scroll bar underneath the timeline and drag it to zoom in and out, as shown in the following image:



Timeline List

The timeline list is kind of like a header for the timeline, showing you each element in the animation in the timeline. The timeline list displays any animation clips, triggers, and/or components that have been added to this object's animation.

If you clck on the arrow next to a component, you can expand the list to show the component's individual properties on the timeline, as shown in the following image:



Once the component properties have been expanded, you can add component keyframes to the timeline in order to change the way a particular component property will behave at various points in the animation.

Keyframes

A keyframe is a drawing of an object that defines the starting and ending point of an animation segment. In the user interface, keyframes look like a small black diamond that turns red when it is selected. A component keyframe consists of a set of values that define the component's properties such as its position, size, rotation, shape, color, etc. In other words, keyframes are the saved values for different properties of a given set of components.

You will put keyframes on a timeline to define the beginning and end of an animation segment. Although keyframes typically define the beginning and end of an animation, you should also put keyframes any place that marks a major movement. See Keyframes and Tweening for more information.

Toolbar

The toolbar contains all the tools you'll need to create and edit custom animations:

Component tools

Used to add or reorder new components. (See Components for more information.) All components will be listed in the timeline list. You can also add a component by dragging an object directly from the Library into the preview window. It will also show up in the timeline list after it has been added. The following table describes each button:

Button	Description

	Adds a new component. To remove a component, select a component from the list and hit the Delete key.
	Moves the selected component up or down in the list.
	Moves the selected component left or right in the list. This creates parent/child relationships with the components. When a parent component's position or rotation is moved, all of its child components will also have their position/rotation moved with respect to their parent.

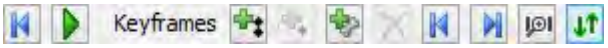
Animation tools

Used to select, add, copy, or delete new animations. To edit a specific animation, make sure that you select it from the Animation menu first. The selected animation will be loaded into the timeline. The following table describes each button:

Button	Description
	Adds a new animation to the object.
	Duplicates the current animation.

	Removes the current animation from the object.

Keyframe and Timeline tools



Used to control playback of the animation preview and to add or edit keyframes on the timeline. The following table describes each button:

Button	Description
	Moves the current time cursor to the beginning of the animation (time 0).
	Plays and pauses the animation. When you press the Play button, it will turn into the Stop button, which can be used to pause the animation. Animations will automatically loop playback.
	Adds a new keyframe to the animation for all components.
	Adds a new keyframe to the animation for the selected component only.
	Opens the clip editor tool. See Creating Animation Clips for more information.

	Removes the selected keyframe(s).
	Moves the time cursor to the previous or next keyframe.
	Zooms the timeline in or out to make all keyframes fit on the screen.
	Sync the 3D Views. When you click this button, the 3D views will update the object to display the current values of all its components as you move the time cursor in the timeline.

Base Position tools

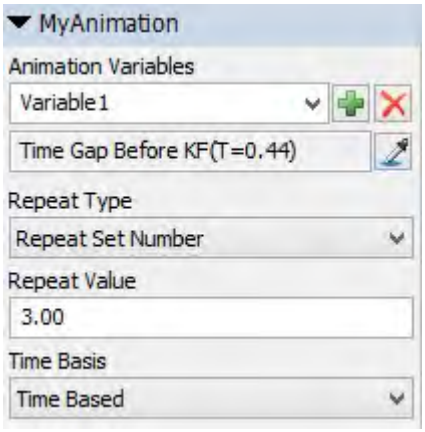
Used to save the base position of the components. You can also return to the base position at any time using these tools. The following table describes each button:

Button	Description
	Saves the current values of all the object's components as the base position. This is how the object will be displayed in the model when the model is reset.

	Updates the 3D views to display the object in its currently saved base positions. Clicking this button does not affect the timeline.
	Removes the saved base positions.

Quick Properties

You can use the settings in Quick Properties to change the basic functionality of the animation, such as its repeat type and time base. You can also use Quick Properties to create new animation variables. The following image shows the available animation settings:



The following sections will explain each setting.

Animation Variables

An animation variable is basically a reference point to an animation-related object, component, or keyframe. See Animation Variables for more in-depth information.

Select a variable you want to edit from the Animation Variable menu. Click the Add button next to the Animation Variable to add a new variable and the Delete button to remove the selected variable.

Use the Sampler button to select the object, keyframe, or timeline position the variable should point to. Animation variables can point to:

- Components
- Surrogates
- Keyframe times
- Time gaps

- Component keyframes (such as the component's position, size, rotation, color, or shape frame) After you've selected an object, it will be listed in the box below the Animation Variable menu.

Repeat Type

Animations can be repeated using the following types:

- Do Not Repeat - The animation will stop as soon as it is complete.
- Repeat Indefinitely - Causes the animation to loop back to the beginning forever until the animation is stopped in the model.
- Repeat Set Number - Causes the animation to loop back to the beginning based on the number of times listed in the Repeat Value box.
- Repeat After Time -This will cause the animation to loop back to the beginning after the amount of time (or distance) listed in the Repeat Value box regardless of whether or not the animation has completely. For example, an animation that is 10 seconds long that has a repeat time of 5 seconds will only play half of its animation.
- Time After Animation End - Once the animation has ended, the animation will wait the amount of time (or distance) listed in the Repeat Value box before starting again from the beginning.

Repeat Value

By default, the Repeat Value box is not available for use. It will become available when you select an option from the Repeat Type menu that requires a repeat value. It allows you to enter in a value for how frequently the animation should be repeated.

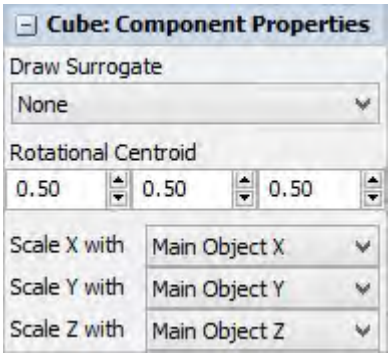
Time Basis

Use the Time Basis menu to determine how the speed of the animation will be calculated. This menu lists the following options:

- Time Based - When an animation is set as Time Based, the animation will play based on the simulation run speed. The numbers displayed on the timeline are in model time units (as defined in your Model Settings window).
- Travel-Distance Based - When an animation is set as Travel-Distance Based, the animation will play as the object moves in 3D space. The numbers displayed on the timeline are model length units (as defined in your Model Settings window), rather than time. An example of a Travel-Distance Based animation is the Operator's Walk animation.

Component Properties

When you add a component to an animation and select that component in the preview windows, a variety of settings will become available in Quick Properties. Some general properties will appear that will allow you to adjust the component's size, rotation, color, shape, etc. Additionally, the Component Properties propertie group will appear, which is shown in the following image:



This group has the following settings:

- Draw Surrogate - This menu can change the currently selected component into a surrogate. A surrogate is a special kind of component that will be replaced visually by another object (such as a flowitem) when an animation is running in the simulation model. It acts like a placeholder for another object in an animation, usually a flowitem. (See Surrogates for more information.) If you select None from this menu, the component will not act as a surrogate. The rest of the menu lists different flowitem shapes that you can select to replace the surrogate during the animation. When the animation runs, the surrogate object will be replaced by the shape you selected.

- Rotational Centroid - Sets the surrogate's rotation.
- Scale X with - Determines the coordinates on the main object that will correlate to the surrogate's X coordinates.

- Scale Y with - Determines the coordinates on the main object that will correlate to the surrogate's Y coordinates.

- Scale Z with - Determines the coordinates on the main object that will correlate to the surrogate's Z coordinates.

Surrogates require animation variables
When you create a surrogate, you will be prompted to create an animation variable that points to that surrogate. This animation variable can be set to an object in the model that will be drawn in place of this component.

Tutorial - Basic Animation Creator Concepts

This tutorial is designed to give you examples of how the animation creator will work in context. In this tutorial, you'll learn the basics of the animation creator. After completing this tutorial, you might possibly be interested in Advanced Animation Creator Concepts tutorial.

In this tutorial, you will create a simple animation that will make an Operator appear to operate a Processor when it receives a flowitem.

Tasks Covered

This tutorial will include the following tasks:

1. Opening the animation creator
2. Creating a new animation
3. Editing components
4. Adding keyframes
5. Changing the animation speed and repeat settings
6. Triggering an animation in a simulation model

For More Information

This tutorial will provide hands-on experience with the animation creator. But if you would like more detailed information about this tool, please refer to:

- Animation Creator Key Concepts
- The Animation Creator Interface

Basic Animation Creator Tutorial

Complete the following steps:

Step 1 - Open the Animation Creator

In this step you will open the animation creator for an Operator object. (See Opening the Animation Creator for more information.) For the purposes of this tutorial, you will use the "Legacy" Operator that was the default Operator in older versions of FlexSim. The Legacy Operator has more customizeable animations and will better illustrate some of the basic animation concepts.

1. With the 3D model view open, drag an Operator from the Library into the simulation model.

2. Double-click on the Operator to open its Properties dialog box.

3. With the Operator tab open, open the Shape menu and select Legacy Operator to use the older version of the Operator.

4. Click the Apply button to save the changes.
5. In the General tab, click the Edit button next to the Visuals/Animations property to open the animation creator for this operator.

6. If needed, press the OK button to close the properties window.

Step 2 - Create a New Animation

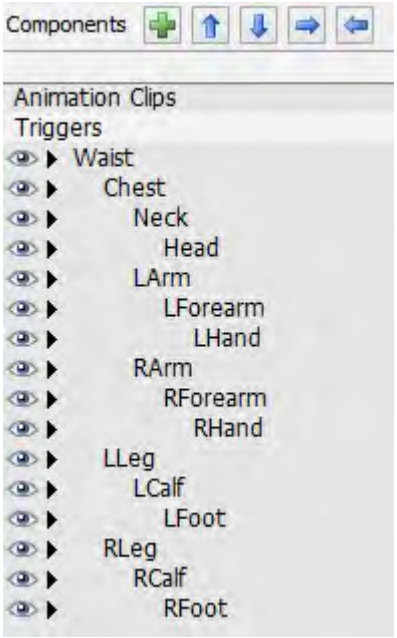
In this scenario, you'll to create an animation of an Operator manipulating the controls on a processor. You first need to add an additional animation to the Operator's list of animations.

1. On the animation toolbar, find the Animation box and click the Add button next to it to add a new animation.

2. The Animation box should now read *Animation4*. Click inside the box to highlight the text and change it to *OperateProcessor* for easier reference.

Step 3 - Edit Components

Next, you'll edit some components to create the initial starting position for the animation. When you look at the timeline list, you'll notice in the Animations and Components window that there is already multiple components that make up the Operator (such as the Operator's waist, chest, etc.), as shown in the following image:



Components that are children of other components (indicated by the indentation) are subject to changes made by their parent component. For example, if you changed the position of the Operator's left arm (LArm), it would also change the position of the left forearm (LForearm) and the left hand (LHand). In other words, if you moved the left arm component, all of is child components would move along with it.

In this step, you will change the rotation of the left and right forearms so that they appear to be extended.

1. Click the Step Left button button to make sure that your cursor at the beginning of the timeline.

2. In the timeline list, click LForearm (the left forearm commponent) to select it. In Quick Properties, under General Properties, change the X-rotation to -90. The left forearm will now appear to be extended.
3. Now you'll do the same to the right forearm. In the timeline list, click RForearm to select it. In Quick Properties, change the X-rotation to -90.

The following image shows how the Operator should look in the preview window after completing this step:

Step 4 - Add Keyframes

Now that the starting position is set, you'll add some keyframes to the animation to mark the major movements in the animation. Then you'll change the middle keyframe so that the Operator's arms will be in a different position.

1. Click the Add Keyframe button button to add a keyframe to time 0. It should appear as a series of black diamonds in the timeline. You'll notice that the cursor on the timeline jumped ahead one second. This allows you to quickly make adjustments and add keyframes all at once without overwriting any changes. You will adjust the timing of the keyframes later in the tutorial.

2. You want to make sure our animation is seamless when it repeats. The easiest way to do this is to create another keyframe, a duplicate of the first. Click the Add Keyframe button again to add another keyframe.

3. Repeat this step so that there are now three identical keyframes.

4. Now you'll change the second (middle) keyframe so that the left and right forearms are extended. Click the second keyframe for the LForearm (left forearm). The keyframe will turn red to indicate it is selected. In Quick Properties, under General Properties, change the X-rotation to -75.

5. With the LForearm still selected, change the Z-rotation to 35 in Quick Properties.

6. Now you'll make similar (but not exactly identical) changes to the right forearm. Click on the second keyframe for the RForearm. In Quick Properties, under General Properties and change the Xrotation to -75.

7. The Z-rotation of the right forearm will be slightly different. With the RForearm still selected, change the Z-rotation to -35 in Quick Properties.

When you preview the animation by pressing the Play button on the toolbar, the Operator will appear to open and close its arms, as shown in the following image:

Step 5 - Change the Animation Speed and Repeat Settings

In this step, you will make the animation go a little bit faster. Then, you will adjust the animation's repeat settngs so that it will only be repeated twice.

1. You'll start by zooming in on the timeline. There are several ways to zoom in and out from the timeline. (See Timeline for more information.) For now, click the Zoom button on the toolbar to zoom in on the three keyframes you have created so far.

2. Now that you have zoomed in, you will change the pace of the second and third keyframes. Click anywhere on the second keyframe and drag it to 0.50 on the timeline.

3. Click anywhere on the third keyframe and drag it to 1.00 on the timeline.

4. Now you'll want to make sure the animation repeats twice when a flowitem enters a processor. In Quick Properties, under the OperateProcessor group, click the Repeat Type menu and select Repeat Set Number.

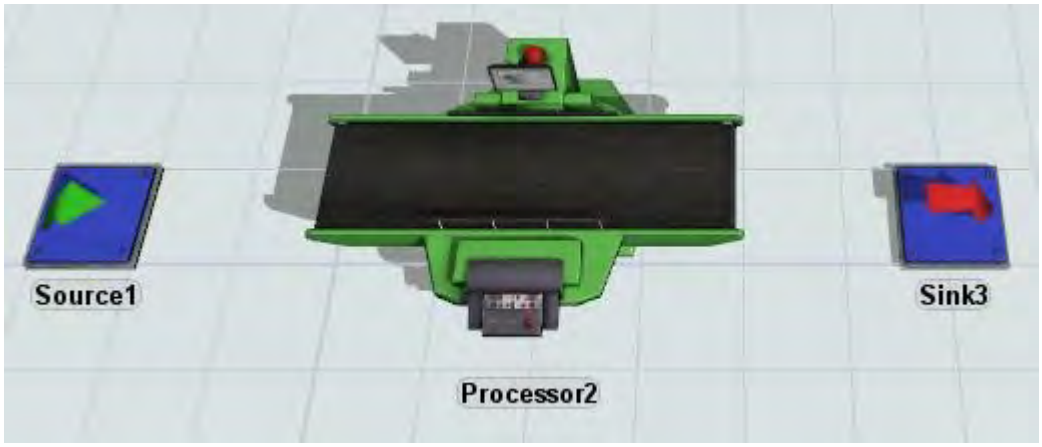
5. Type 2 in the Repeat Value box.

Step 6 - Trigger an Animation in a Simulation Model

In the last step, you'll create a simple simulation model and call the Operator's newly created *OperateProcessor* animation.

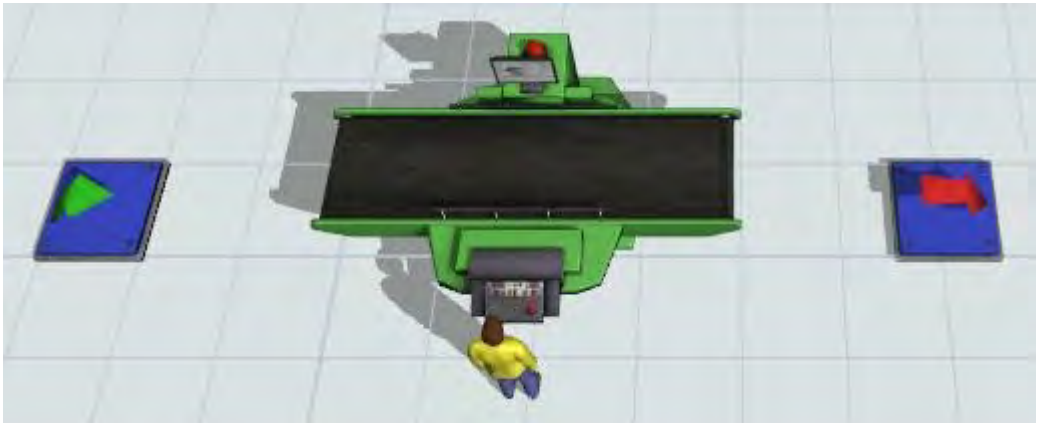
1. Return to the 3D model by clicking the Model tab on the center pane.

2. From the library, drag out a Source, a Processor, and a Sink. Move them in the model so they are in the approximate positions shown in the following image:



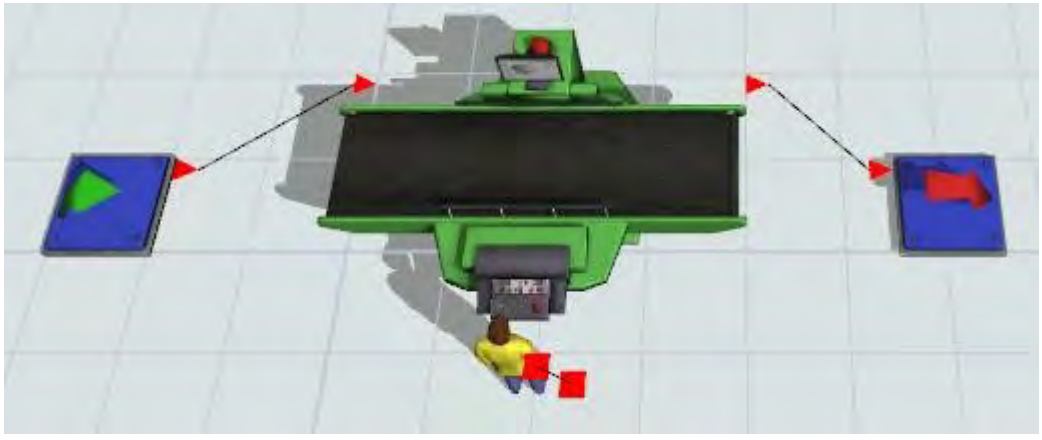
3. Move the Operator so that it is by the control panel on the Processor.

4. With the Operator still selected, in Quick Properties under General Properties, change the Z-rotation ↺to 90 so that the Operator is facing the Processor, as shown in the following image:



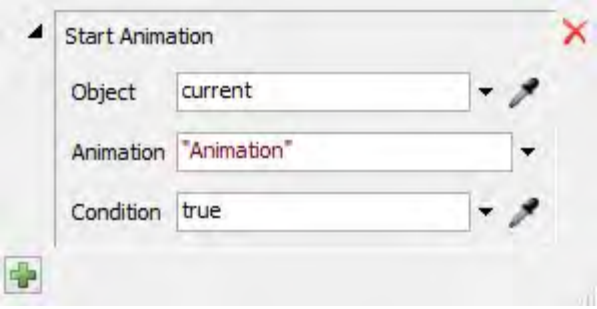
5. Next you'll need to connect the objects in the model. Make 'A' connections between the Source and the Processor and between the Processor and the Sink.

6. Make an 'S' connection between the Processor and the Operator, as shown in the following image:



7. Now you'll set up some triggers on the processor that will start and stop the animation when a flowitem enters the processor. Double-click the Processor to open its Properties dialog box. You might want to move the dialog box so that the Operator is still visible in the simulation model.
8. In the Triggers tab, click the Add button next to the OnEntry trigger to open a menu.

9. Select Visual then Start Animation from the menu to open up the Start Animation picklist options, as shown in the following image:



10. Click the Sampler button next to the Object box to enter sampler mode.

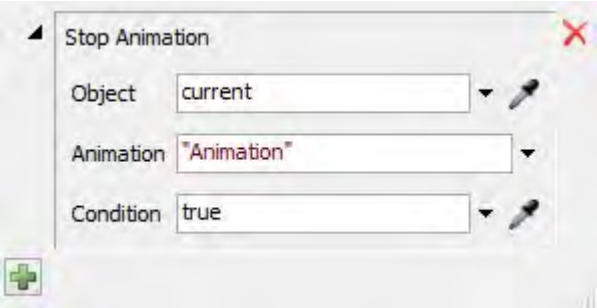
11. Click the Operator in the simulation model to sample it. The Object box should now read centerobject(current, 1), which indicates that the object that will be animated is the one connected to the Processor's first center port.

12. Click the arrow next to the Animation box to open a menu of the Operator's available animations. Select OperateProcessor animation.

13. Click somewhere outside of these boxes to save the changes.

14. Now click the Add button next to the OnExit trigger to open a menu.

15. Select Visual then Stop Animation from the menu to open up the Stop Animation picklist options, as shown in the following image:



16. Click the Sampler button next to the Object box to enter sampler mode.

17. Click the Operator in the simulation model to sample it. The Object box should now read centerobject(current, 1).

18. Click the arrow next to the Animation box to open a menu of the Operator's available animations. Select OperateProcessor animation.

19. Click somewhere outside of these boxes to save the changes.

20. Click the OK button to save the changes and close the Properties dialog box.

21. If you haven't already done so, save this model so that you can use it in the next tutorial.

Starting Animations with FlexScript

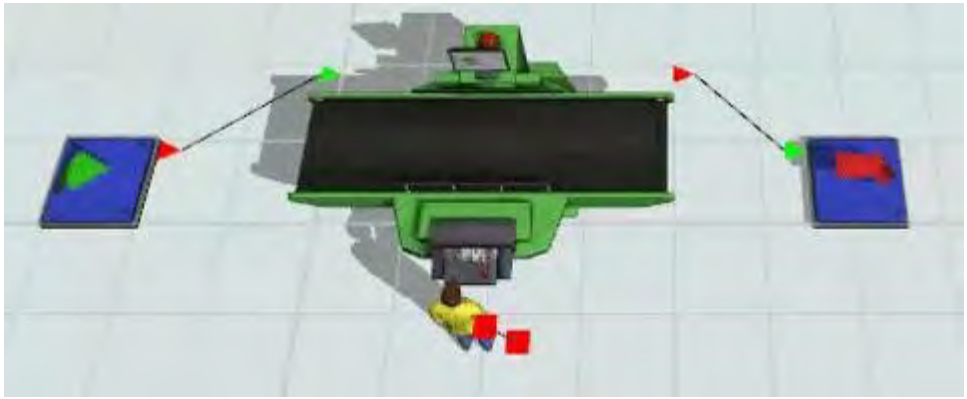
If you want to create custom codes that start an animation, you can use the following Flexscript code:

```
startanimation(operator, "OperateProcessor");
```

To stop the animation call:

```
stopanimation(operator, "OperateProcessor");
```

You can now test your animation by running your simulation model. Click the Reset and Run buttons on the simulation control panel. When your simulation runs, the Operator's OperateProcessor animation should run twice every time a flowitem enters the Processor, as shown in the following image:



This completes this the Basic Animation Creator Concepts tutorial. After completing this tutorial, you might possibly be interested in Advanced Animation Creator Concepts tutorial.

Tutorial - Advanced Animation Creator Concepts

This tutorial is designed to give you examples of how the animation creator will work in context. In this tutorial, you'll learn some of the advanced concepts of the animation creator. Before completing this tutorial, you should consider completing the Basic Animation Creator Concepts tutorial.

In this tutorial, you will create a surrogate item for flowitems that will enter a Processor. Using this surrogate item, you'll make the items appear to rotate as they move through the Processor.

Tasks Covered

This tutorial will include the following tasks:

1. Creating a basic model
2. Adding a component
3. Animating the component
4. Creating a surrogate and animation variable
5. Triggering an animation in a simulation model

For More Information

This tutorial will provide hands-on experience with the animation creator. But if you would like more detailed information about this tool, please refer to:

- Animation Creator Key Concepts
- The Animation Creator Interface

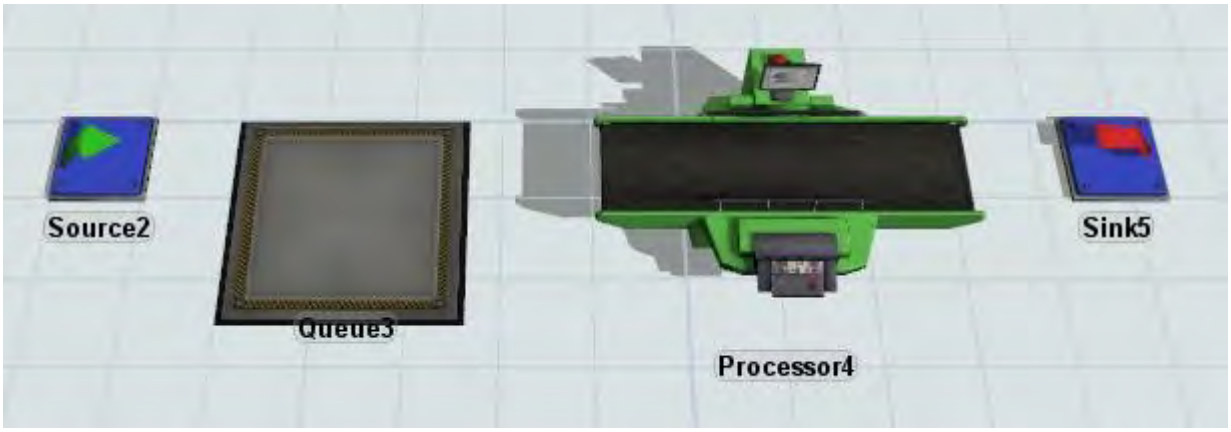
Advanced Animation Creator Tutorial

Complete the following steps:

Step 1 - Create a Basic Model

You'll start by creating a simple model that will have three different types of flowitems that will each have three different colors.

1. With a blank model open, drag out a Source, a Queue, a Processor, and a Sink. Move them in the model so they are in the approximate positions shown in the following image:



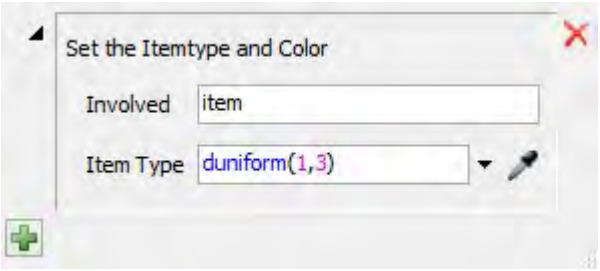
2. Next you'll need to connect the objects in the model. Make 'A' connections between:

   o The Source and the Queue o The Queue and the Processor o The Processor and the Sink

3. Double-click the Source to open its Properties dialog box.

4. In the Triggers tab, click the Add button next to the OnCreation trigger to open a menu.

5. Select Data then Set Item Type and Color from the menu to open up the Set Item Type and Color picklist options, as shown in the following image:



6. You can just use the default settings, which randomly creates 3 different flowitem types and assigns it one of three colors based on its item type.

You can now test your animation by running your simulation model. Click the Reset and Run buttons on the simulation control panel. When your simulation runs, you'll see flowitems with three different colors moving through the model.

Step 2 - Add a Component

Now you're going to create a component on the Processor that will eventually be used as a surrogate. A surrogate is a special kind of component that will be replaced visually by another object (such as a flowitem) when an animation is running in the simulation model. It acts like a placeholder for another object in an animation.

1. Right-click on the Processor and select Edit, then Animations to open the animation creator.

2. On the animation toolbar, find the Add Component button to add a new component to the Processor animation.
3. Make sure the new component is selected. In Quick Properties, under General Properties, change the component's name to *MyItem* (in the top-most box).

4. Now you'll change the component so that it is the same shape as the default flowitem (a box). Change the X-size to 0.61, the Y-size to 0.61, and the Z-size to 0.30. NOTE: If you were to look at the dimensions of a box in the Flowitem Bin, it would have these settings.

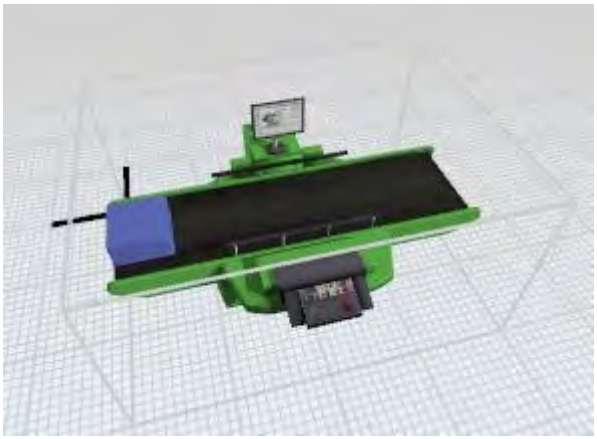
5. Now you'll add an animation to the Processor object. On the animation toolbar, find the Animation box and click the Add button next to it to add a new animation.

6. The Animation box should now read *Animation1*. Click inside the box to highlight the text and change it to *ProcessItems* for easier reference.

Step 3 - Animate the Component

Now you'll animate the MyItem component so that it will rotate as it moves through the Processor. You'll set the initial position of the MyItem component. Then, you'll add two keyframes and change the position and rotation of the MyItem component in the two different keyframes.

1. Using the preview windows, move the component so that it is on the beginnging of the Processor, as shown in the following image:



2. Click the Add Keyframe button to add a keyframe at time 0.

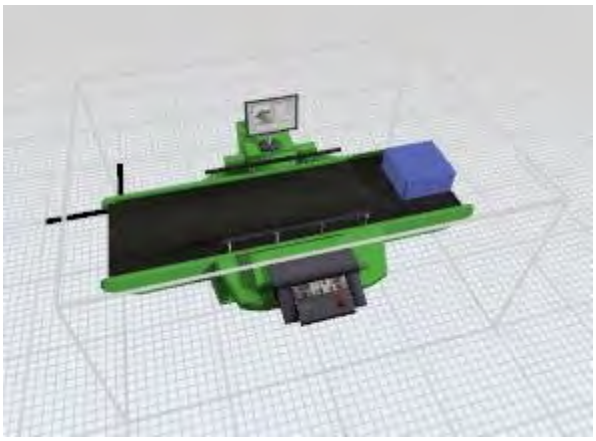
3. Repeat this step to add a second keyframe.

4. Click the second keyframe and drag it approximately to time 8 on the timeline.

5. With the keyframes in place, you will now change the component's animation. Click the arrow next to MyItem in the timeline header to expand the list of properties for the MyItem component, as shown in the following image:

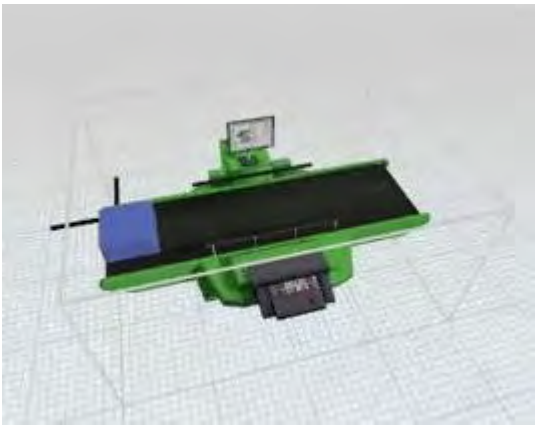
6. Click the second keyframe so that all the keyframes turn red. Check to make sure the component is selected in the preview windows as well.

7. Using the preview windows, move the component so that it is at the other end of the Processor, as shown in the following image:



8. In Quick Properties, under General Properties, change the component's Z-rotation to 360.

Now when the animation runs, the MyItem component will rotate 360 degrees on its z-axis in 8 seconds, meaning it will complete a full rotation in 8 seconds. Test your animation by clicking the Play button on the animation toolbar. You should see the MyItem component continuously spin around, as shown in the following image:



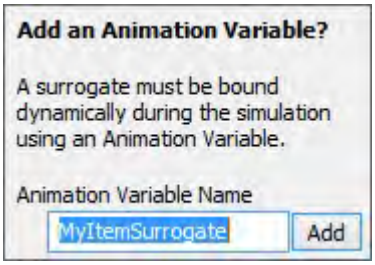
Step 4 - Create a Surrogate and Animation Variable

Now you will change the MyItem component into a surrogate. A surrogate is a special kind of component that will be replaced visually by another object (such as a flowitem) when an animation is running in the simulation model. It acts like a placeholder for another object in an animation. In this case, the MyItem component will be replaced by the default flowitem shape: a box.

1. Click the MyItem component in one of the preview windows to select it.

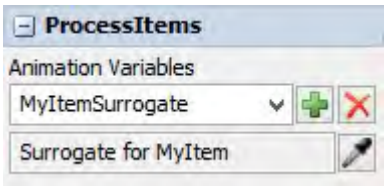
2. In Quick Properties, under MyItem: Component Properties, open the Draw Surrogate menu and select Flow Item: Box. Now the surrogate will change visually to look like a flowitem box during the animation.

3. Right below the Draw Surrogate menu, a window will appear that will prompt you to create an animation variable, as shown in the following image:



4. If this window doesn't appear, it's possible that you outside the Quick Properties pane and closed the window. To make the window re-appear, open the Draw Surrogate menu and select Flow Item: Box again.

5. Click the Add button to add a new animation variable called *MyItemSurrogate*. Notice that this animation variable now appears in Quick Properties under ProcessItems in the Animation Variables box, as shown in the following image:



Step 5 - Trigger an Animation in a Simulation Model

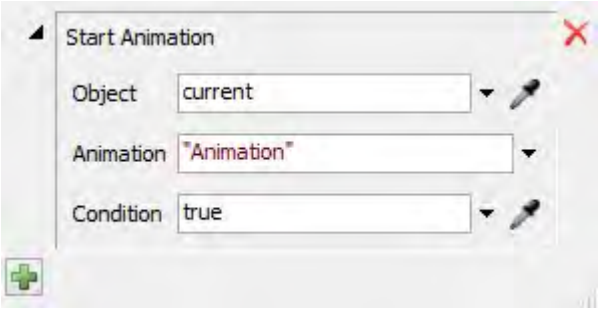
By default, surrogates are hidden in the 3D model until it is assigned a valid 3D object. In this step, you will change the Processor's settings to display the surrogate when a flowitem enters it.

1. Return to the 3D model by clicking the Model tab on the center pane.

2. Double-click the Processor to open its Properties dialog box.

3. In the Triggers tab, click the Add button next to the OnEntry trigger to open a menu.

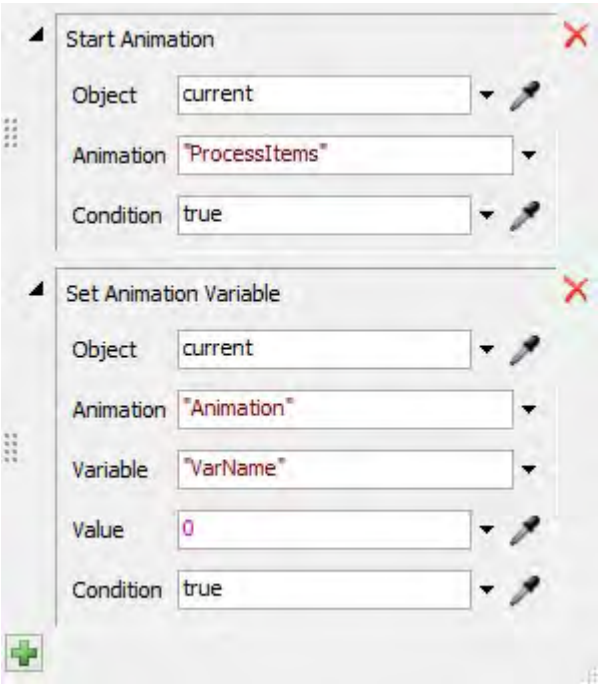
4. Select Visual then Start Animation from the menu to open up the Start Animation picklist options, as shown in the following image:



5. Click the arrow next to the Animation box to open a menu. Select ProcessItems (the animation you just created).

6. Click the Add button underneath the Start Animation picklist options to open another menu.

7. Select Visual then Set Animation Variable from the menu to open up the Set Animation Variable picklist options, as shown in the following image:



8. Click the arrow next to the Animation box to open a menu. Select ProcessItems.

9. Click the arrow next to the Variable box to open a menu. Select MyItemSurrogate (the animation variable you created for the surrogate).

10. Click the arrow nex to the Value to open a menu. Select item so that it will replace the flowitem in the animation.

11. Click outside of the box to save the changes.

12. With the Processor's Propertues dialog box still open, click in the General tab. Under Flags, clear the Show Contents checkbox so that the surrogate will replace the incoming flowitems in the Processor.

13. Click the OK button to save the changes and close the Properties dialog box.

Starting Animations with FlexScript

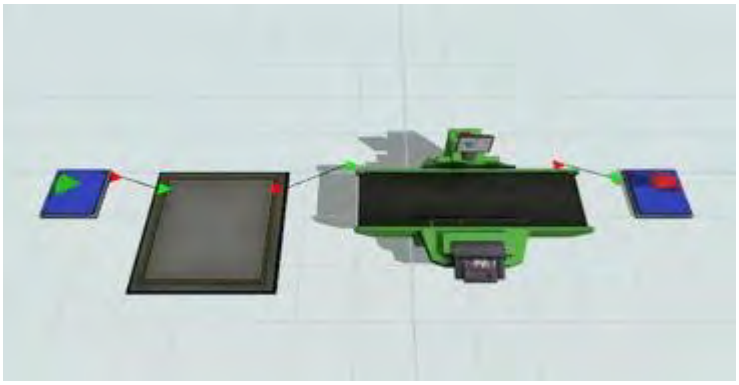
If you want to create custom codes that start an animation, you can use the following Flexscript code on the OnEntry trigger:

```
setanimationvar(current, "Process", "MyItemSurrogate", item); startanimation(current,
"Process");
```

And the following code on the OnExit trigger:

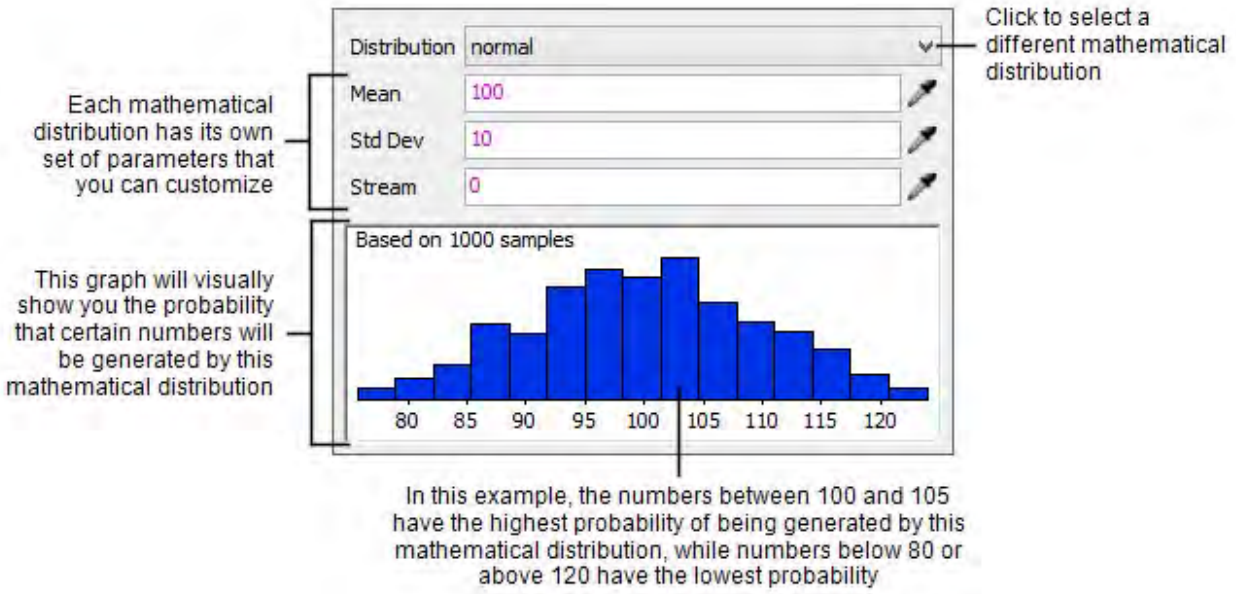
```
setanimationvar(current, "Process", "MyItemSurrogate", 0); stopanimation(current,
"Process");
```

You can now test your animation by running your simulation model. Click the Reset and Run buttons on the simulation control panel. When your simulation runs, the flowitems will rotate inside the Processor, as shown in the following image:



This completes this the Advanced Animation Creator Concepts tutorial.

One feature that sets FlexSim apart from other simulation tools is the ability to quickly and easily build a little bit of variability and randomness into your simulation model. FlexSim gives you the ability to simulate many kinds of fluctuations in your business system. Fortunately, you don't need to know a lot of complicated mathematics or programming to create these mathematical expressions. FlexSim has a tool called the Distribution Chooser that will help you select an appropriate statistical distribution to randomly calculate any kind of randomness. When you're done using the Distribution Chooser, it will create a mathematical expression based on the options you selected and it will automatically apply it to the daily demand or process times settings. (See Using the Distribution Chooser for more information.) The Distribution Chooser is shown in the following image:



The Distribution Chooser allows you to select from thirty possible statistical distributions to generate random numbers. Some of the most commonly used distributions are:

- Exponential - Will randomly select most numbers near the beginning of a range of numbers. The frequency that numbers near the end of the scale will be generated decreases exponentially. This is the default distribution strategy because it is the distribution strategy most commonly found in the real world.
- Duniform - Will randomly select any whole numbers within a specific range. If you want to be able to include numbers with decimal points, use the Uniform distribution strategy instead.
- Normal - Has a distribution strategy similar to a bell curve in which numbers in the middle of the distribution scale are more likely to be generated at random.
- Triangular - Similar to the normal distribution strategy, but has a greater range of numbers in the middle of the distribution scale that are more likely to be generated at random.
- Uniform - Will randomly select any numbers within a specific range. The Uniform distribution strategy will include numbers with decimal points. If you'd rather only create whole numbers, use the Duniform distribution strategy instead.

You can change the settings for each statistical distribution so that it uses different criteria or ranges of numbers to generate the random numbers. When you make changes to the settings, the blue bar graph will update to display the probabilities that certain numbers will be generated.

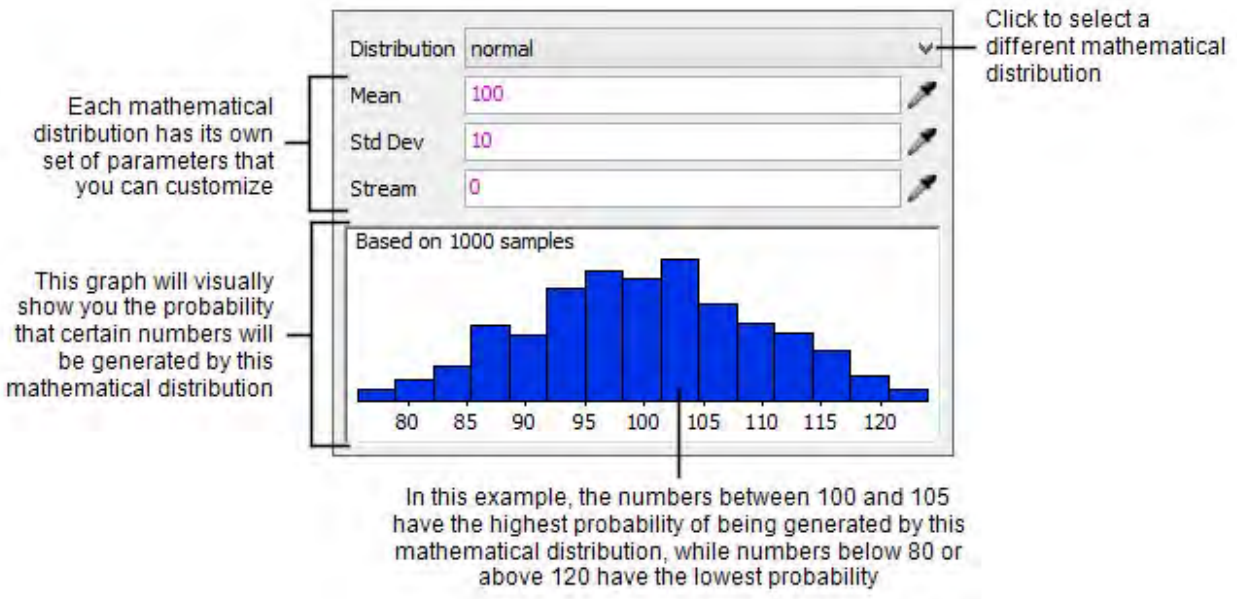
Learning About the Other Distributions

Consider looking up some of the other statistical distributions in a book about statistics to learn more about them. FlexSim's ExpertFit tool can also help you decide on an appropriate distribution strategy using historical data or data gathered from a time study.

Many object properties are capable of using a statistical distribution. To use the Distribution Chooser for these properties:

1. Click the arrow next to the property to open a menu.

2. Select Statistical Distribution to open the Distribution Chooser tool, as shown in the image below:



3. Click the Distribution menu to select a different mathematical distribution if needed.

4. Edit the specific parameters for that mathematical distribution to adjust the range and probability that certain numbers might be generated.

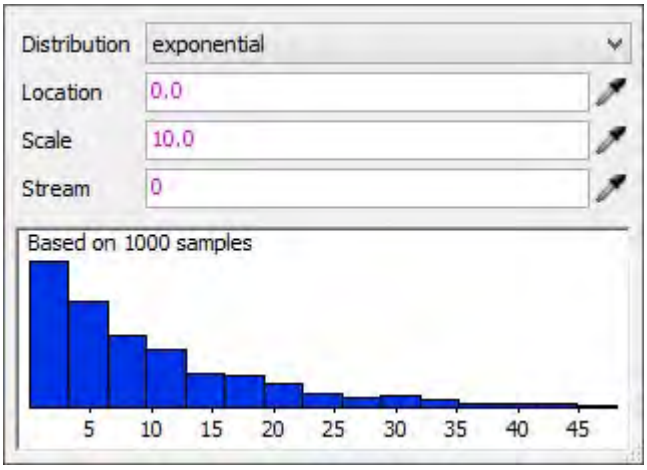
5. Use the bar graph to check to see if the probabilities look accurate and make adjustments as needed.

6. Click somewhere outside the Distribution Chooser to close the tool. You'll notice that the property's box will now show a mathematical expression based on the parameters you specified.

7. If you need to choose a different distribution or you need to edit the parameters, you can click the Properties button next to the property to re-open the Distribution Chooser. Alternatively you can click inside the property's box and edit the mathematical distribution directly.

For more information about some of the different distributions and their parameters, read the next topics in this section.

The Exponential distribution will randomly select most numbers near the beginning of a range of numbers. The frequency that numbers near the end of the scale will be generated decreases exponentially. This is the default distribution strategy because it is the distribution strategy most commonly found in the real world. The following image shows an example of the Exponential distribution strategy:

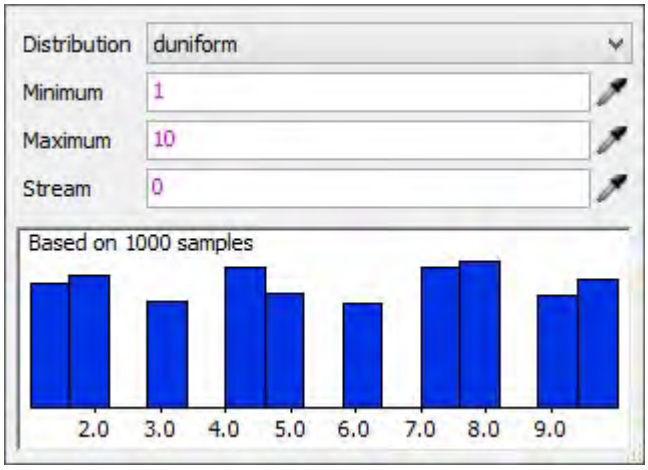


The Exponential distribution has three settings:

- Location - Sets the beginning of the range of numbers. Most of the random numbers generated will be roughly close to the number you use for this parameter.
- Scale - Sets the mean of the distribution and affects the overall pitch of the curve.
- Stream - Determines which one of FlexSim's random number streams should be used to generate these random numbers. Typically you won't need to edit this parameter.

When you make changes to the settings, the blue bar graph will update to display the probabilities that certain numbers will be generated. Usually, the bar graph doesn't change much for the Exponential distribution, but the range of the numbers listed on the bottom of the graph will change.

The Duniform distribution will randomly select numbers within a specific range. If you want to be able to include numbers with decimal points, use the Uniform distribution strategy instead. The following image shows an example of the Duniform distribution strategy:

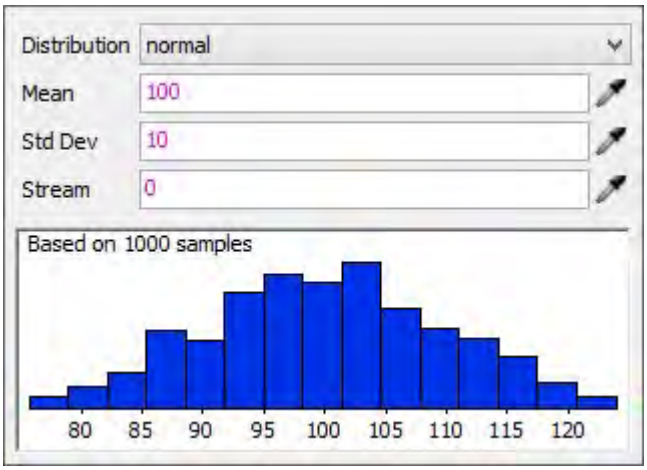


The Duniform distribution has three settings:

- Minimum - The lowest possible number that might be generated at random.
- Maximum - The highest possible number that might be generated at random.
- Stream - Determines which one of FlexSim's random number streams should be used to generate these random numbers. Typically you won't need to edit this parameter.

When you make changes to the settings, the blue bar graph will update to display the probabilities that certain numbers will be generated.

The Normal distribution has a distribution strategy similar to a bell curve in which numbers in the middle of the distribution scale are more likely to be generated at random, as shown in the following image:

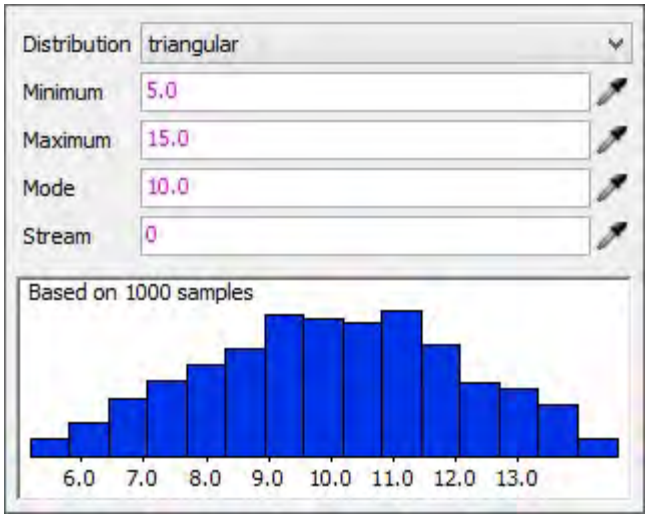


The Normal distribution has three settings:

- Mean - The most common number on average that will be generated at random.
- Standard Deviation (Std Dev) - The amount of variation or dispersion of values relative to the mean. A lower standard deviation will cause the numbers to be very close to the mean. A higher standard deviation will generate numbers that are more widely dispersed from the mean.
- Stream - Determines which one of FlexSim's random number streams should be used to generate these random numbers. Typically you won't need to edit this parameter.

When you make changes to the settings, the blue bar graph will update to display the probabilities that certain numbers will be generated.

The Triangular distribution is similar to the normal distribution strategy, but has a greater range of numbers in the middle of the distribution scale that are more likely to be generated at random, as shown in the following image:



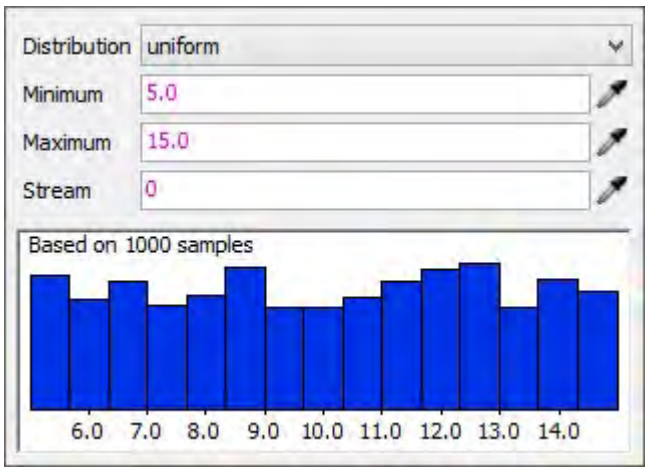
The Triangular distribution has four settings:

- Minimum - The lowest possible number that might be generated at random.
- Maximum - The highest possible number that might be generated at random.
- Mode - The most common number that will be generated at random.

- Stream - Determines which one of FlexSim's random number streams should be used to generate these random numbers. Typically you won't need to edit this parameter.

When you make changes to the settings, the blue bar graph will update to display the probabilities that certain numbers will be generated.

The Uniform distribution will randomly select numbers within a specific range. The uniform distribution strategy will include numbers with decimal points. If you'd rather only create whole numbers, use the Duniform distribution strategy instead. The following image shows an example of the Uniform distribution strategy:



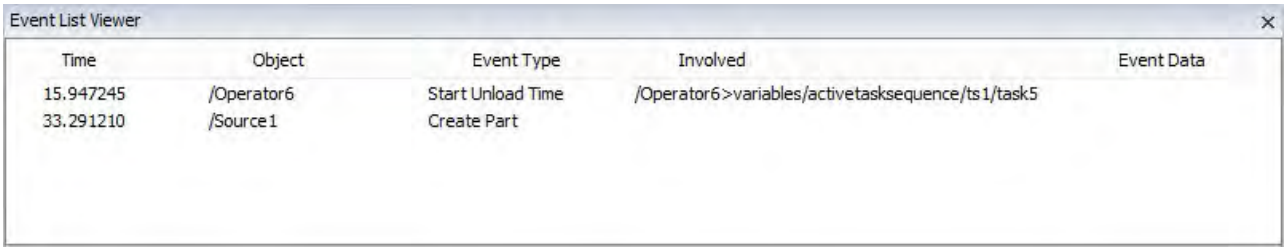
The Uniform distribution has three settings:

- Minimum - The lowest possible number that might be generated at random.
- Maximum - The highest possible number that might be generated at random.

- Stream - Determines which one of FlexSim's random number streams should be used to generate these random numbers. Typically you won't need to edit this parameter.

When you make changes to the settings, the blue bar graph will update to display the probabilities that certain numbers will be generated.

Event List



The Event List is accessed from the Debug menu > Event List.

The Event List shows all the pending events for the model. It is useful for seeing when different events will occur in order to debug modeling issues. If you have a problem that happens during a particular event, the Event List is useful for seeing information about that event to help track down the source of the problem. If you want to only view the events for a particular object, you can right-click on the object in the 3D View and select View > View Object Events .

Time - This is the time that the event occurs.

Object - This is the path to the object, relative to the model, that the event affects.

Event Type - This is the type of event. It is the event code and will show a number value for event codes without registered names. You can use the "seteventlistlegendentry" application command to register a name for custom event types.
For example: applicationcommand("seteventlistlegendentry", 102, "My Event Type", 0); will set event code 102 to show "My Event Type" as its name in the list.

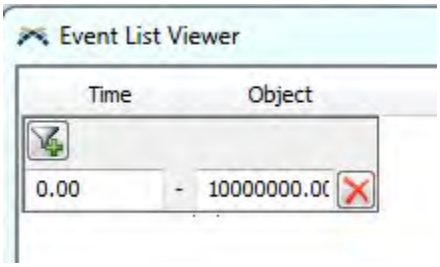
Involved - This is the path to the involved object for the event.

Event Data - This value's use depends on the event and may not be used for all event types.

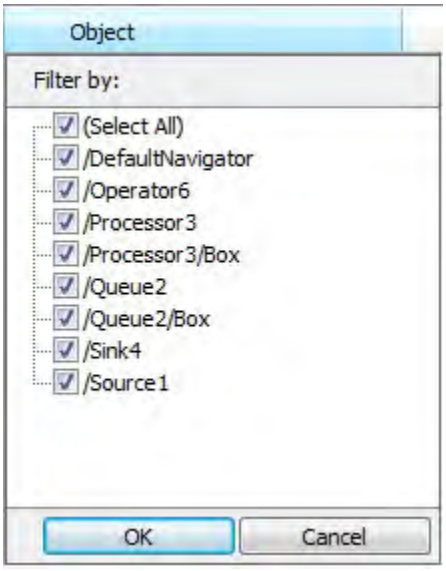
Filters

The Event List can be filtered based on the Time, Object and Involved columns. Columns with an active filter will display a . To add/edit a filter, left-click on the header name for the desired filter.

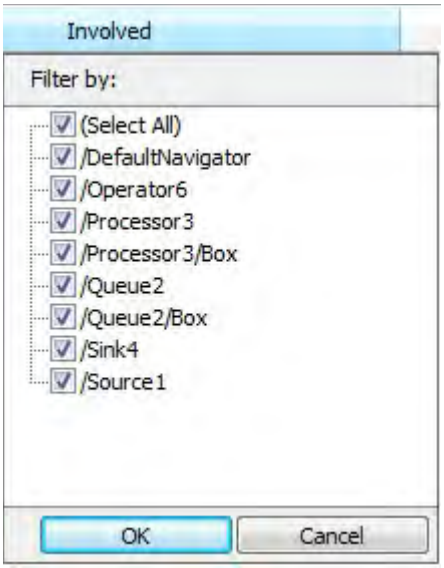
Time - Each time filter has a begin (left) and an end (right) field. Only events that occurred within those two times will be displayed. To add a filter, click the icon. To remove a filter, click the icon.



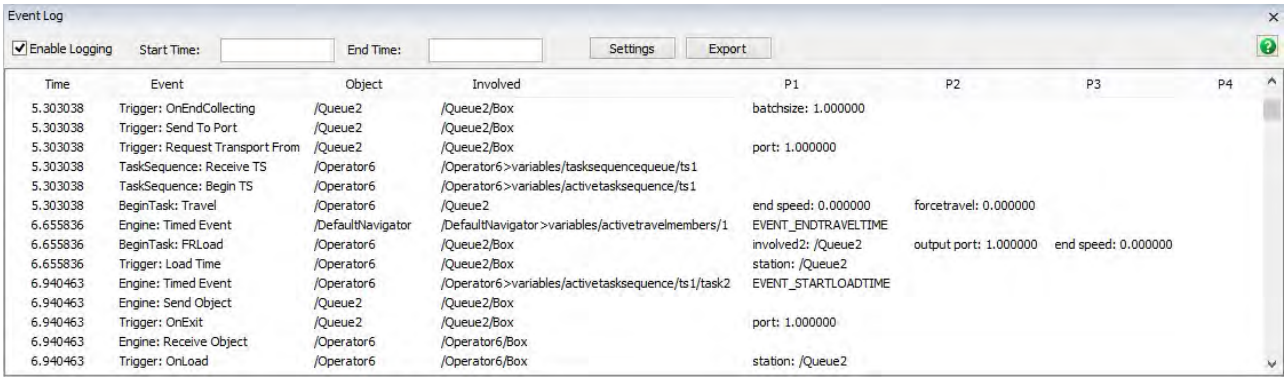
Object - This list allows you to filter the event list by which object generated the event. To include or exclude an object in the list, check or uncheck the box next to its name.



Involved - This list allows you to filter the event list by which object is involved in the event. To include or exclude an object in the list, check or uncheck the box next to its name.



Event Log



The Event Log is accessed from the Debug menu > Event Log.

When Enable Loggin is checked, the Event Log will create a record of events that occur in the model. It is useful for seeing the order in which certain events took place. For each event that happens in the model, multiple entries may be made in the Event Log to explain what happened during that event. These multiple entries will all have the same time and all be logged simultaneously when you press the Step button. The event log will be cleared when the model is reset.

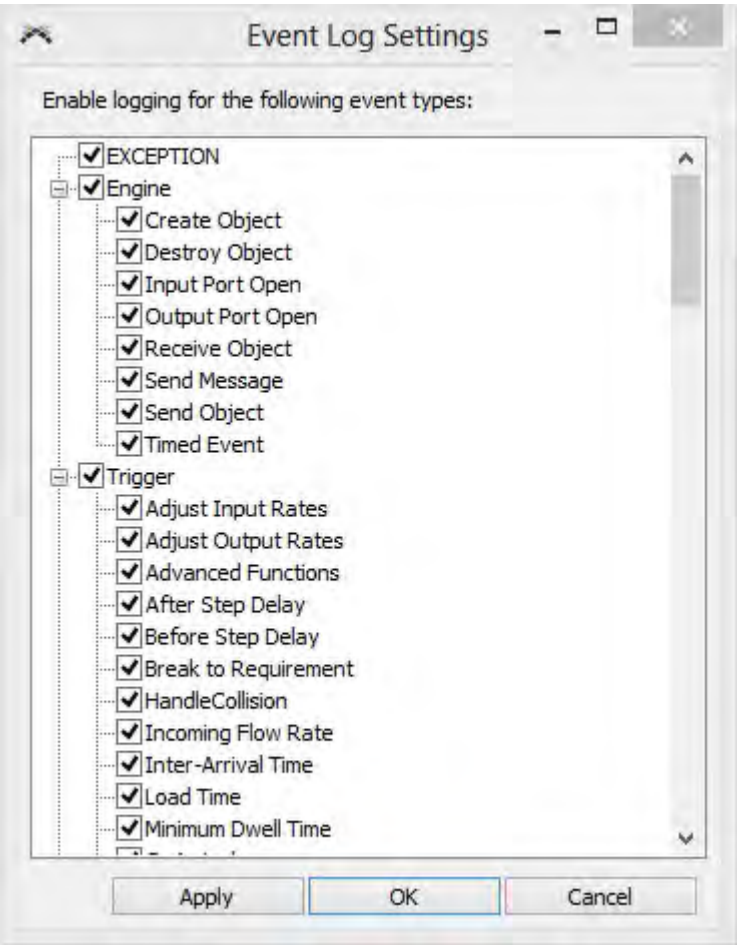
Some exceptions will be recorded in the event log. The entry immediately preceeding the exception entry will give you a clue as to where the code is that caused the exception to happen. This is particularly useful if the exception was caused by improper code in an object's trigger. More information about the exception may be available in the System Console. The model may not be behaving correctly if there are exceptions happening in the code.

Enable Logging - This will enable or disable event logging. The model will run much slower when logging is enabled so you should disable logging when you are finished using the event log.

Start Time - If you only want to log a specific time period, you can enter a start time for when the logging will begin. This will automatically be applied after editing this field without having to reset and rerun the model.

End Time - Optionally, you can specify an end time for when you want the logging to stop. If the end time is less than or equal to the start time, it will be ignored.

Settings - Within the settings window, you can set up which events should be recorded in the event log. Events that have already been recorded will not be affected by changing these settings. Events that occur after changing these settings will only be recorded if they are enabled here.



Export - This will export the Event Log as a csv file. If will only export valid events, ignoring any events that have been filtered out.

The Table

Time - This is the time that the event happened. The entries happened in order from top to bottom. Entries recorded with the same time happened in the order shown and may have happened during the same model event.

Event - This is the type of event. You can enable or disable logging for certain event types in the Settings window.

Object - This is the path to the event's object.

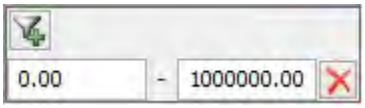
Involved - This is the path to the involved object for the event.

P1 - P4 - These values depend on the event and may not be used for all event types. Usually they give you information about what parameters were passed into the event or more information about the event type. This is useful for debugging if parameter values are not what you expect them to be.

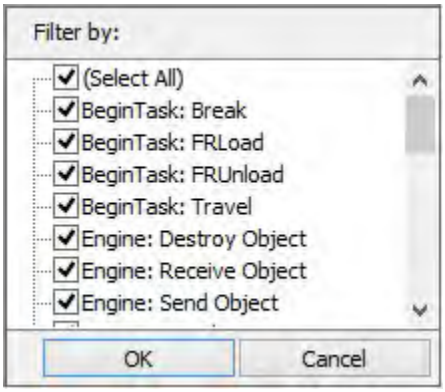
Filters

The Event Log can be filtered based on the Time, Event, Object and Involved columns. Columns with an active filter will display a . To add/edit a filter, left-click on the header name for the desired filter. Event log entries that are no longer displayed because they have been filtered will not be exported with the Export button.

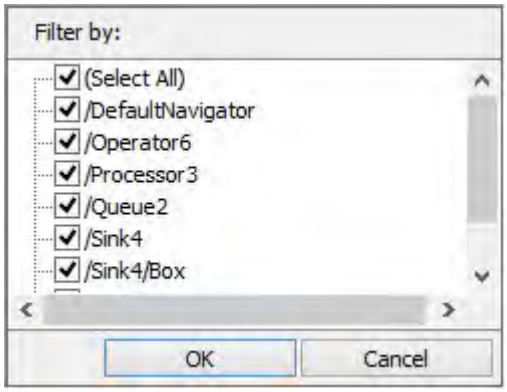
Time - Each time filter has a begin (left) and an end (right) field. Only events that occurred within those two times will be displayed. To add a filter, click the icon. To remove a filter, click the icon.



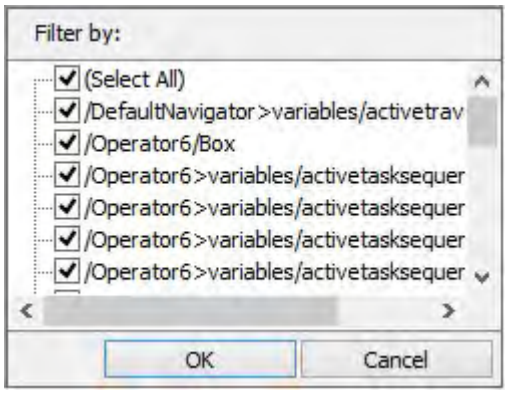
Event - This list allows you to filter the list by which event or trigger the event was associated with. To include or exclude an event from the list, check or uncheck the box next to its name.



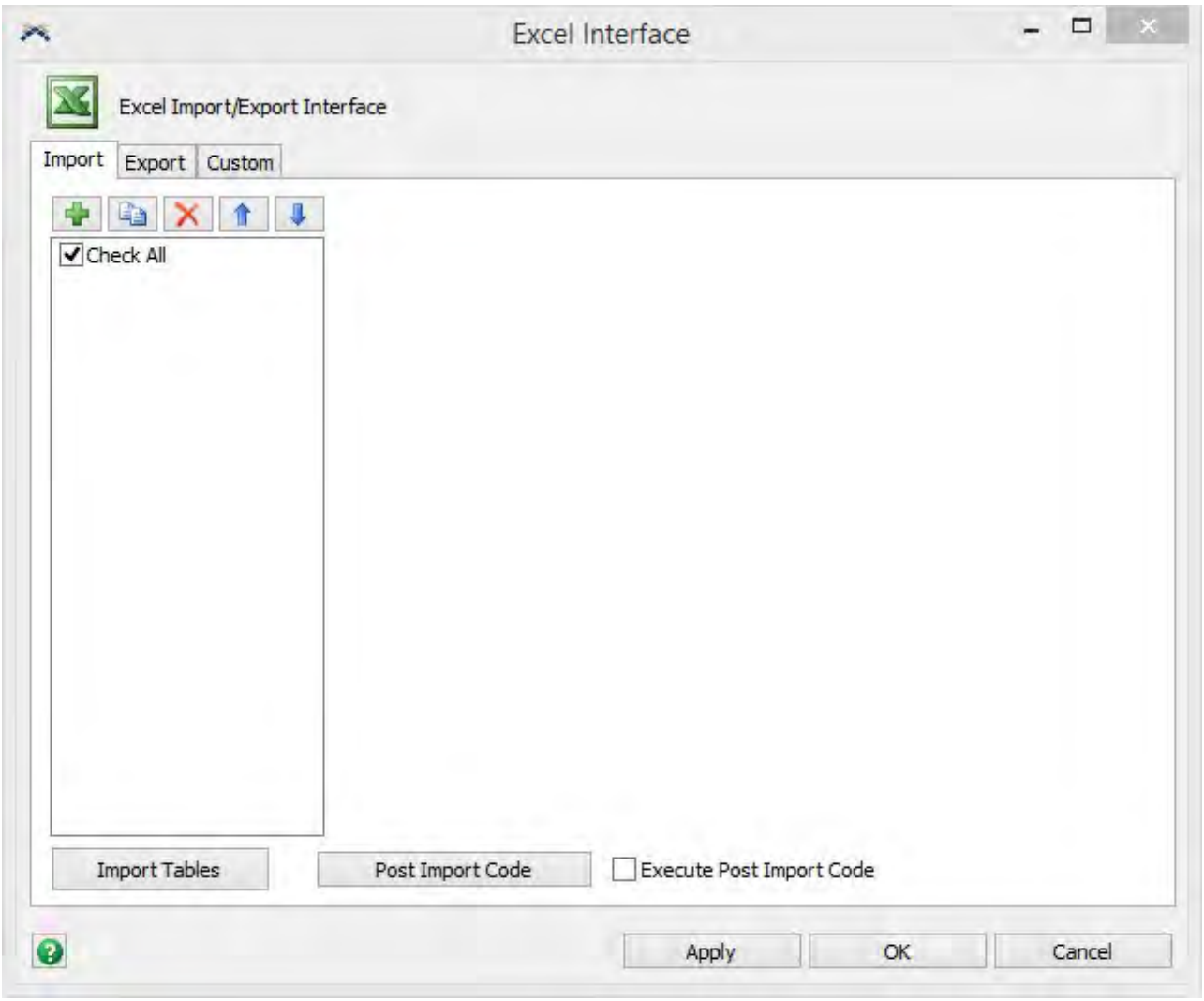
Object - This list allows you to filter the list by which object generated the event. To include or exclude an object in the list, check or uncheck the box next to its name.



Involved - This list allows you to filter the list by which object is involved in the event. To include or exclude an object in the list, check or uncheck the box next to its name.



Excel Interface



Excel or from the Toolbox. (View menu > Toolbox

The Excel Interface is accessed from the main toolbar > Add > Import > Excel Import/Export).

The Excel Interface was designed to make importing and exporting multiple worksheets from more than one workbook very fast and easy to do. You can also create your own custom Import/Export code.

Overview

Multiple Table Excel Import (MTEI)

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 automated in its implementation, it is extremely useful for importing data that will change over time.

Multiple Table Excel Export (MTEE)

The MTEE allows you to export multiple tables to multiple different Excel workbooks. The controls and features of this page are the same as the MTEI page. See the documentation above for more information on this page.

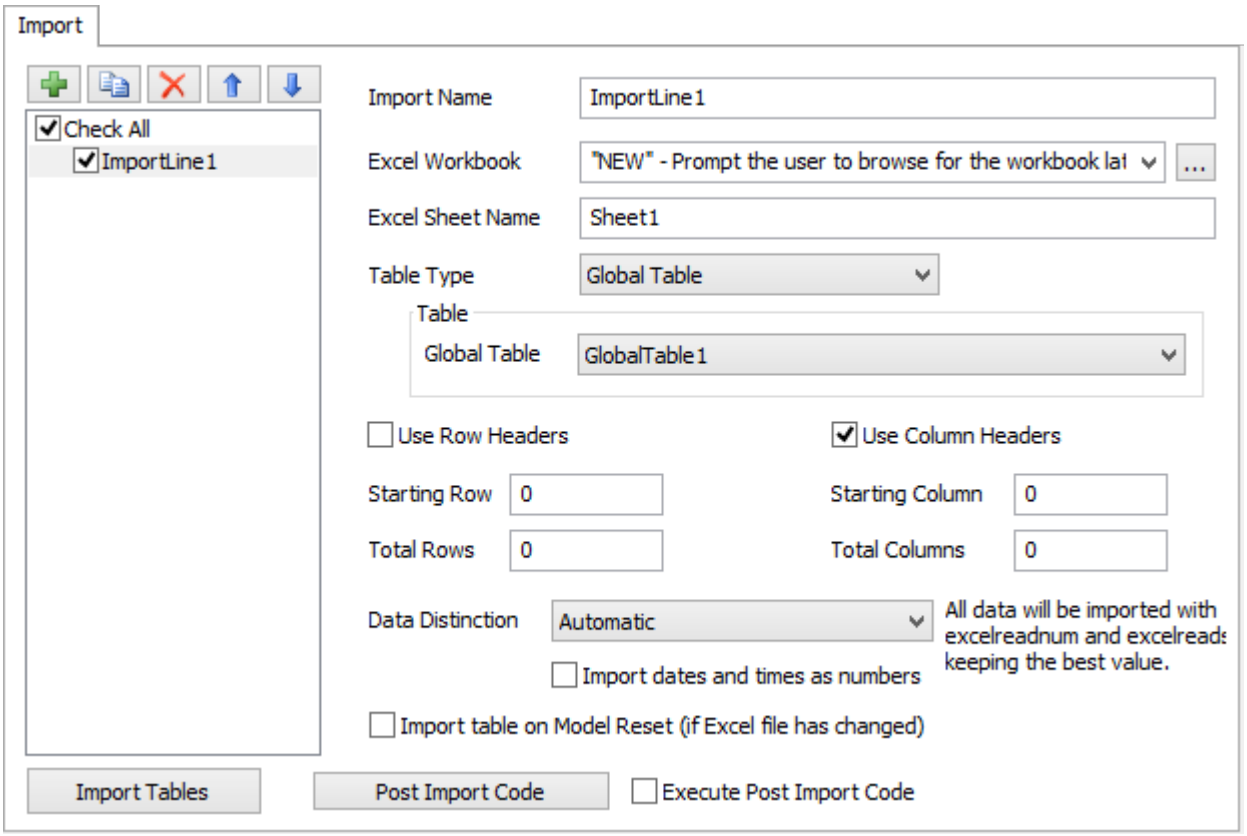
Custom Import/Export

The custom Import/Export page allows you to write your own custom code to import and export to Excel workbooks. There is sample code in the picklists on importing and exporting from Excel.

Excel Interface Pages

- Import Page
- Export Page
- Custom Page

Import Page



Import Lines (list)- Displays all of the import lines for the MTEI. You can use the Add and Duplicate buttons to add additional lines. You may also rearrange lines to group import lines that are importing from the same

Excel workbook (this will improve speeds greatly as opposed to trying to switch back and forth between the same workbooks).

Import Tables - The MTEI will go in order (from the top) through each checked line of the Import Lines window and execute the import based on its properties specified on the right side.
Note: The MTEI and MTEE may also be started by calling excelmultitableimport() or excelmultitableexport() respectively in a flexscript node.

Import Name - The import name is only used in the Import Lines window to help identify the line.

Excel Workbook

The Excel Workbook is where you define the name of the Excel workbook file that contains the information that you want to import. There are four ways to enter information into this column.

- 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 select the "NEW" option. Using "NEW" in this field will cause the browse window to open, prompting the user to find the Excel file they want to use when executing the import. 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 import line then you can leave this field blank. This is the recommended option when you are importing information from multiple sheets in the same workbook. You can reorder import lines using the arrows on the left side of the window in order to group imports that use the same workbook.
- Absolute Path (ABSOLUTE) If the location and the 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"
  If the file is unable to be found, the import will be stopped and you'll have the option to skip the file or manually select a file to import.

  Note: The ".xls" extension is essential in order for FlexSim to find the right Excel spreadsheets. You can also use ".xlsx" to import Excel 2007 or newer spreadsheets.
- Relative Path (RELATIVE) If your Excel file is in the same directory as your model, or in a sub directory, you can enter the relative path of the Excel file, or browsing for the file will give the relative path to the file. Alternitvely, you can enter the relative path with respect to the install directory of FlexSim. For example "userprojects\myproject\myfile.xls"
  If the file is unable to be found, the import will be stopped and you will have the option to skip the file or manually select a file to import.

Excel Sheet Name - The name of the Excel sheet that contains the import information. For example "Sheet1". If the MTEI does not find the name of the sheet because it does not exist in the workbook or has been entered incorrectly, the import will pause and alert you of the problem. You will then 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.
Table Type - Select the type of table in FlexSim that should receive the data from the import. There are 8 table types:

1. Combiner Component Table
2. Conveyor Layout Table
3. Global Table
4. Pipe Layout Table
5. Source Table
6. Time Table
7. Traffic Control Table

8. Other

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 row or column for the header information is automatically calculated. The header information should always come before any data distinction information or actual data.

Starting Row and Starting Column - The starting row and starting column fields 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. 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.

Total Rows and Total Columns - The Total Rows and Total Columns fields determine the amount of rows and columns that the MTEI will import. If you set these entries to 0 the MTEI will automatically calculate the number of rows or columns for you. 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 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.

Data Distinction - Data distinction determines how the MTEI will extract values from the Excel workbook. There are three commands that the MTEI can use to read Excel data:

- excelreadnum() - If the specified cell contains number data, this command returns the number. Otherwise, it returns 0.
- excelreadstr() - If the specified cell contains text data, this command returns the text. Otherwise, it returns the value displayed in the cell as text.
- excelrangeread() - This command reads all cells in a specified range. If a cell in that range contains text data, this command extracts the text. Otherwise, it extracts the value as a number.

When you import a spreadsheet, you must choose a data distinction mode. The mode you choose will tell the MTEI which of the three commands (or what combination) it should use to extract data from a spreadsheet. The modes are described in the following list:

1. Numeric - The MTEI uses the excelreadnum() command to extract each cell's value.
2. Automatic (default) - The MTEI uses both excelreadnum() and excelreadstr() to extract each cell's value. It then analyzes both results to determine whether the cell contains string or number data, and based on that, which result to put in the table.
3. Per Column - For all the cells in a given column, the MTEI gets the value in the first non-header row (the beginning of the column), which must be a number. Depending on that value, the MTEI will use excelreadnum() or excelreadstr() to extract the data in that column. The cell containing this value is excluded from the final table in FlexSim.
4. Per Row - For all the cells in a given row, the MTEI gets the value in the first non-header column (the beginning of the row), which must be a number. Depending on that value, the MTEI will use excelreadnum() or excelreadstr() to extract the data in that row. The cell containing this value is excluded from the final table in FlexSim.
5. Text - The MTEI uses the excelreadstr() command to extract each cell's value.

6. Values Only (very fast) - The MTEI will use the excelrangeread() command to read in all values of the table.

For the Automatic mode, the MTEI will attempt to determine whether the cell contains text or number data. While this process works fairly well, the MTEI will occasionally be wrong. For example, a cell with number data, formatted as a fraction, will be imported as text like "1/5" rather than a number like 0.20. Empty cells will be read in as a string. In order to correctly import a worksheet, you may need to change the formatting in Excel or the data distinction mode.

For the Per Column and Per Row modes, the MTEI reads the first value in a row or column and uses that value to determine how to extract the remaining values in that row or column. The MTEI recognizes four possible values:

1. Numeric Data - The MTEI will extract the data in this row/column using excelreadnum()
2. Text Data - The MTEI will extract the data in this row/column using excelreadstr()
3. Flexscript Data - The MTEI will extract the data in this row/column using excelreadstr(). If importing into a table, the MTEI will toggle this cell's node as FlexScript.
4. C++ Data - The MTEI will extract the data in this row/column using excelreadstr(). If importing into a table, the MTEI will toggle this cell's node as C++. If this occurs, you will be prompted to compile the model after the import is complete.
5. Pointer / Coupling Data - The MTEI will import this row/column using excelreadstr() and then use that as a path to an object/node in the model using model().find("IMPORTED_TEXT")
6. Dynamic Data - The MTEI will import this row/column using excelreadstr() and then will try to determine what the data is. This could be numeric data, strings, pointers to objects or arrays. To import an array use square brackets and commas for each element. For example, [1, 2, 3, "Text", Source1]

The Values Only mode uses the excelrangeread() command. This command imports text cells the same way excelreadstr() does. However, all other values, including dates and times, are interpreted as raw numbers. You can use the VALUE() function in Excel to see the raw number for any non-text cell. The following table shows some examples of how an Excel cell might be extracted with each of the three commands:

Cell Value	Cell Format	excelreadnum()	excelreadstr()	excelrangeread()
1.54	Number	1.54	"1.54"	1.54
10	Number	10.0	"10"	10.0
2/8/2016	Date	42408.0	"2/8/2016"	42408.0

some text	Text	0	"some text"	"some text"
1/5	Fraction	0.20	"1/5"	0.20

Note on Excel Import Performance: For small tables, the data distinction mode has very little effect on the import speed; small tables are usually imported very quickly. For larger tables, however, the data distinction mode can have a dramatic effect on import speed. To ensure the fastest speed possible, choose a data distinction mode that only uses excelreadstr() to read cells with text data. For example, if you have a column of text data and a column of number data in a spreadsheet, use the Per Column distinction mode (with the correct data distinction values), or the Values Only mode.

Import Dates and Times as Numbers - This option is only available for the Automatic Data Distinction setting. When checked, the importer will import cells from Excel that are formatted as dates and times as numbers. It will also convert the number to be useable by FlexSim which is the number of seconds since Jan 1 1601.

Import Table on Model Reset - If checked, FlexSim will re-import this table when the model is reset. This will ONLY occur if the Excel file has been changed since the last time the table was imported.

Post Import Code - After all the import lines are executed, the MTEI can execute Post Import Code. You can write custom code in this trigger to do any additional operations after all tables have been imported. Execute Post Import Code - If checked, the Post Import Code will be executed once all tables have been imported.

Importing to a Bundle

Global Tables may be set to use a bundle to store their data. Or, when using the Other table type, you can select a node in the tree that has bundle data. In these cases the import will create the correct fields in the bundle for the imported data. Due to the way bundles work, there are a few restrictions when importing to a bundle.

Automatic Data Distinction - When automatic data distinction is used, the importer will only look at the datatype of the first cell in a column and set the entire column's data type to that type. This is because each column of a bundle must either be all numeric or all string data.

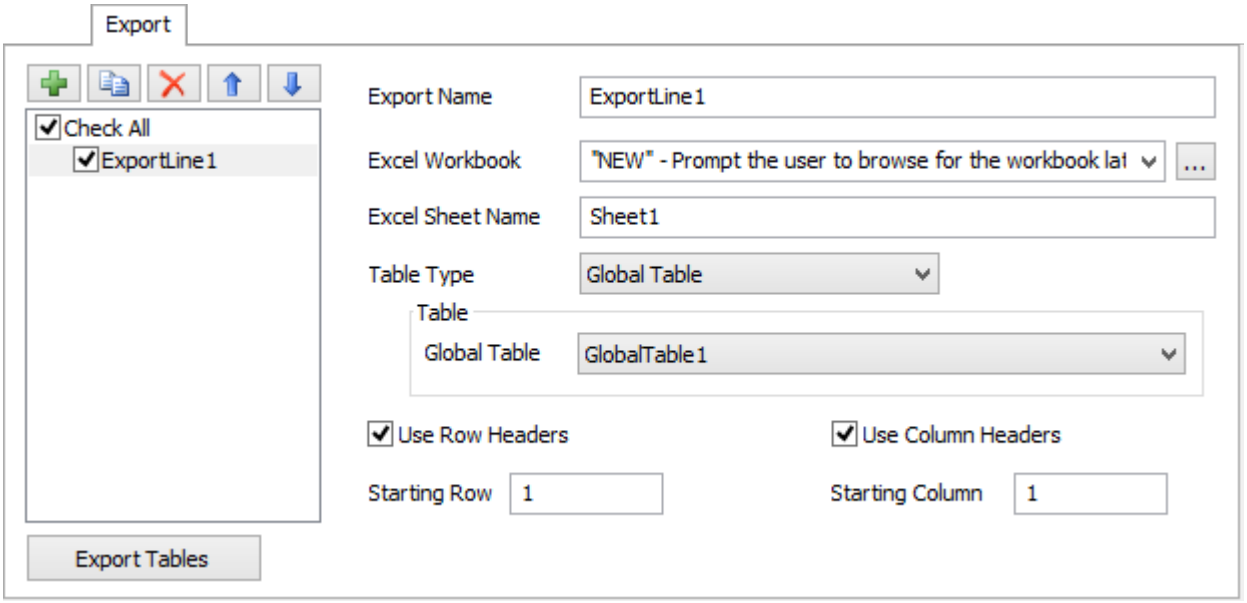
Column Data Distinction - Selecting Column will cause the importer to behave as if using Automatic Data Distinction. This is because rows in a bundle column may not have differing data types.

Row Headers - As bundles do not have row headers, checking Use Row Headers has no affect on the import.

Column Headers - Bundles must have a Column (or field) name. Therefore, Use Column Headers is always used whether checked or unchecked.

Flexscript/C++ Data - Bundles can only have string and numeric data.

Export Page



Unlike the MTEI, the MTEE does not use any data distinction when writing to the Excel file. The data type is taken from the FlexSim node data type.

Starting Row and Starting Column - The starting row and starting column fields specify which cell in Excel the data from FlexSim will start being exported to.

There is no Number of Rows or Number of Columns fields. The export will take the entire table.

Exporting Bundle Data

When using the Other table type, you can select any node in the tree to export your data from. If the node has bundle data, the bundle's data will be exported to Excel. Because there are no Row Headers in a bundle, selecting Use Row Headers has no effect.

Custom Page

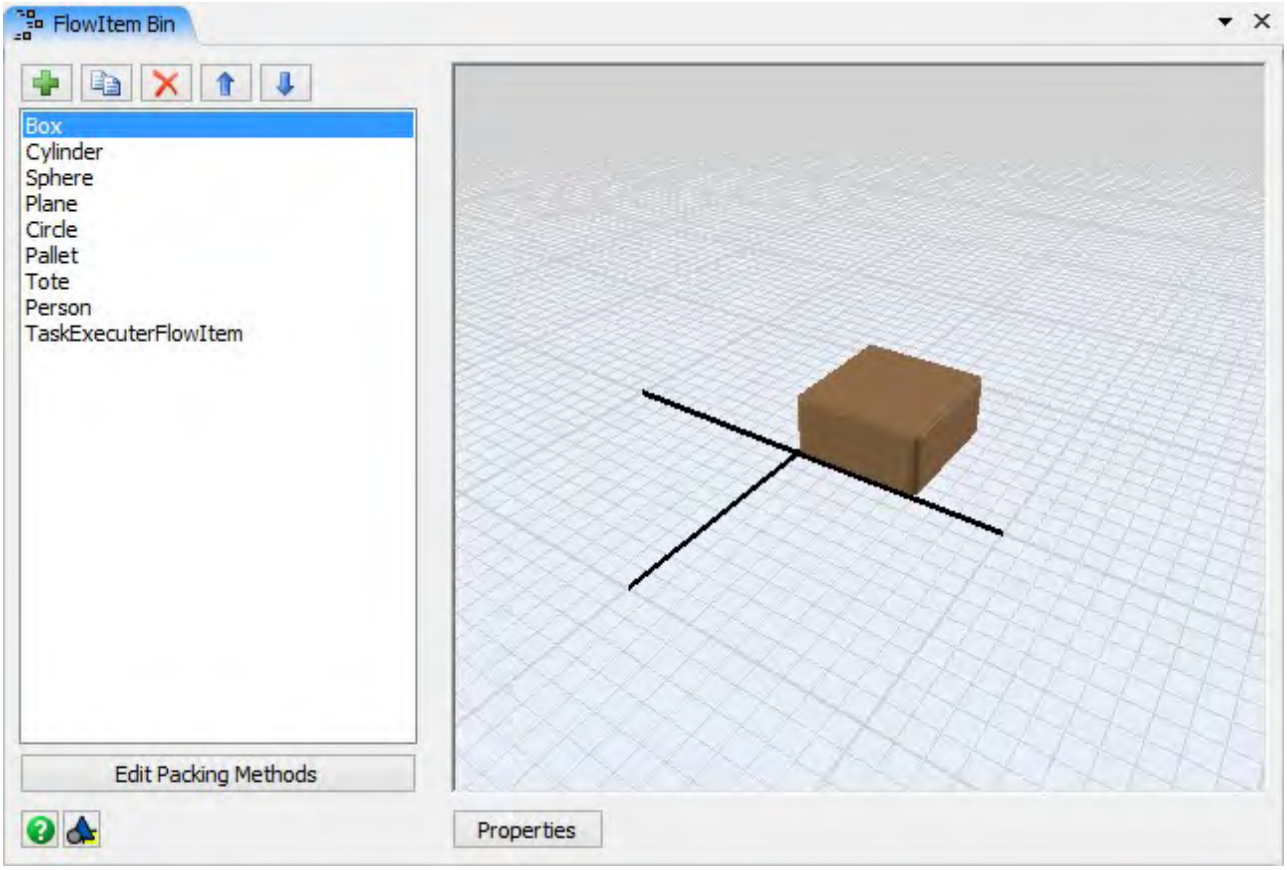
Description - This field has no impact on the custom import/export. It is purely for the user's information.

Code - Enter your own code to import/export from Excel.

To execute your custom import or export code through a flexscript node or trigger, call the following:

```
treenode excelauto = node("/Tools/ExcelAuto", model()); executefsnode(getvarnode(excelauto,
"CustomImport"),NULL);
// or
executefsnode(getvarnode(excelauto, "CustomExport"),NULL);
```

Flowitem Bin Concepts



The FlowItem Bin is accessed FlexSim's toolbar or from the Toolbox. (View menu > Toolbox > FlowItem Bin).

The Flowitem bin stores all of the Flowitems used in your model. You can learn more about Flowitems in the FlexSim Concepts - Flowitems page. Different classes of flowitems are created in this window and are stored in the Flowitem Bin.

Flowitems that are created in the model are exact copies of the Flowitems in the Flowitem Bin. A Source object specifies what Flowitem class to create. It then creates a copy of the Flowitem and places it in the model.

Flowitem Types

There are three types of flowitems, Basic, Container and TaskExecuter. When a new model is created, a default set of Flowitems is added to the Flowitem bin. The Box, Cylinder, Sphere, Plane, Circle and Person are all Basic Flowitems. The Pallet and Tote are Container Flowitems and the TaskExecuterFlowItem is a TaskExecuter Flowitem.

Basic Flowitems

Basic Flowitems have the following properties:

Name - Each Flowitem has its own name. However, unlike FixedResource and TaskExecuter objects, Flowitem names do not need to be unique within the model. If the Flowitem is named "Box" in the Flowitem bin, then all copies of that Flowitem will be named "Box" unless the name is explicitly changed.

Itemtype - See the FlexSim Concepts - Itemtype page for more information on Itemtypes.

Labels - See the FlexSim Concepts - Labels page for more information on Labels.

Location, Size, Rotation - Flowitems take up physical space in your model. This allows you to set your Flowitems to be the same size as the parts you are modeling (ie bottles in a bottling line).

Color - As with other objects in FlexSim, Flowitems have a designated color. That color can be changed in the Flowitem bin, but it can also be changed in the model as the Flowitem makes its way through processes. Often, colors are used to represent specific product types and allow the modeler to more easily follow what is happening in a model.

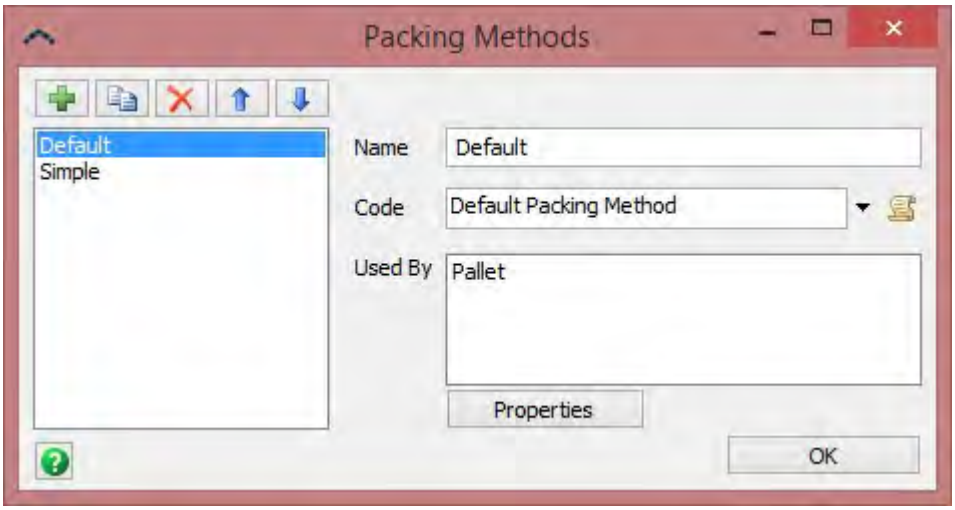
3D Shape - See the 3D Media section for more information on 3D Shapes.

Shape Frames - As with other objects, Flowitems can have Shape Frames. This is particularly useful with Flowitems as it can help show how a part changes as it moves through the model. For example, in a bottling line, the Flowitem might begin as a piece of plastic or glass, then become a bottle, then it is filled, and a cap is put on top. This could be shown by creating five different 3D shapes and changing the Shape Frame.

Animations - New with FlexSim 7, Flowitems can have their own custom Animations using the Animation Creator.

Container Flowitems

Container Flowitems have all of the same functionality as Basic Flowitems. Though all Flowitems can act as containers, only the Container Flowitem executes code when another item is placed inside of it (packed). You can create custom packing methods by clicking on the *Edit Packing Methods* button in the Flowitem bin.



The Packing Methods window allows you to create your own custom packing methods and to alter existing methods. Selecting a packing method will also display the current Flowitems that are using that packing method.

Container Flowitems will display their packing method in the Quick Properties. They also have an extra page in their properties window that allows you to specify which packing method to use. For more information, see the Container Page.

TaskExecuter Flowitems

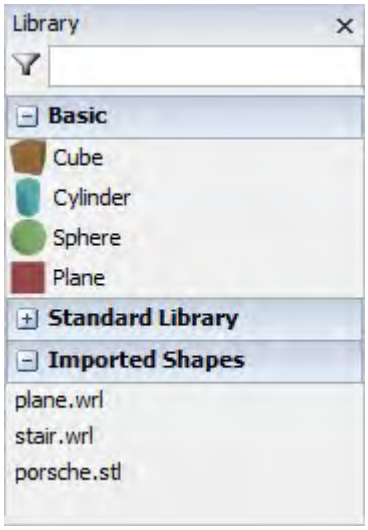
TaskExecuter Flowitems have all of the same functionality as Basic Flowitems. They also have the ability to act as a standard TaskExecuter object. Unlike other Flowitems, they can have their own connections which can be added dynamically using the *contextdragconnection* command.

TaskExecuter Flowitems have the following pages added to their Properties window:

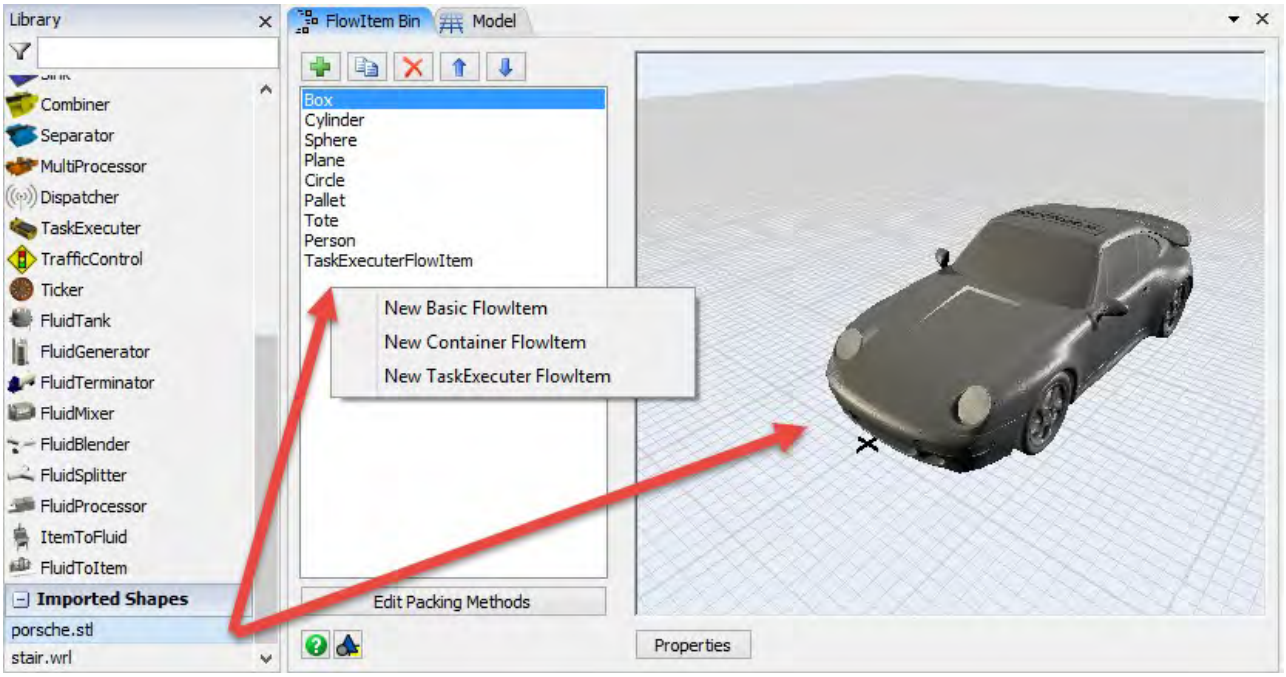
- TaskExecuter
- Breaks
- Collision
- Triggers
- Statistics

Library Icon Grid

When the Flowitem Bin is active, the Library Icon Grid changes to display a list of shapes:



Shapes that have been imported into your model will appear at the bottom of the Library. You can drag and drop shapes from the Library onto the 3D view to quickly change the 3D shape of the Flowitem, or you can drag and drop into the Flowitem Class list to create a new Flowitem:

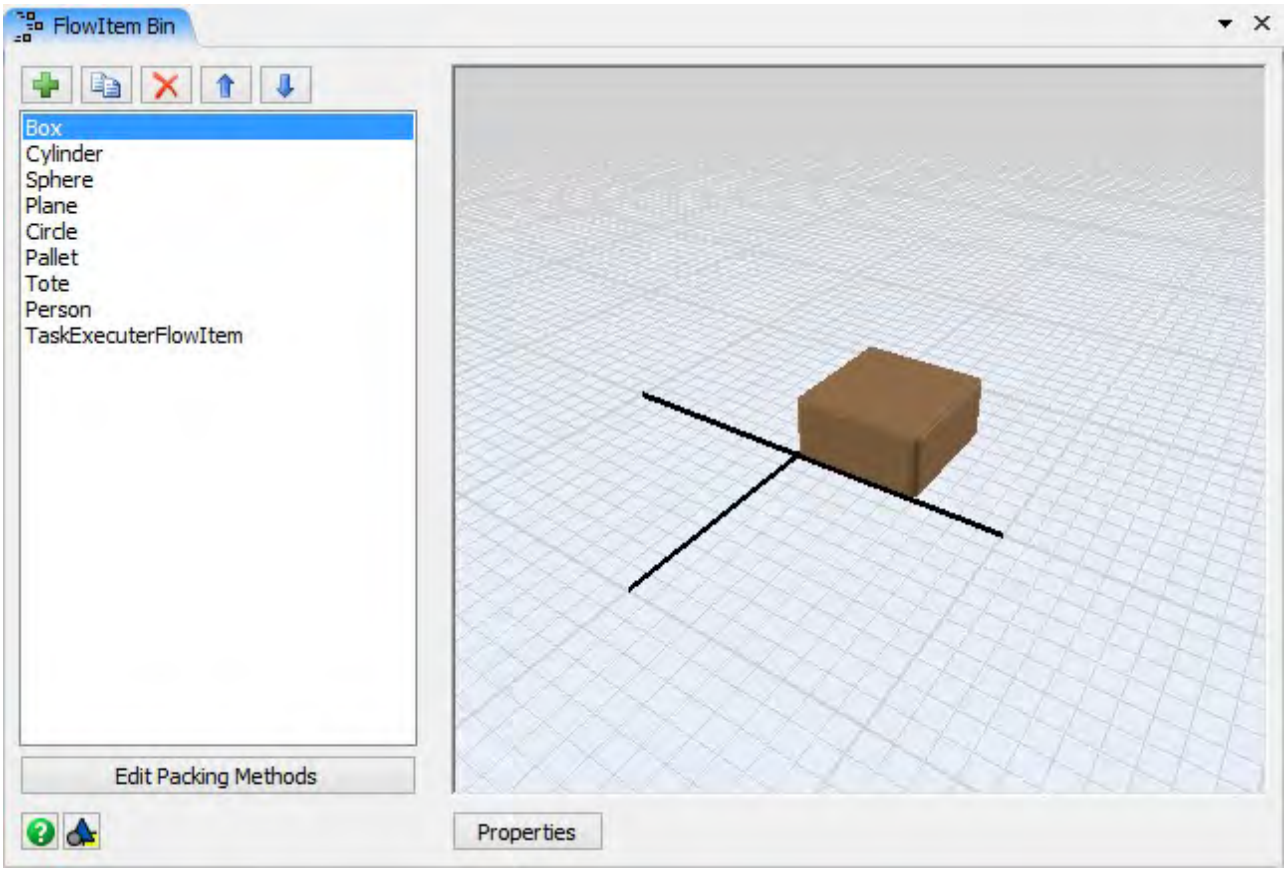


You can filter the list of shapes by entering text into the field.

Flowitem Bin Reference

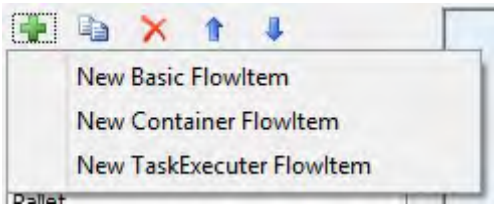
Topics

- Flowitem Classes
- Packing Methods
- Right-Click Menu
- Quick Properties



Flowitem Classes

- Adds a new Flowitem of either Basic, Container or TaskExecuter type to the Flowitem Bin.



- Duplicates the currently selected Flowitem.

- Removes the currently selected Flowitem from the Flowitem Bin.

- Reorders the currently selected Flowitem up or down in the list.

Warning on reordering Flowitems: The Source object refers to the Flowitems it will create based upon rank. Reordering the list may cause Source's to no longer create the correct Flowitems.

Flowitem Class List - This list contains all of the available Flowitem classes. When one is selected, it is shown in the 3D view to the right. The Quick Properties will also update to display the properties for that Flowitem class.

Edit Packing Methods - Opens the Packing Methods window.

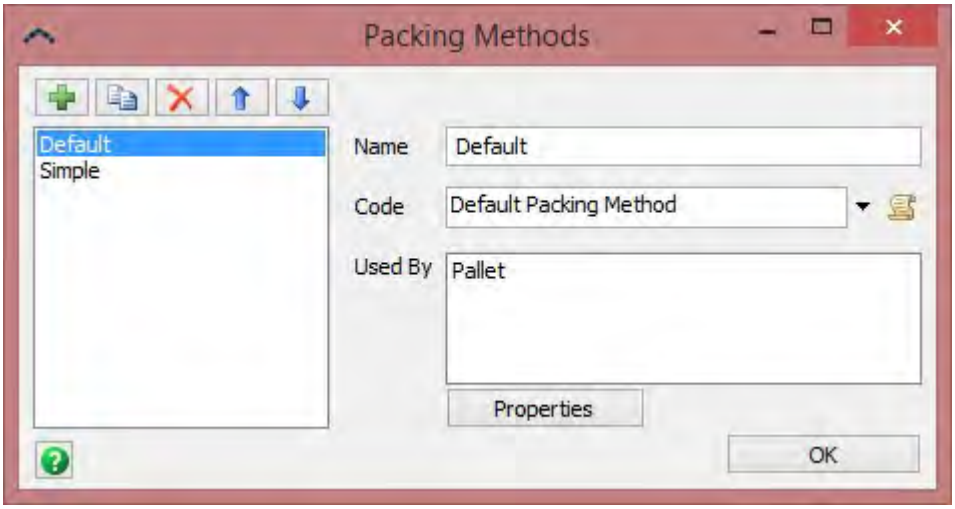
- Displays the help page.

- Adds the selected Flowitem to a User Library as either a Draggable Icon or an Auto-Install Component.

3D View - Displays the selected Flowitem. Here you can preview, resize and edit your Flowitem.

Properties - This button opens the currently selected Flowitem's Properties window. You can also access this window by double-clicking on the Flowitem in the 3D view.

Packing Methods



- Adds a new Packing Method.

- Duplicates the currently selected Packing Method.

- Removes the currently selected Packing Method.

- Moves the currently selected Packing Method up in the list.

- Moves the currently selected Packing Method down in the list.

Packing Methods List - Displays the list of all Packing Methods.

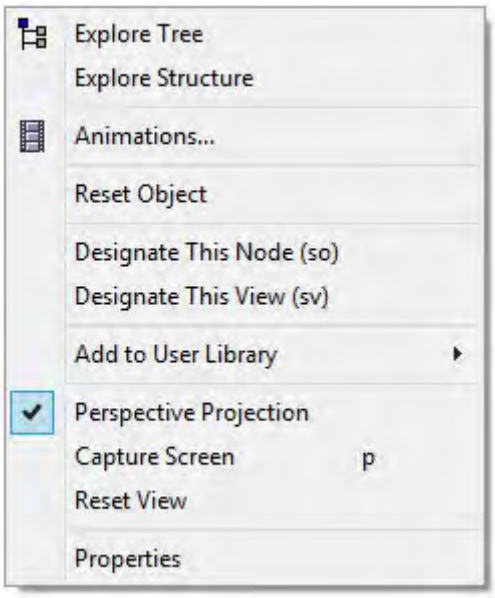
Name - Change the name of the Packing Method by entering the new name in this field.

Code - Allows you to pick from a list of preset packing methods or to edit the code manually in a Code Editor window.

Used By - Displays all Flowitems currently using the selected packing method.

Assigning Packing Methods: To assign a packing method to a Container Flowitem, open the Flowitem's properties window and set the Pack Contents of the Container Page. Alternatively, change the Pack Contents combo box displayed in the Quick Properties under the FlowItem section.

Right-Click Menu



Explore Tree - Opens a Tree Window and displays the Flowitem object in the Tree.

Explore Structure - This option brings up a tree window exploring the tree structure of the 3D window itself.

Animations... - Opens the Animation Creator to edit the object's animations.

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

Desginate This Node (so) - This designates the object as the "selected object", which can then be referenced in code as so(). You will usually use this option for writing code in the script console. There can only be one so() at any time.

Designate This View (sv) - This designates the window as the "selected view", which can then be referenced in code as sv(). You will usually use this option for writing code in the script console. There can only be one sv() at any time.

Add To User Library - Adds the Flowitem to a User Library as either a Draggable Icon or an Auto-Install Component.

Perspective Projection - Toggles the 3D view from Orthographic to Perspective view. For more information see the Perspective View page.

Capture Screen - Saves a PNG file of the current 3D view.

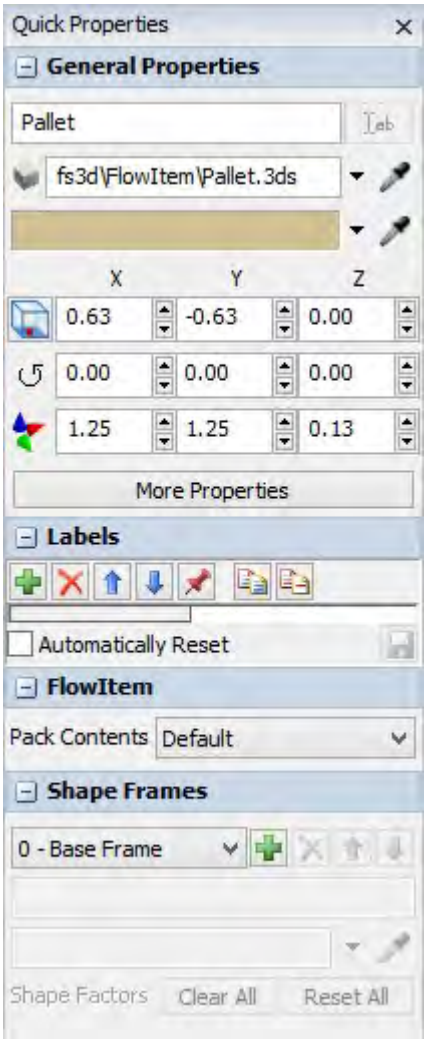
Reset View - Resets the view rotation and position to 0.

Properties - Opens the Flowitem's Properties window.

3D View - Displays the Flowitem in 3D.

Quick Properties

The Quick Properties will change to display the following panels:



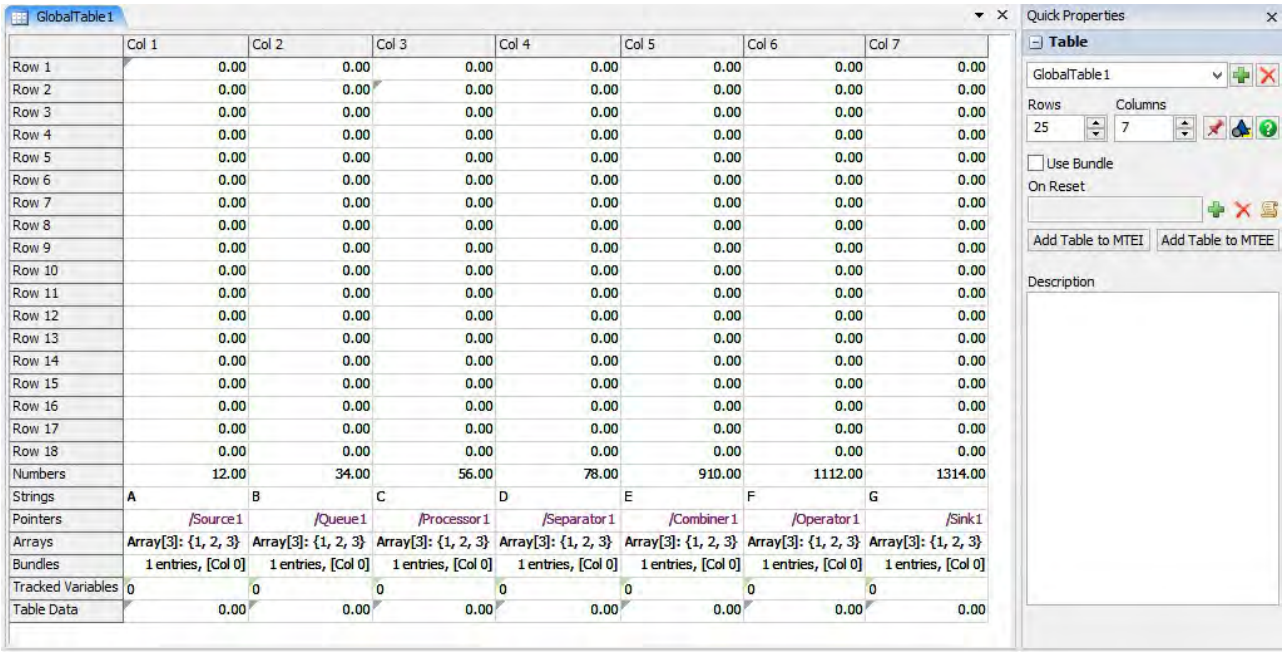
The same properties can also be changed by opening the Flowitem's Properties window.

General Properties - See the General page for more information.

Labels - Add, remove and edit labels for this Flowitem.

Flowitem - If the Flowitem is a Container Flowitem, this panel will appear and display the Packing Method. Shape Frames - See the Shape Frames page for more information.

Global Tables



	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7
Row 1	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Row 2	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Row 3	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Row 4	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Row 5	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Row 6	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Row 7	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Row 8	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Row 9	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Row 10	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Row 11	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Row 12	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Row 13	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Row 14	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Row 15	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Row 16	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Row 17	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Row 18	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Numbers	12.00	34.00	56.00	78.00	910.00	1112.00	1314.00
Strings	A	B	C	D	E	F	G
Pointers	/Source1	/Queue1	/Processor1	/Separator1	/Combiner1	/Operator1	/Sink1
Arrays	Array[3]: {1, 2, 3}	Array[3]: {1, 2, 3}	Array[3]: {1, 2, 3}	Array[3]: {1, 2, 3}	Array[3]: {1, 2, 3}	Array[3]: {1, 2, 3}	Array[3]: {1, 2, 3}
Bundles	1 entries, [Col 0]	1 entries, [Col 0]	1 entries, [Col 0]	1 entries, [Col 0]	1 entries, [Col 0]	1 entries, [Col 0]	1 entries, [Col 0]
Tracked Variables	0	0	0	0	0	0	0
Table Data	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Global Tables are accessed from the Toolbox. (View menu > Toolbox > Add > Global Table).

Global Tables can store numbers, strings, pointers, arrays, FlexScript, bundles or tracked variables. This data can be accessed by any object in the model using the Table class interface. A model may have any number of Global Tables.

Getting and Setting Table Data

Here are some code examples of getting and setting table data

```
Table myTable = Table("GlobalTable1");
myTable[1][1] = 5; myTable[2]["MyCol"] = "Test";
item.labelName = myTable[1][2];
double value = Table("GlobalTable1")[1]["Col 3"];
```

Editing the Table

To edit a cell in the table, click the desired cell and begin typing to overwrite all data in the cell, or double click on the cell to select the cell's contents. You can also right-click a cell and select Explore As Table or Explore As Tree and edit the cell's data through there. If the cell has array or bundle data, double-clicking the cell will open a new table window for editing.

Use the arrow keys to navigate between cells. Cells hold number data by default. You can change a cell's data type by right-clicking on the cell and selecting an option under the Assign Data menu. The right-click menu also has options for insert/deleting rows and columns, clearing cell data, and sorting by column.

Name Combobox - This is the table's name and has a list of all of the model's Global Tables. The name should be memorable and describe the table's function. The commands to read and write to global tables access them by name. You can view other Global Tables in this window by clicking the dropdown arrow next to the name.

 - Adds a Global Table to your model.

 - Removes the current Global Table from your model.

 - Pins the entire global table to a Dashboard as either a table of values, bar chart or line graph.

 - This button lets you add this table to a user library as either a draggable icon or as a component for automatic install. For more information, refer to the user library documentation.

Rows - This is the number of rows in the table.

Columns - This is the number of columns in the table.

Use Bundle - If this box is checked, the internal data type of the table will be changed to bundles. See the Bundle Data heading below.

On Reset Trigger - The On Reset trigger gives you the ability to define what happens to the table's data when the model is reset. This could be clearing all cell data, delete all rows, or some other custom defined behavior.

Add Table to MTEI - This buttons adds the table as a row in the Multiple Table Excel Import.

Add Table to MTEE - This buttons adds the table as a row in the Multiple Table Excel Export.

Description - This text box allows you to document your model by writing a description of the table. Here you can explain the purpose and organization of your table.

Note: A similar window to this is used when editing a label table from an object's labels or when editing a node or array as a table (accessible through the right-click menu in the Tree Window or Labels page). When editing a table or array that is not a Global Table, some options in the Quick Properties will not be available.

Bundle Data

If the "Use Bundles" checkbox is checked, your table will change internally to use the bundle data type. This means that your table will take significantly less memory. However, there are some limitations. All data in a column must be of one type, either number or string, and rows can no longer be named. The data type of a column can be changed by right-clicking in a cell in the column. All of the normal table functions will work with the bundle data type. Using bundles for your table data is highly recommended for large tables with over 500 rows or columns.

Cell Data Types

Cells in a table can store a variety of different types of data. These include:

- Numbers: Integer or floating point.
- Strings: Text like names or sentences.
- Pointers: References to other nodes or objects. The value displayed is the path to the node/object and will be purple.

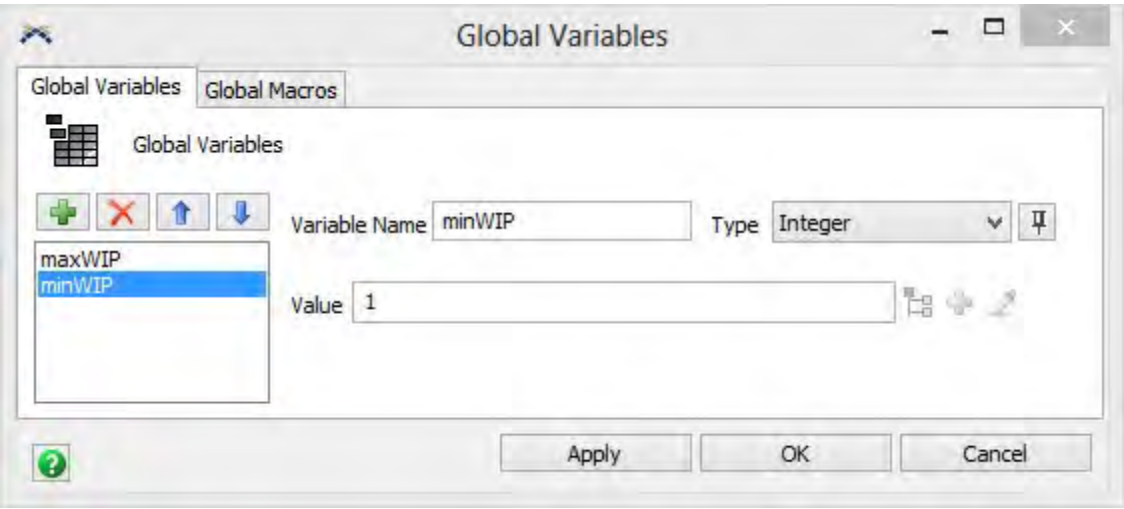
- Arrays: Arrays can have any number of elements. Each element can store a number, string, pointer or array.
- FlexScript: FlexScript toggled nodes allow you to write code that can that be evaluated using the evaluate() method. For example, Table("GlobalTable1").cell(1, 1).evaluate()
- Bundles: Data table that stores numbers and strings.
- Tracked Variables: Tracked variables are often used for statistic collection. For more information see Tracked Variables.

Additionally, a cell can also store a table of data. This can be done regardless of the data stored on the cell. To create a table on a cell, right click on the desired cell and select Explore As Table. This will open a new table window where you can add rows and columns to the cell. If a cell has a table, a small gray triangle will be drawn in the upper right corner of the cell.

Tracked Variable cells will display with a small green triangle in the upper right corner.

If a cell has array or bundle data, double-clicking on the cell will open a table view that can be edited.

Global Variables



Global Variables are accessed from the Toolbox. (View menu > Toolbox > Add > Modeling Logic > Global Variables).

The Global Variables window lets you create global variables and macro definitions that are accessible in FlexScript and C++. Once a variable has been created, you can get and set the value of that variable in a Code Editor window or Script Console.

Note: The value shown is the initial value of the variable. It is not the current value of the variable. The current value of the variable is stored in memory and can be seen by returning it in a script window or printing it from somewhere in code. The current value is not stored in the model tree anywhere. Global variable values are reset when you open the model, reset or compile.

There are 5 variable types you can use: integer, double, treenode, string, and Array. For the Array type, you can specify the size of the array and the initial value of each array element.

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 model malfunction and compile errors.

- Adds a new Global Variable.

- Removes the selected Global Variable.

- Reorder's the selected Global Variables Up or Down in the list.

Variable List - Displays all the model's Global Variables. Click to edit.

Variable Name - The name of the Global Variable. This is the name that will be used when writing code, ie setlabel(current, "wIP", maxWIP).

Type - Specify the Global Variable's type.

- Only available for Integer and Double types. Pins the global variable to a Dashboard as either the current value, bar chart or line graph.

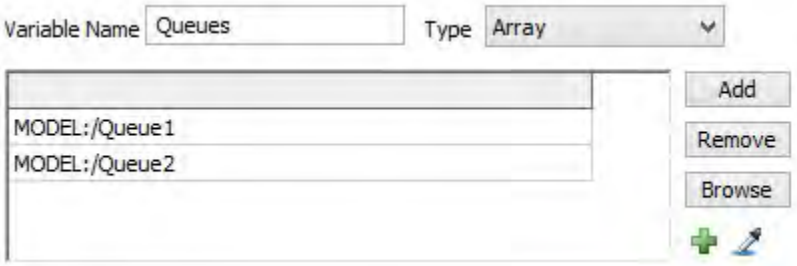
Value - The initial value of the Global Variable.

- Only available for Tree Node type. Opens a Tree Browse Dialog allowing you to select a node from the tree. Any node or object attribute may be selected.

- Only available for Tree Node type. Opens a popup allowing you to select an object in the model.

- Only available for Tree Node type. Click to enter "Sample" mode, then sample an object, node or attribute in your model.

If the Array type is chosen, the Global Variables window will display the following:



Add - Adds an empty value to the end of the array.

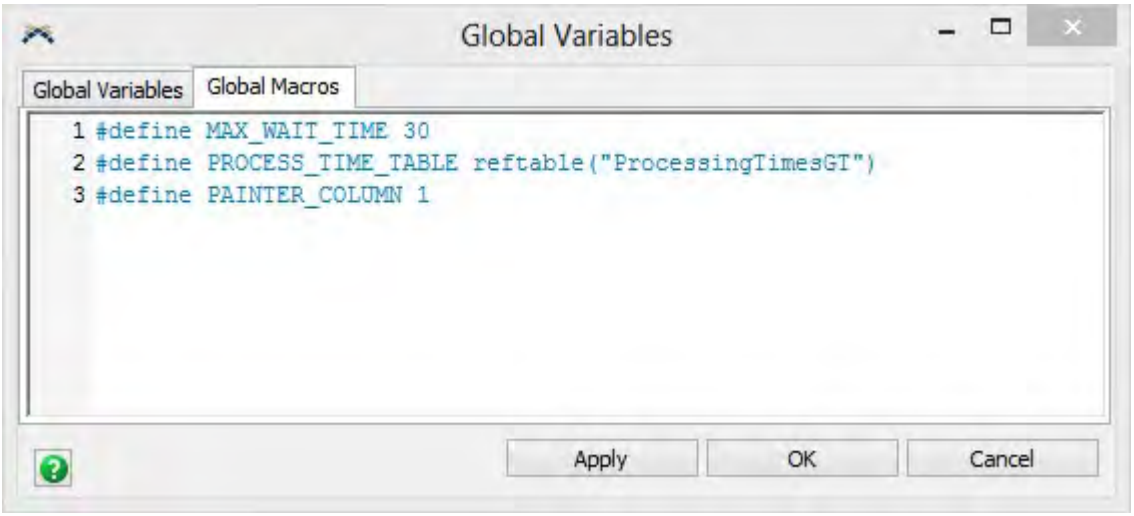
Remove - Removes the selected value.

Browse - Opens a Tree Browse Dialog allowing you to select a node from the tree. Any node or object attribute may be selected.

- Opens a popup allowing you to select an object(s) in the model. If a value is selected in the list, this sets that value, otherwise, it adds new values to the end of the array.

- Click to enter "Sample" mode, then sample an object, node or attribute in your model. If a value is selected in the list, this sets that value, otherwise, it adds a new value to the end of the array.

Global Macros



The global macros page lets you make macro definitions.

You can define macros using #define statements, as follows:

#define MAX_WAIT_TIME 30

#define PROCESS_TIME_TABLE reftable("ProcessingTimesGT");

#define PAINTER_COLUMN 1

Once you have made these definitions, you can use them in your code: gettablenum(PROCESS_TIME_TABLE, 1, PAINTER_COLUMN)

Note: Macro definitions do not end with a semicolon. If you put a semicolon in the macro definition, it may do things you don't expect it to do. Macros essentially replace the given text with the following specified value/text

throughout your code. If you have a semicolon at the end of the statement, you may end up with semicolons in incorrect places in your code.

Graphical User Interfaces Concepts

Topics

- Why Create GUIs?
- GUI Views
- Designing a GUI
- GUI Building Tips
- Working with the GUI Editor
- Traversing the Tree
- Attribute Lists
- Tree View - Viewing Attributes vs. View Structure
- Linking to the Model
- Copy an Existing GUI

Graphical User Interfaces (GUIs) are accessed from the Toolbox. (View menu > Toolbox > Add > Modeling Logic > Graphical User Interface).

GUIs allow you to create your own window interfaces for your model and its objects. A GUI can communicate with any object in the model, in any way you want.

GUIs are so common that there is a button in the toolbar to open such a window. 

In FlexSim, GUIs are stored as a node with sub-nodes in the tree. Each node represents a part of your GUI. The attributes in the object data of each of those nodes represent variables that affect how that part of your GUI works. When a GUI is opened, it creates a new node in VIEW:/active that is a copy.

Why Create GUIs?

- Not everyone knows how to navigate FlexSim. You can give others the ability to manipulate certain parts of your model without having to know FlexSim.
- Save time on extensive testing. GUIs can help you to change parameters quickly in your model during the testing process.
- GUIs look professional.

GUI Views

FlexSim GUIs are made up of building blocks called views. Views are windows that perform specialized roles and can be combined hierarchically. These views will allow you to view and manipulate data in the FlexSim tree structure. Since data takes many forms, there are many types of views.
Below is a list of the View Types available in FlexSim. The number next to each view type is the viewwindowtype. This number is a direct reference to the type of view and will not change in future releases of FlexSim.

For more information on GUI views, see the View Attributes Reference page.

View Types

Windows Common Controls

Static (or label) Button	103
Radiobutton	100
Checkbox	106
Edit	105
Trackbar	101
Combobox	122
Listbox	109
Tabcontrol	114
Scrollbar	115
Statusbar	104
Spinner	110
DateTimePicker	123
Treeview	125
	119

FlexSim Registered Controls

Dialog Panel	4 102
Groupbox	107
Table	5
Graph	6
3D View	2
Tree	0
IconGrid Script	7
HTML	8
	124

Creating a Modal Window: To create a modal window, use viewwindowtype 4 (Dialog) and add the subnode *FS_MODAL* into the object's style attribute.

Designing a GUI

There are whole books written about the philosphy of GUI design. The major topic, ease of use. Avoid the temptation of making a GUI window try to do too much. GUIs should be simple to navigate and focused in their function.

It is recommended to sketch out what you want a GUI window to look like before you open the GUI editor.

GUI Building Tips

It is important to have a good Control naming convention. A good short name will make referencing the object in code much easier.

Use Panel controls (invisible and not invisible) to group, move, and copy sections of your GUI.

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.

Another 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.
Practice, practice, 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.

Working with the GUI Editor

Here are some tips with working with the GUI editor:

- Think about not only how to arrange the controls in the GUI canvas (window editor) but also in the tree. Having familiarity with the Tree is a requirement for good GUI building.
- Moving controls in the tree can be done using the Edit Highlighted Object and Tree Movement fields in the GUI builder window.

- The attribute nodes of the GUI controls define how the control will look/behave. Each control has a default set of attribute nodes but others can be added from the Controls palette on the GUI builder window.

Traversing the Tree

@ : 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.

.. : This symbol tells the traversal to go to the current node's parent tree, or up one level.

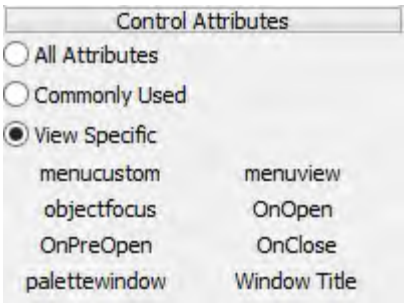
? :This symbol causes a recursive tree search for the subsequent name. For example, node("/?FlowItemBin", model()) will search for a node with the name FlowItemBin in the model tree. This returns the same node reference as an explicit path definition would: node("/Tools/FlowItemBin", model()).

$flexscript$: This syntax allows you to enter in custom flexscript code when accessing the path. For example, if you had a Global Macro #define MY_NODE_RANK 4 then having a path of *>objectfocus+/MY_NODE_RANK* would be read as *>objectfocus+/4*. You can execute any valid flexscript code. Another example, if you have an object in the model with a label called focus that contains "Tool4", then the following path *>objectfocus+/$getlabel(node("MyObject", model()), "focus")$* would be read as *>objectfocus+/Tool4*.

Below is a example table for Tree Referencing (Tree shown below table)

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.



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.

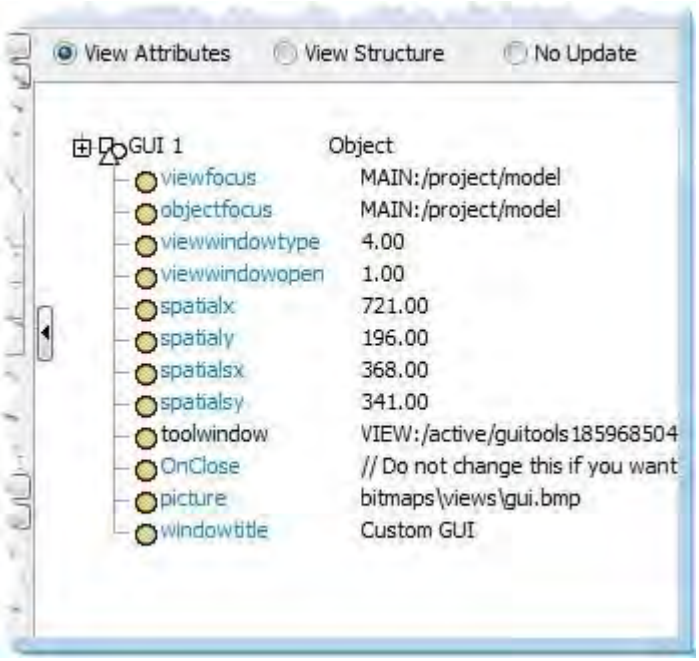
View Specific - This option is the default option. If it is selected, then the attribute list that is shown will be specific to the type of view, or view, that you are currently editing. If you click on a button view, an attribute list will appear that is specific to a button. This includes an OnPress 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.

To see a detailed list of attributes and their usage by control type, go to View Attributes Reference.

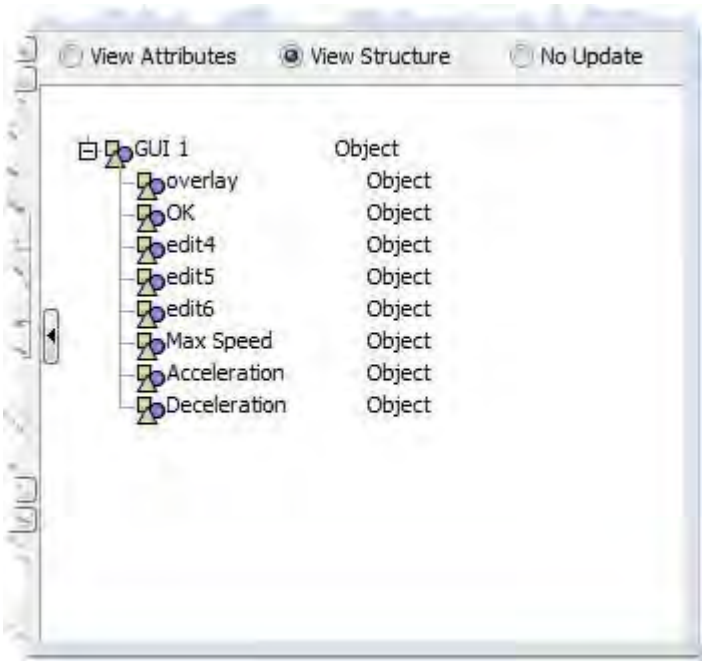
Tree View - Viewing Attributes vs. View Structure

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.

Linking to the Model

FlexSim uses two types of linking when tieing a GUI object into the model. Coldlinks, and hotlinks.

coldlinks

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.
Example: On the Processor tab of the Processor, the Max Content field is an example of a coldink.

hotlinks

A "hot" link, on the other hand, would continuously update its text field as the value in the model changes.
Example: On the statistics page of the Processor, the Content statistics are examples of a hotlink.
hotlinkx/coldlinkx

Similar to the above, but rather than using a special string of characters, FlexScript code is allowed to establish the link. The "c" passed in as argument 2 of the below example represents the owner object of the attribute.
Example: return(node("@>objectfocus+",c));

Copy an Existing GUI

If you see a window view or GUI that you want to copy verbatim from an existing window to your GUI there are two ways to copy.

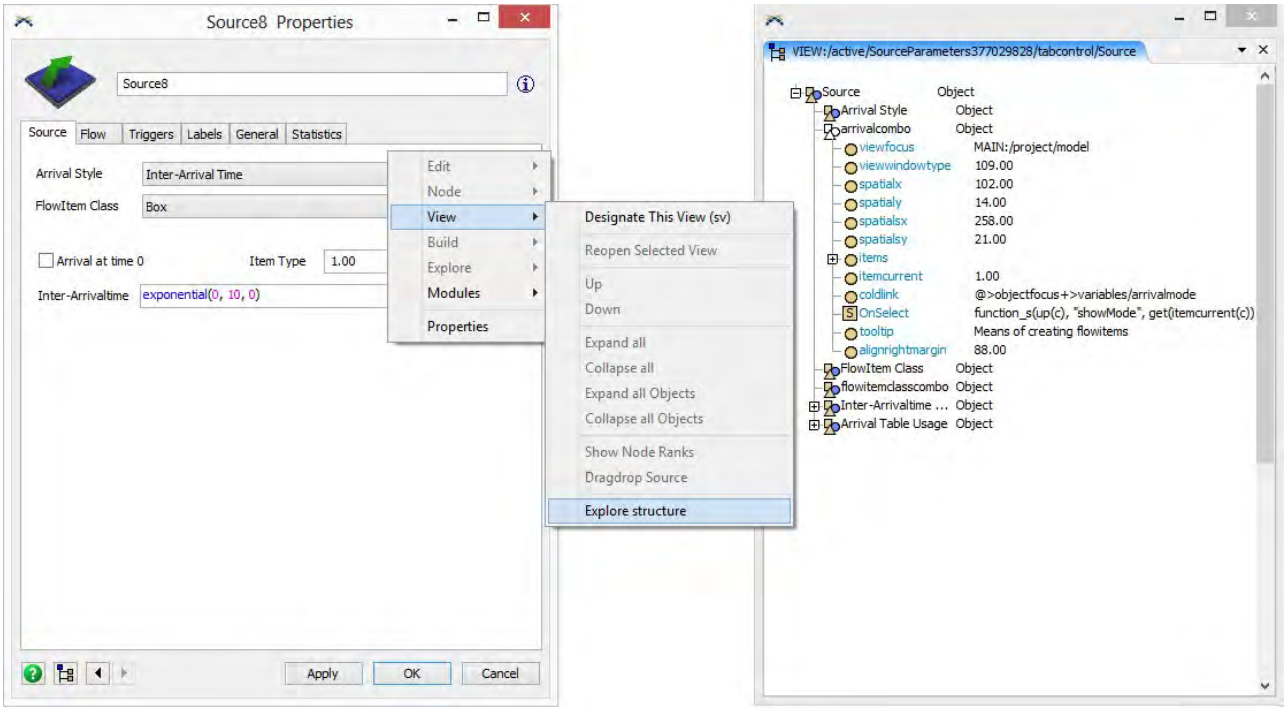
Copy from Tree

Find that view in the tree, right click on it and select "Edit > Copy"

Copy from Window

Open up the window you want to copy. Right click on it, and select "View > Explore Structure" from the popup 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. Right click on the view node and select "Edit > Copy"

Now 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 "Node > 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.



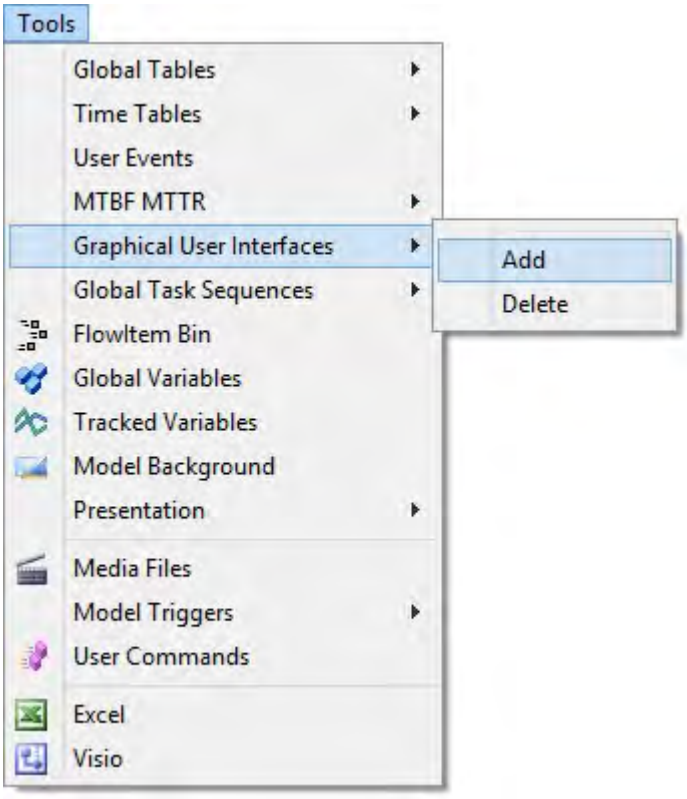
Graphical User Interfaces Example

Topics

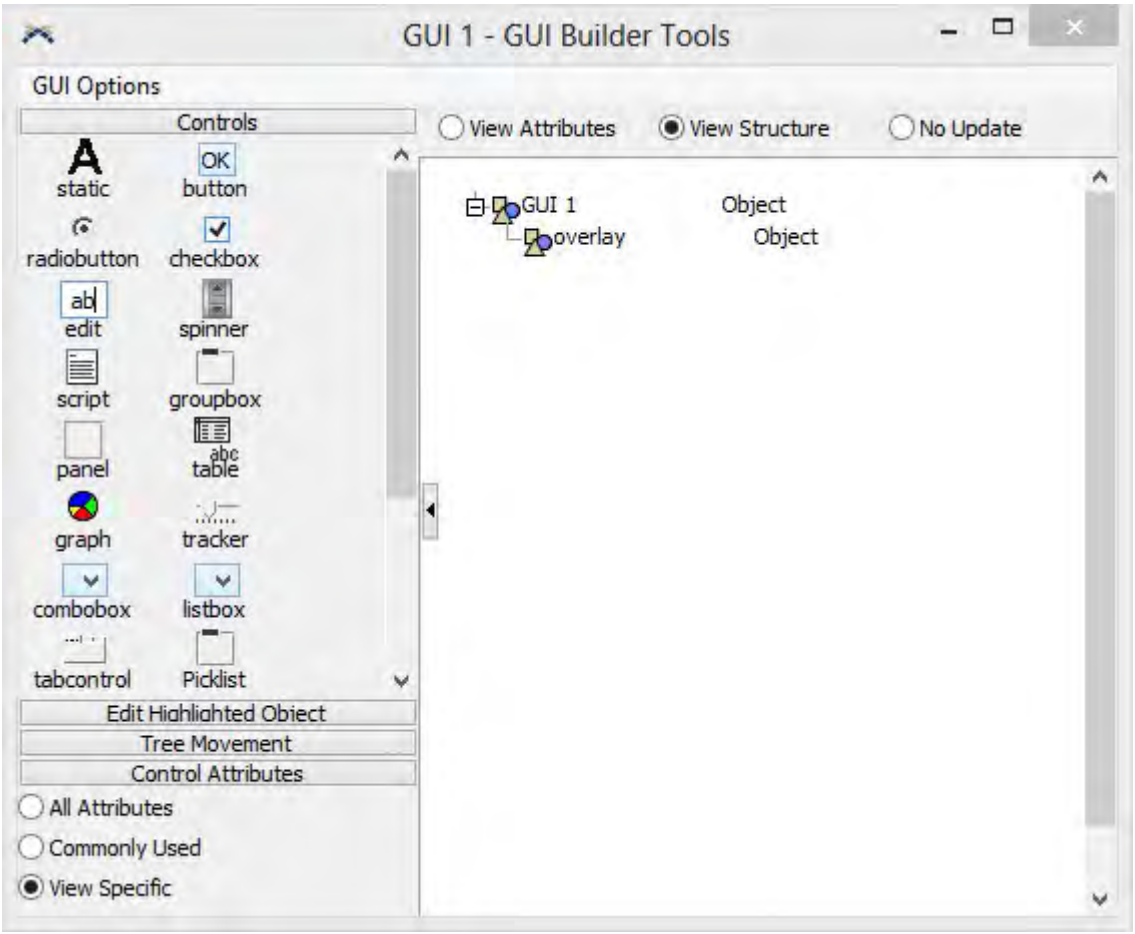
- Building a GUI
- Change View Names
- Link the Edit Views
- Direct Model Objects to This GUI

Building a GUI

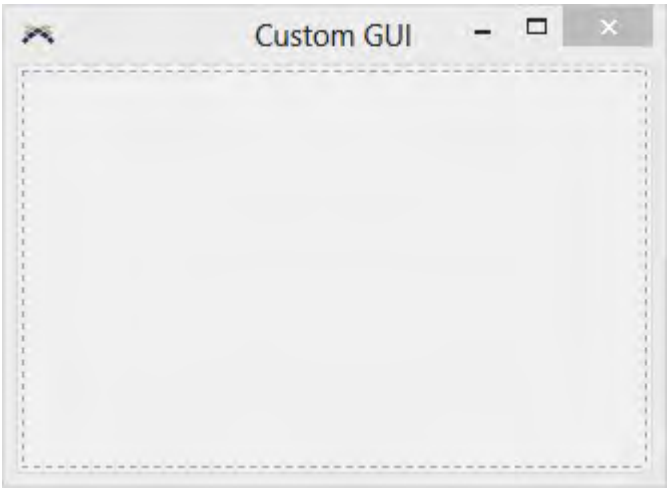
Add a GUI by going to the Tools Menu > Graphical User Interfaces > Add.



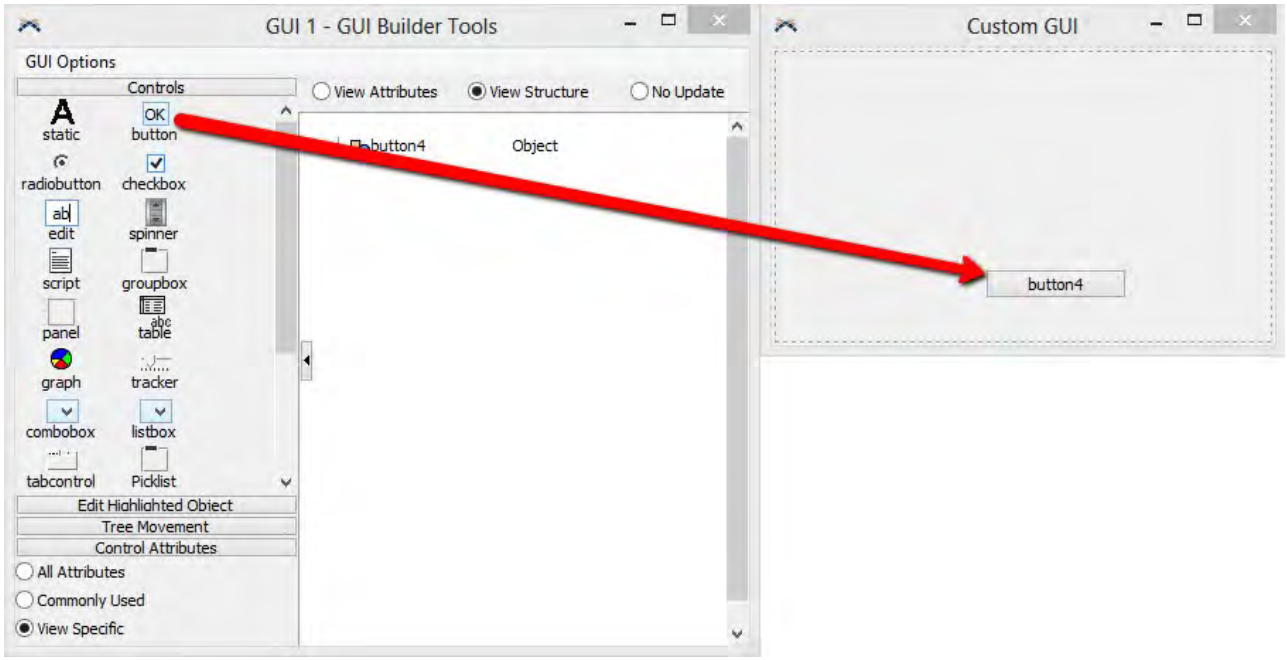
Once you have added a GUI, two windows will open. The window on the left is called the GUI builder. This window provides you with several tools for building your GUI.



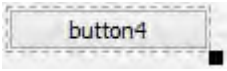
The window on the right is called the GUI canvas. This window shows what your GUI looks like. Initially it is blank. We will add GUI views to it in a drag-drop fashion from the GUI builder window.



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.



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.



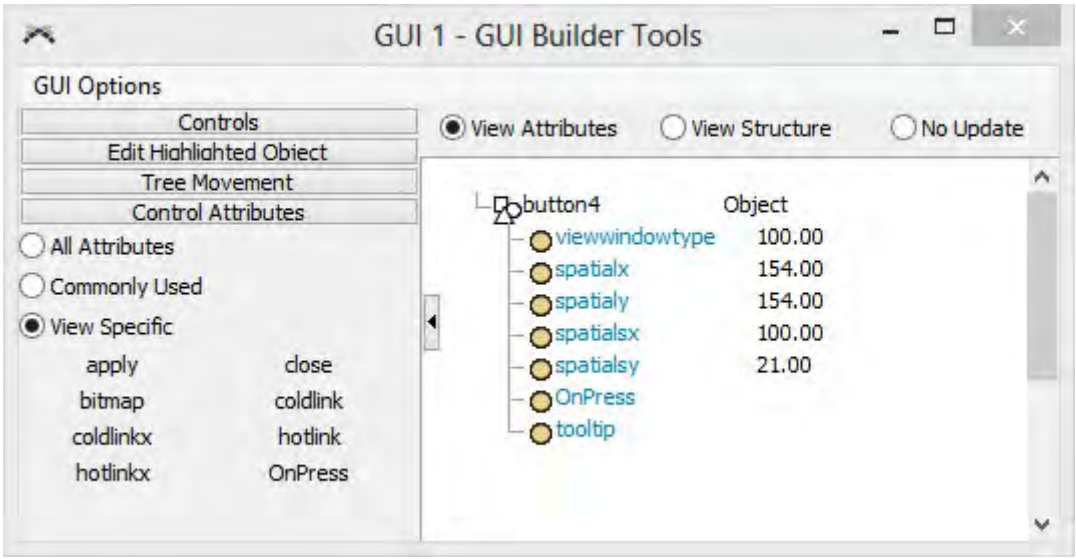
To move the button, just click and drag the button to the location you want it at. To change the size of the button, click and drag the black square.

Now add three static views and three edit views onto the GUI canvas, as shown below.

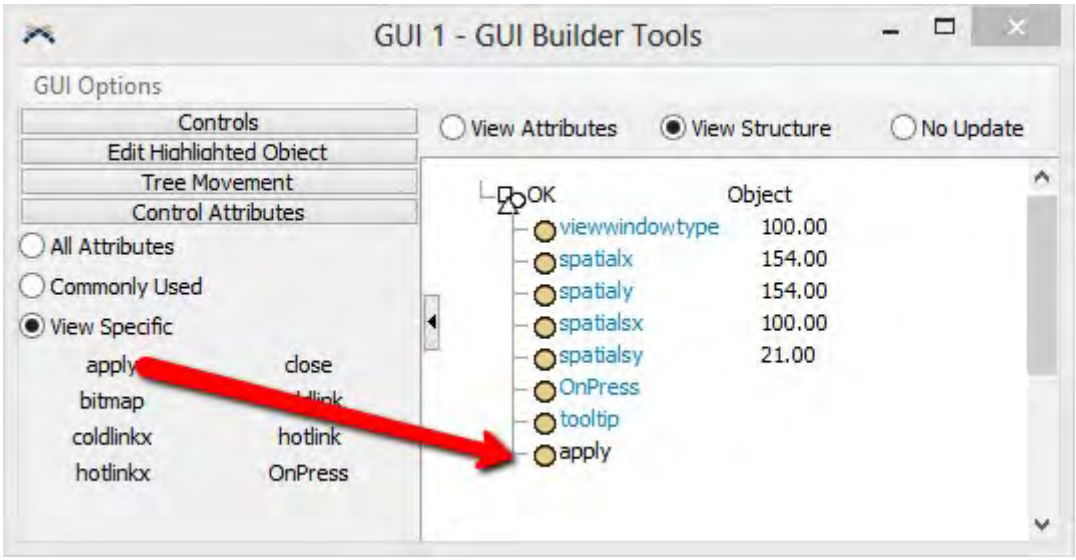
Change View Names

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.



Click on the button name ("button4" in our case) and rename it to "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.

Alternative Method - Alternatively, instead of adding the apply and close attributes, you could add the following code the the OnPress attribute of the button:

```
applylinks(ownerview(c)); postclosewindowmessage(ownerview(c));
```

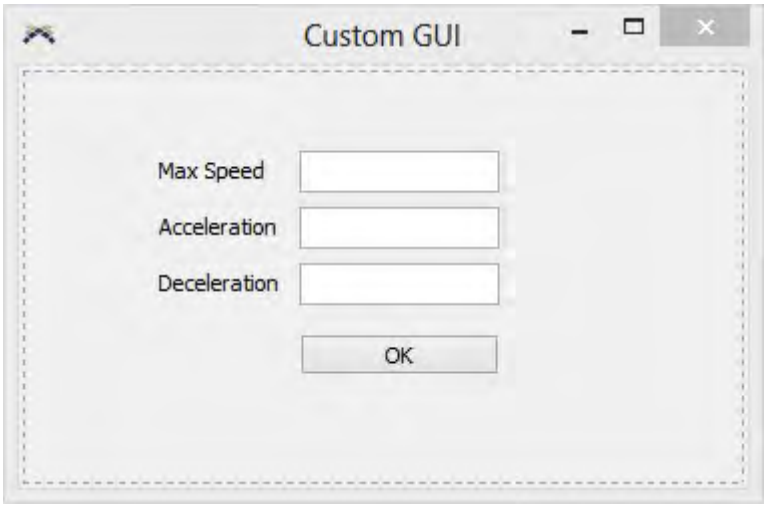
This code would have the same effect as the apply and close attributes.

Now click on the first static view. Again, the static view and its attributes should appear in the tree view of the GUI builder. For the name of the first static view, enter "Max Speed". You will not see the name change on the static view until the view is refreshed. Now click on the second static view and set its name to "Acceleration". Then click on the last static view and change its name to "Deceleration".

Now that you have made a few changes to the GUI views we will 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

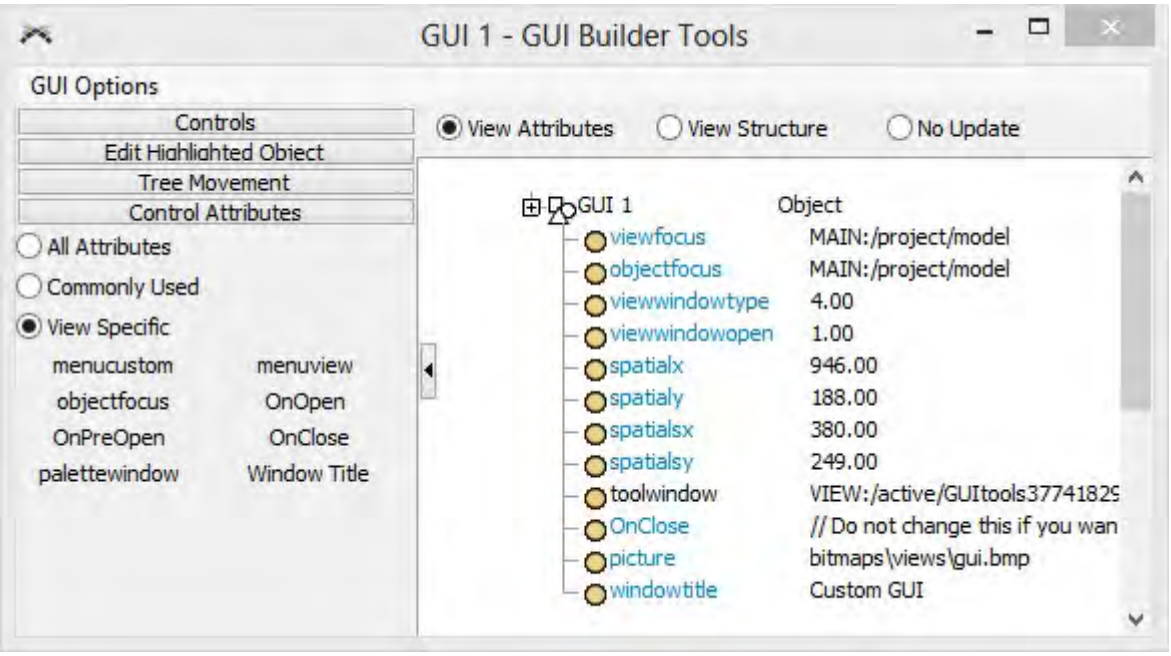
mode to regular viewing mode. Now your window should look like a normal window without the dotted lines around it.

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.



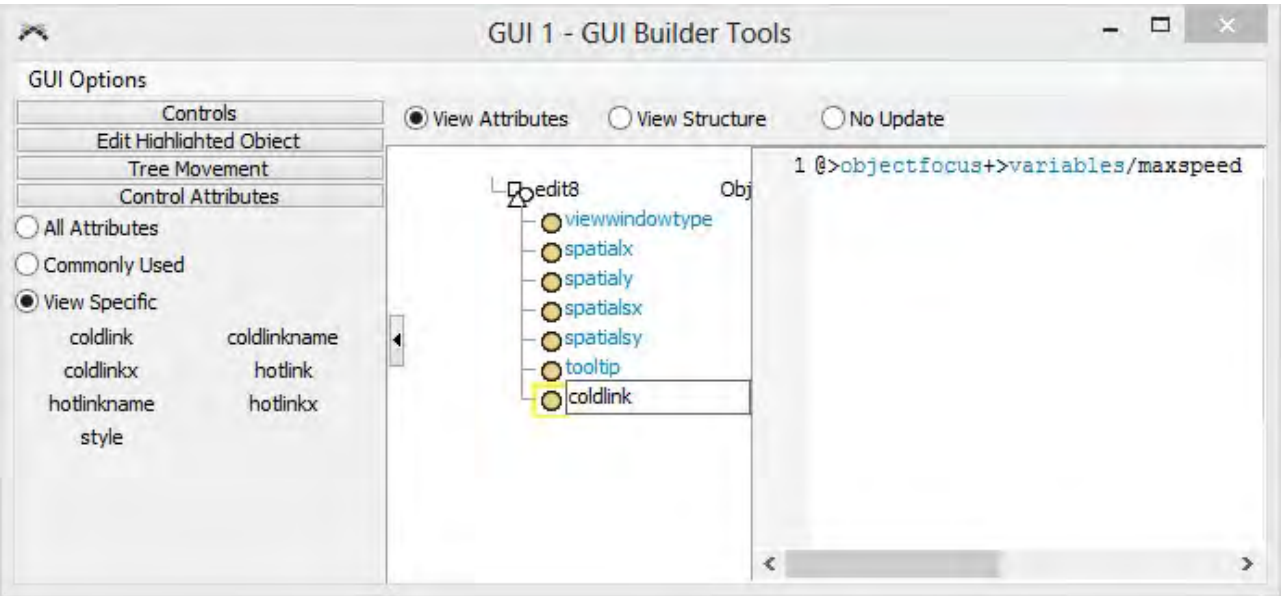
Link the Edit Views

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 to 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



Links should be made to a node that contains some desired value. Sometimes linking directly to the object property node is undesireable, such as code nodes. In this case, link to a label and use the label value as part of the code property (use executestring()).

The coldlink/hotlink 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.

The coldlink you have specified does the following:

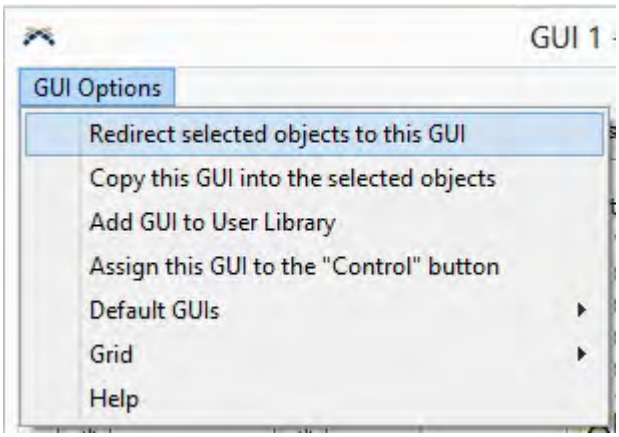
1. Starting at the coldlink node, go to its ownerview, or the main GUI window (@).
2. From there, go into its attribute tree and find the attribute named objectfocus (>objectfocus).
3. Interpret the text of the objectfocus node as a path to a node, and go to that node (+). Remember that when we open 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 "GUI Options > Redirect Selected Objects to this GUI".



This will redirect all selected objects in the model to use this GUI instead of their usual properties 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 properties window.

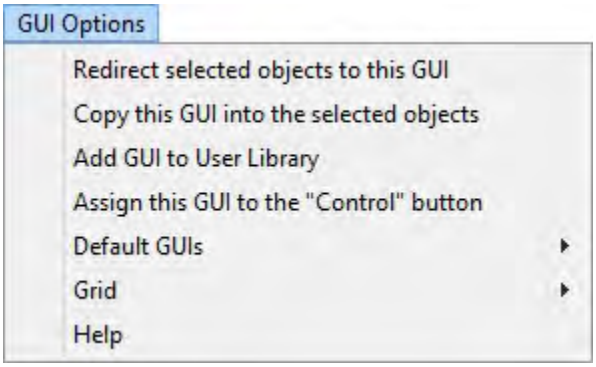
Make some changes to the max speed, acceleration and deceleration values, then press the OK button to apply those changes. Open the window again to verify that your changes were applied correctly.

Graphical User Interfaces Reference

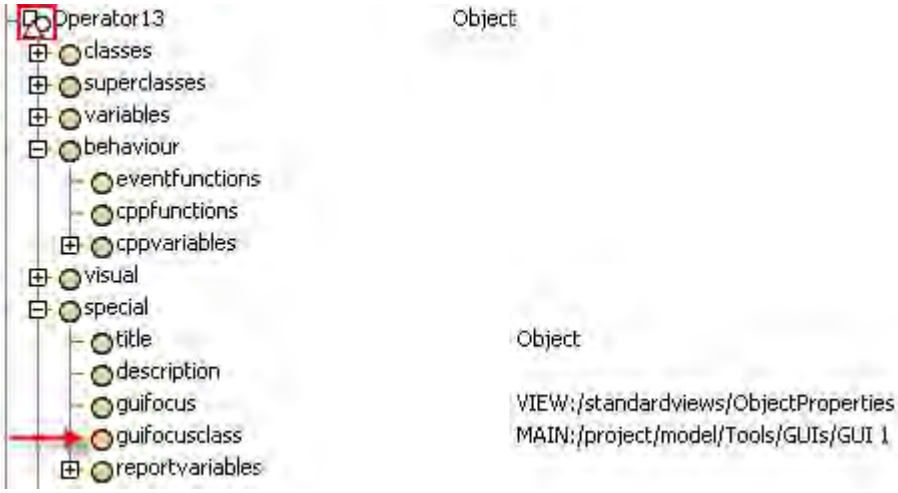
For information on using the Graphical User Interface, see the Example page.

GUI Builder Menu

The GUI builder's edit menu allows you to do several operations with the GUI once you have created it.

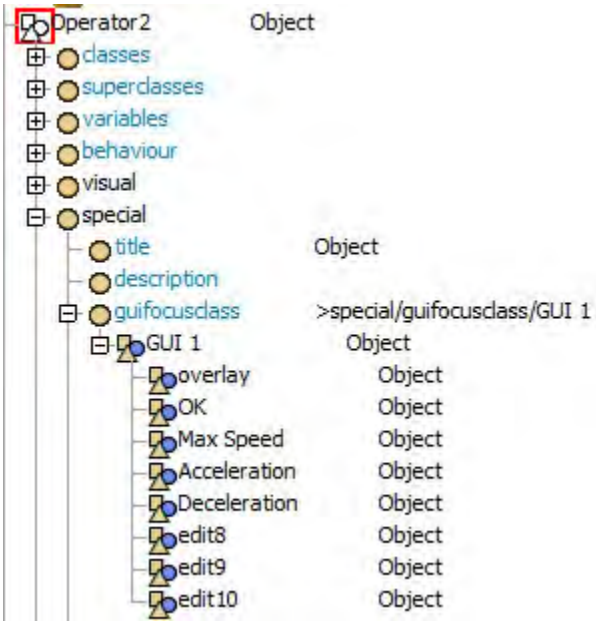


Redirect Selected Objects to this GUI - This option will redirect the guifocusclass attribute of every selected object in the model to point to this GUI. To illustrate this, explore the model tree, and find the operator that you redirected to the GUI you created. Expand its attribute tree and the expand 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 Properties 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 the GUI we created in our example. 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 GUI to User Library - This option will add the GUI to the selected user library in the library icon grid. For more information on user libraries, refer to the user library documentation.

Assign this GUI to the "Control" button - 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 GUI button. Its exclusive purpose is to allow model builders to define custom GUIs for controlling models and their properties without having to change FlexSim's view tree. If this button is no longer available, you may add it to your user toolbar through the Global Preferences Window.

Default GUIs - This sub-menu lets you also edit how other buttons on FlexSim's main toolbar operates. The 3D View button on the toolbar can be customized to open custom GUIs that you have created. By selecting "Make this GUI the Default 3D View GUI" you can cause the 3D View buttons on the main toolbar to open your GUI. Select "Reset Default 3D View GUI to Original" to cause the 3D View buttons to revert back to their original GUIs. Note that making a custom GUI for the 3D View buttons only applies to the model you are editing. When you open another model, the 3D View buttons will reset to the original default view.

Grid - This sub-menu lets you set whether the GUI canvas snaps to grid and what that grid size is.

List Key Concepts

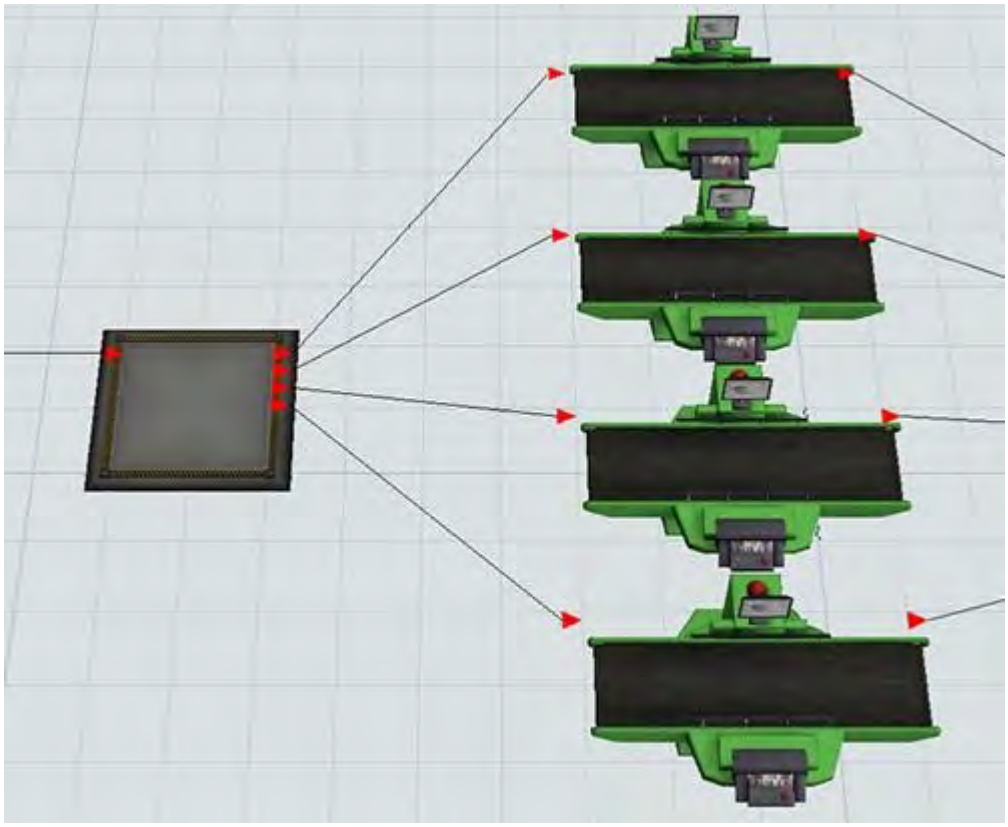
Lists are a powerful tool that can have many applications in FlexSim. At the most basic level, a List is just that: a list of values. Each value in a list may be either a number, a string, or a reference to an object in the simulation model. Lists also harness the expressive power of SQL for searching, filtering and prioritizing the values on the List. Entries in a List can have user-defined fields. These fields essentially make the List into a dynamic database table. The List can then be filtered and prioritized using SQL queries, as if you were querying a standard database table.

Topics

- Why Use Lists?
- Concepts and Terminology
- List Types

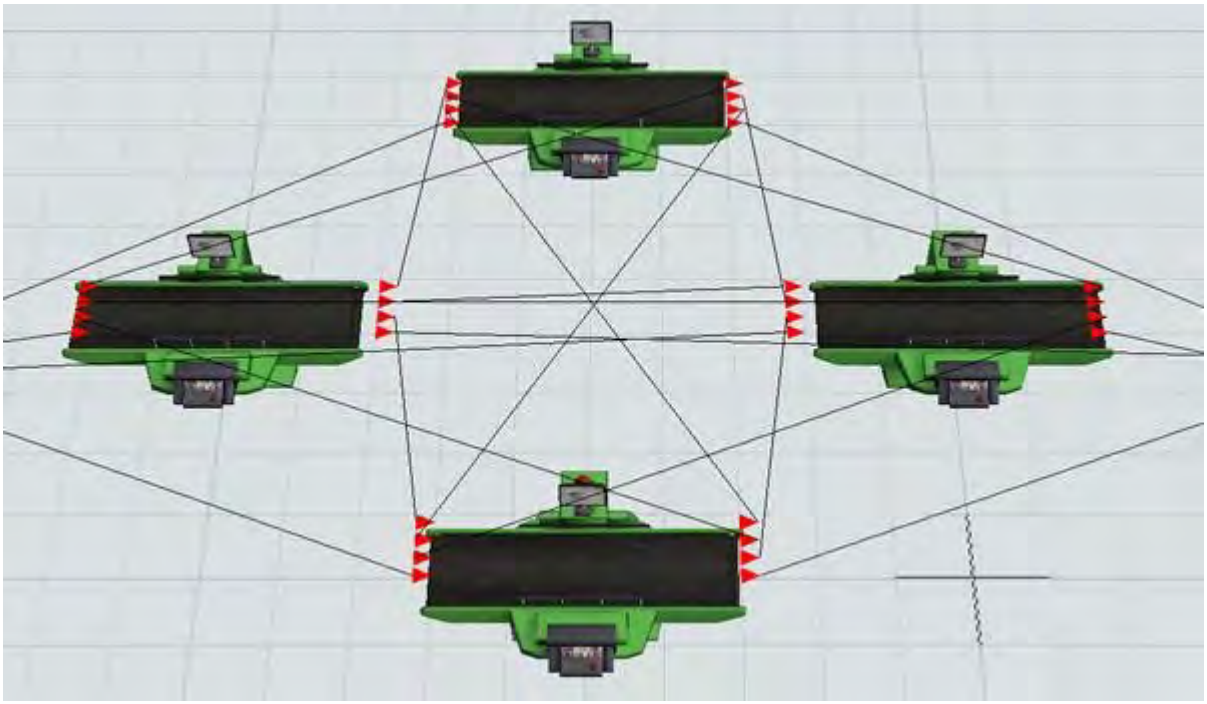
Why Use Lists?

There are many applications for Lists in FlexSim. A primary reason for using them is to separate the operations of search, filter and prioritization away from the physical layout, connections, etc. of a model. Take for example the problem of Fixed Resource routing/pulling. In some scenarios, simply connecting objects together and implementing port-based send-to and/or pull strategies works quite well. The following image shows a simple one-to-many routing operation.



This scenario is pretty straight forward. You send to a downstream station based on some basic rules.

However, some situations don't lend themselves as easily to simple routing. For example, in many job shop operations, the routing becomes a many-to-many problem. This can make using port connections more difficult. The number of needed port connections grows quadratically with the number of processing stations. Proper port ranking becomes an issue. Also, since search order in a pull operation is primarily by port and secondarily by the items in the upstream queues, prioritization is difficult, especially when handling priority ties. The following image demonstrates a much more complex routing problem.



Fixed Resource routing is just one example among many where using the structure of a model as the back-bone for searching can sometimes be more difficulty than help. It could also apply to Task Executer dispatching. Often Task Executers are grouped together by a connection to their Dispatcher, or "team leader." As such, a given Task Executer usually only has one object that you dispatch Task Sequences through. However, in real life scenarios, the set of valid Task Executers to whom you want to dispatch a task might not be properly represented using a strict separation into teams. One task could potentially be performed by multiple teams, or perhaps a single member of a team might be able to perform one set of tasks while another cannot. Task Executers might each have a skill set, and/or pay grade that may not be exactly correlated with the team that the Task Executer is assigned to. The lines of team vs capability can become very blurry. In these scenarios, using the Dispatcher team groupings for dispatching logic can be difficult and error prone.

Enter Lists

Using Lists can make these complex problems much simpler. For example, in the Fixed Resource routing problem, if when an item is finished at one station, you just put that item onto a global list of "items ready to go to their next station," then when a station becomes ready to pull its next item, it can just query that list and grab the best item it finds. It can use SQL to easily filter and prioritize the item that it wants. For the Task Executer dispatching problem, if all Task Executers in a model were to put themselves onto a global list of "Available Task Executers" when they become available to take up new work, then when an object has a Task Sequence that needs to be done, it can query the list to find the best Task Executer to dispatch the Task Sequence to. Filtering and prioritizing becomes a simple expression in an SQL query.

Concepts and Terminology

When talking about Lists, we use several important terms, as follows:

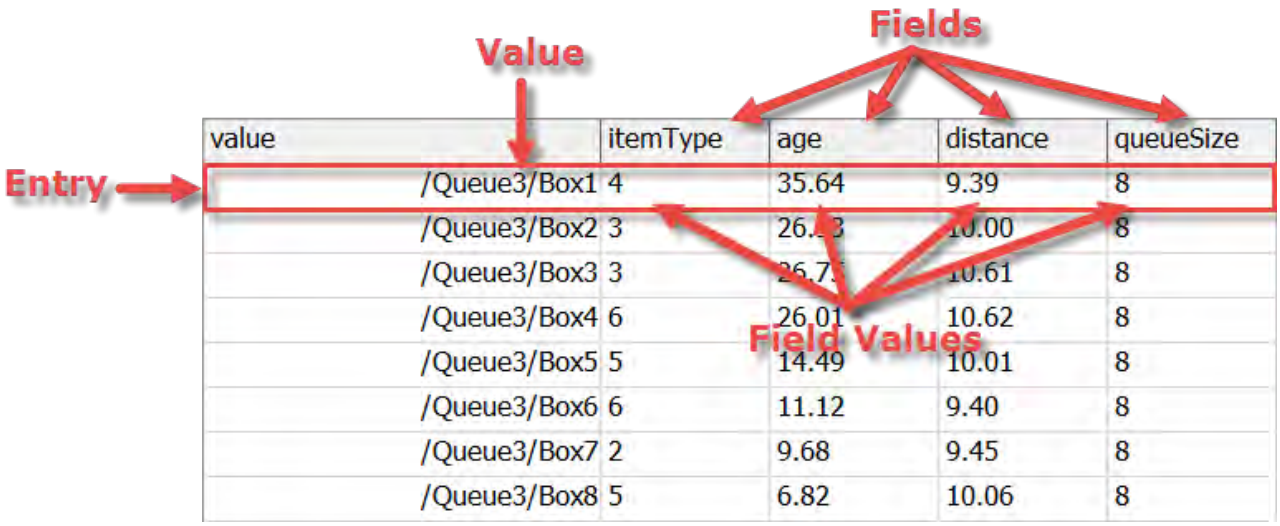
- Value - A value in a List is the primary value associated with a given entry. As mentioned previously, the value can either be a number, a string, an object, or node reference.

- Entry - An entry represents the entire record that is placed on or removed from the list. An entry contains its primary value as well as all of the required field values. Note that the entry is not the same as the value in that, in some scenarios you may have in a List multiple entries with the same value. If you were to think of a list like a database table, an entry essentially represents the row (or entry) of the table.

- Field - A field designates a value to be stored with each entry. This is additional data associated with the entry's value, and can be used for filtering and prioritizing the list via SQL queries. In database table terms, this would be the column (or field) in the database table.

- Field Value A field value is the specific value associated with a given field for an entry. Field values are similar, but not exactly the same as THE value of an entry, defined above. While both field values and primary values can be used interchangeably in querying Lists, the primary value of the entry is special in that it is the value that is retrieved as part of a pull request. On the other hand, field values are only used in querying the list in a pull request. Note that in some cases we may use the terms field and field value interchangeably.

The following diagram shows example entries in a list. The first entry has a value of /Queue3/Box1 (a reference to a box in a Queue), with field values for the box's itemType, age, distance and queueSize.



value	itemType	age	distance	queueSize
/Queue3/Box1	4	35.64	9.39	8
/Queue3/Box2	3	26.[illegible]3	10.00	8
/Queue3/Box3	3	26.7[illegible]	10.61	8
/Queue3/Box4	6	26.01	10.62	8
/Queue3/Box5	5	14.49	10.01	8
/Queue3/Box6	6	11.12	9.40	8
/Queue3/Box7	2	9.68	9.45	8
/Queue3/Box8	5	6.82	10.06	8

- Push - The verb push is used for adding something to a list.

- Pull - The verb pull is used for removing something from a list.

Push/Pull vs Add/Remove

You might ask why we don't just call push and pull operations *add* and *remove*. We name them differently because the pull operation represents much more than just the removal of a value. A pull may include a search of the list based on filtering criteria and/or a prioritization rule. Also, a pull operation, if it does not immediately find a value to remove, will create a back order on the list. Once a matching value is then pushed onto the list, the back order will be fulfilled and the value will be immediately removed. Additionally, the pull operation includes a return value, namely the value that was pulled from the list. So a pull operation involves much more than just the removal of a value from the list. Hence we use the specialized terms *push* and *pull* to describe these operations.

- Query - When you pull something from a list, you can optionally include a query to filter or prioritize what you want pulled from the list. This is the where the expressive power of SQL can be used. For example, using the previous list diagram, let's say you were to pull from the list using the query:

  ```
  WHERE itemType < 4 ORDER BY distance ASC
  ```

  This would tell the pull request to only pull boxes whose itemType is less than 4 (WHERE itemType < 4), and then to find among those filtered boxes the box with the smallest distance (ORDER BY distance in ASCending order). In this case, the value /Queue3/Box7 would be pulled from the list.

- Puller - Puller designates the thing that is pulling from the list in a pull operation. Usually the field values that are used in a pull request retrieve data associated only with the primary value (value-only fields), things such as labels on the pushed value. However, when a pull request is made, you can optionally include a puller object or value in the pull request. Some field values may be dependent on who or what the puller is. For example, refer back to the distance field in the previous image. This field was created as a puller-dependent field. It represents the distance from the value (/Queue3/Box1) to some puller object that is included in a pull request.

- Back Order - When a pull request is not completely fulfilled, i.e. the things you want to pull have not yet been pushed to the list, then by default a back order will be created for the list. The back order stores needed data associated with the pull request, such as the puller and the pull query. When entries are subsequently pushed onto the list, pending back orders will be processed to see if those pushed entries fulfill the back orders. Once fulfilled, back orders will be removed from the back order queue.

- Partition - Lists also have the feature of being able to be separated into "partitions." This essentially turns the List into multiple separate lists of values, or partitions. Each partition is uniquely identified by its partition ID.

- Partition ID - Each partition of a List has a partition ID associated with it. This value is what uniquely distinguishes a partition from other partitions. When you push or pull from a List, you can define the partition ID to push/pull from, and the List push/pull logic will only reference entries and back orders that are in the partition with that partition ID. The partition ID can be a number, a string, or a reference to an object or node in the tree. Partitions are dynamic in that, if you push a value onto a partition that does not yet exist, the partition will be created automatically.

Lists are Logical Constructs

In using lists, it's important to remember that they are purely logical constructs. For example, you may have a queue object in your model push an item onto a list. This does not put the item physically onto the list. The 3D item object remains in the queue. Instead, a reference, or pointer, to that item is added to the list. The logical nature of lists is in fact one of their advantages, in that you can separate the problem of search and prioritization from the physical structure of the model. In other words, you can have items physically located in one or more areas/objects in the model, while logically searching those items through a central mechanism, namely by putting references to those items (or "pushing the items") onto the list.

List Types

•

When you add Global Lists to a model, you add specific types of lists. Some types include:

- Item List - A list of flowitems. Items are pushed onto the list, and Fixed Resources usually pull from it. Fixed Resource List - A list of Fixed Resources. Fixed Resources are pushed onto the list, and flowitems usually pull from it.
- Task Sequence List - A list of Task Sequences. Task Sequences are pushed onto the list, and Task Executers usually pull from it.
- Task Executer List - A list of Task Executers. Task Executers are pushed onto the list, and Task Sequences usually pull from it.

List Types Aren't THAT Critical
While using the different types of Lists properly can make model building easier, it's important to note that the delineation of list types is more semantic than functional. At its core, a List is just a List. The user interface for adding fields, etc., will be customized based on the list's type, however, there is nothing to stop you from pushing a Fixed Resource onto an Item List, or to have a flowitem (as opposed to a Fixed Resource) pull itself or another item from an Item List. I say this not to encourage you to put Fixed Resources onto Item Lists (I certainly do not advise that, especially if field values depend on it being a flowitem), but just to point out that, fundamentally, a List is a List is a List.

When to Use Which Type

The choice of which List type to use in which situation is essentially up to you. If you are doing Fixed Resource routing, you should either use an Item List or a Fixed Resource List, but usually not both. When you use an Item List, items are pushed onto the List and Fixed Resources choose which items they want to process next by pulling them from the List. On the other hand, when you use a Fixed Resource List, Fixed Resources push themselves onto the list, while the items choose the Fixed Resources to be sent to by pulling them from the list. The main difference is in who is doing the "querying" or pulling. Since the SQL query is used when you pull from the list, the puller is the one who gets to filter and prioritize the entries on the List easiest.

However, when a pull request is made, the puller and its data can be referenced as part of the query (like in the distance example mentioned above). If you decide to use an Item List, for example, then when a Fixed Resource pulls from the List, the query can access information both on the items in the List as well as on the Fixed Resource itself, the puller. And the same rule applies if you were to use a Fixed Resource List, with flowitems pulling from the List. Thus the query can be a way of dynamically matching puller to value, instead of only getting data about the values in the List. What all of this means is that, for the most part, an Item List can have full functional parity with a Fixed Resource List, and a Task Sequence List can have full functional parity with a Task Executer List. So, in the end, which type of List you decide to use is really a decision of intuitive feel. Use the type that makes most sense.

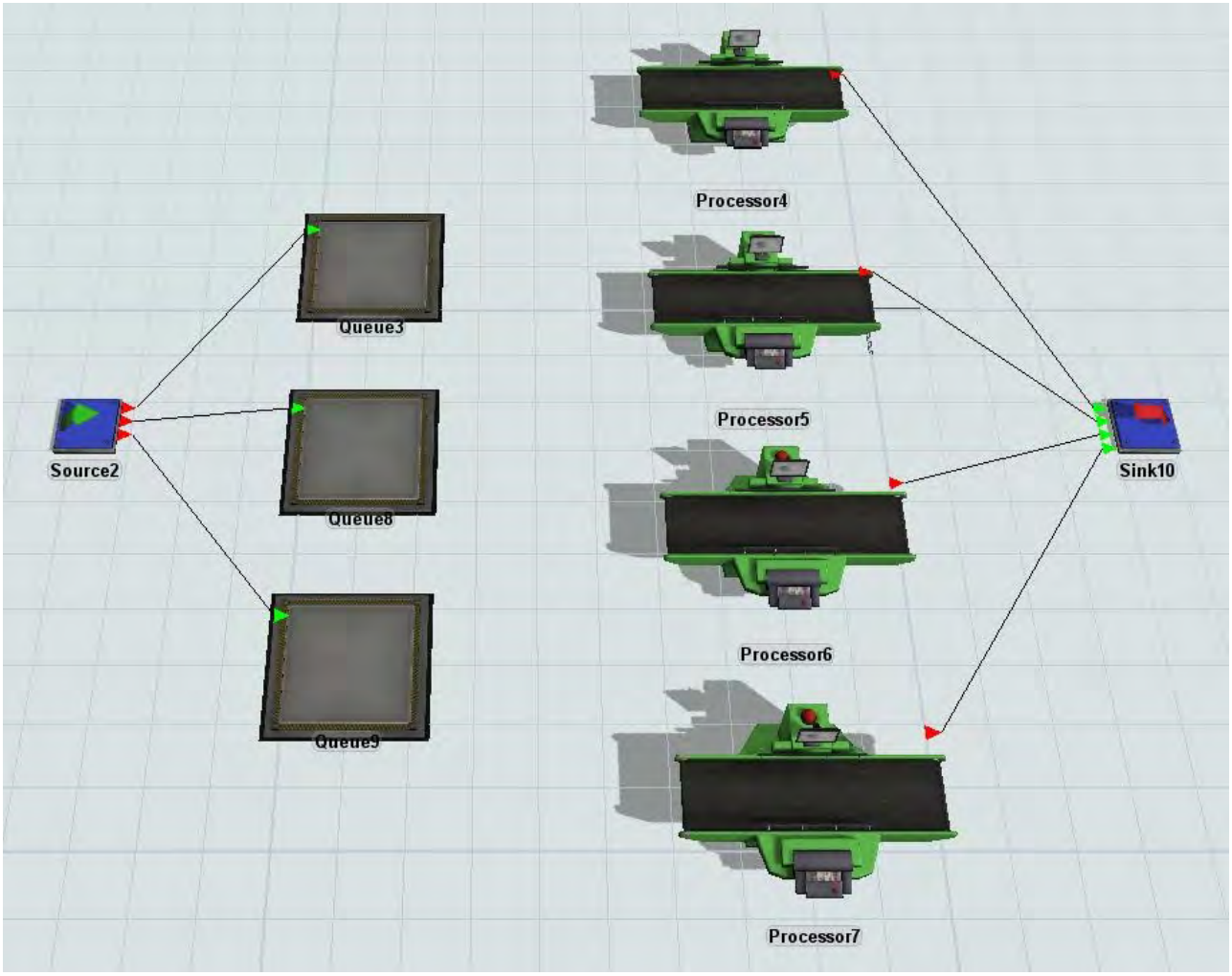
Global Lists vs. Local Lists

When using lists with standard 3D objects in a model, you will usually use global lists. These are lists that you add, remove, and edit through the Toolbox. However, if you use the Process Flow module in building your model, Process Flows can contain lists that are local to the Process Flow. Any lists that are only needed locally within the Process Flow can use internal lists, whereas lists that coordinate functionality

- between different Process Flows, or between a Process Flow and standard 3D object functionality, would use global lists.

Connectionless Routing

Here we will build a model that uses lists for basic many-to-many routing of items. The layout for this model is shown in the following image.

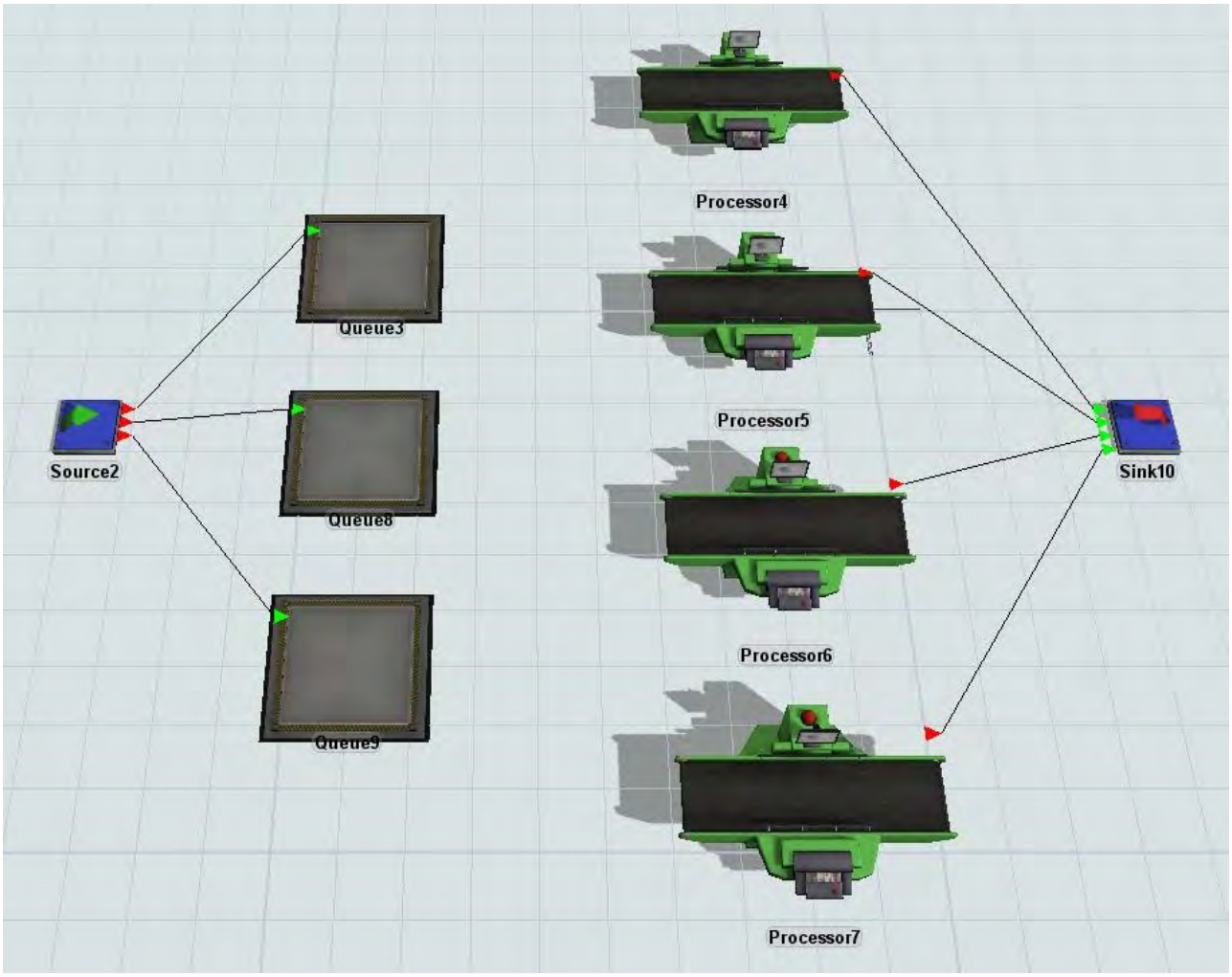


How It Will Work

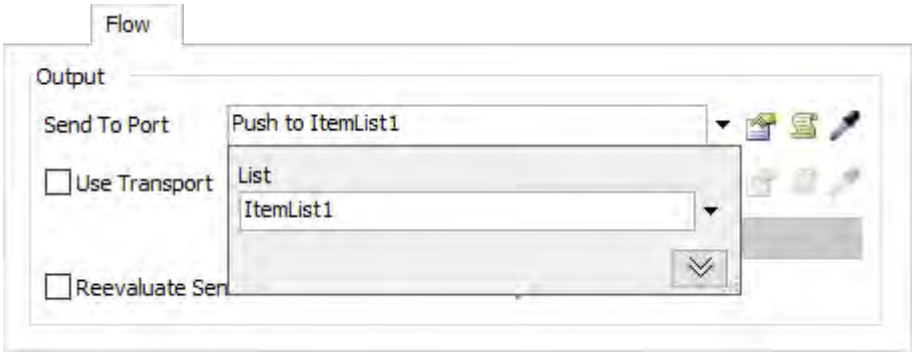
In this model, items will be sent from three queues to one of four processors. We will use an item list. The queues will push the items onto the list, and the processors will pull them from the list and process them. Initially we will implement a simple pull, but later we will experiment with filtering and prioritizing.

Modeling Steps

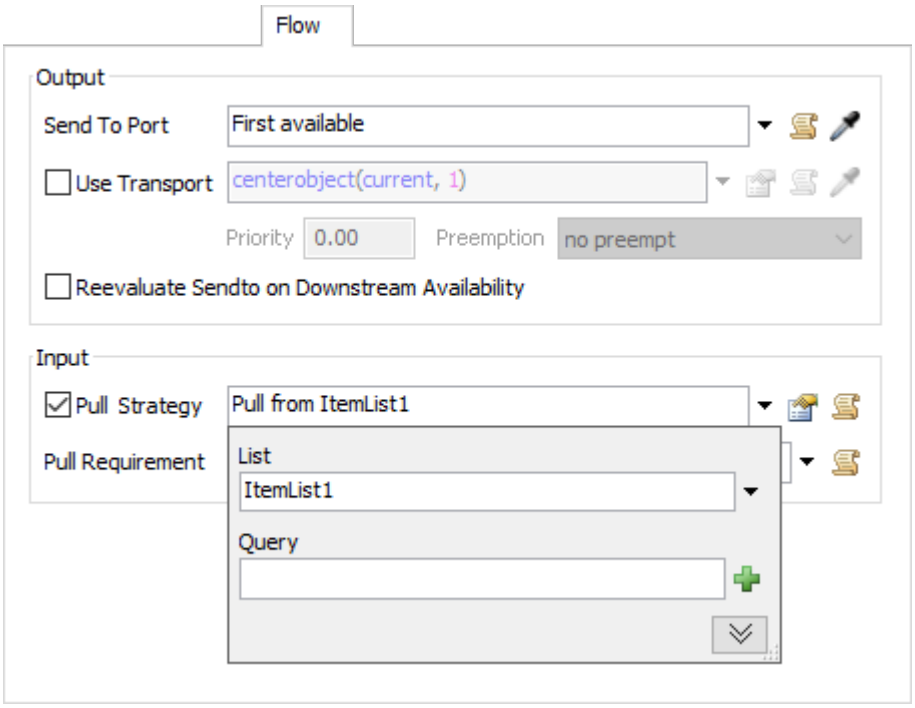
1. Build the Model Layout - Starting with a new model, create and connect objects in the model according to the following image, namely, a source that connects to three queues, and four processors that connect to a single sink. Do not connect the queues to the processors.



2. Define Source Properties
    1. Open the Source properties window.
    2. Define the Inter-Arrivaltime as exponential(0, 2.5, 0).
    3. In the Flow tab under Send To Port choose Random Port from the pick list.
    4. Also, let's make three different item types, which we will use later. Add an OnCreation trigger to Set Item Type and Color to duniform(1, 3).
3. Add an Item List - From the Toolbox, press the button and select Global List > Item List. This will add an item list to the model. We don't need to define any custom properties right now, so close the list's properties window.
4. Make Queues Push to the List - Define each queue's Send To Port by choosing Use List > Push to Item List from its pick list. Make sure the list corresponds the the item list's name that you added.



5. Make Processors Pull from the List - For each processor's Input, check Pull and under Pull Strategy choose Use List > Pull from Item List from the pick list. Make sure the list name corresponds with the name of the item list that you added. You can leave the query blank.



6. Run the Model - Now we're ready to run the model. Before we do that, though, let's open live views of what is on the list and who is waiting to pull from the list. In the Toolbox, double-click on the item list to open its properties. From the General tab, press both View Entries... and View Back Orders.... These buttons will open the entry viewer and back order viewer respectively.

Now run the model.

When the model first starts, you'll notice that there are four rows in the back order viewer.

ItemList1 Back Orders

puller	query	requested	required	fulfilled
/Processor4		1	1	0
/Processor5		1	1	0
/Processor6		1	1	0
/Processor7		1	1	0

This means that the four processors have initiated a pull from the list, but since no items have been pushed to the list, they are waiting for their pull to be fulfilled.

As the model runs, items will enter the queues and be pushed onto the list, fulfilling the processor back orders, and then start to be processed. When an item is pushed onto the list and there are outstanding back orders, the item will be immediately pulled from the list. The back order will be fulfilled and subsequently removed from the back order list.

Eventually all outstanding back orders will be fulfilled, meaning all processors have found an item to process and are now processing it. When the back order list becomes empty, subsequent items that enter the queues are pushed onto the list, and, having no back orders to fulfill, they will remain on the list until a processor is finished processing its current item and pulls from the list again.

ItemList1 Entries

value	itemType	age	distance	queueSize
/Queue9/Box	1	6.20	Puller Required	4
/Queue8/Box	2	6.04	Puller Required	2
/Queue3/Box	1	5.07	Puller Required	1
/Queue9/2	2	3.49	Puller Required	4
/Queue9/3	3	2.56	Puller Required	4
/Queue9/4	2	1.97	Puller Required	4
/Queue8/2	3	1.86	Puller Required	2

When items are pushed onto the list, they are added to the end of the list. If pull operations do not include prioritization as part of their query, the items will be pulled off in FIFO (first-in-first-out) order. Back orders will also be fulfilled in FIFO order if you do not define a Back Order Queue Strategy.

Model Experimentation

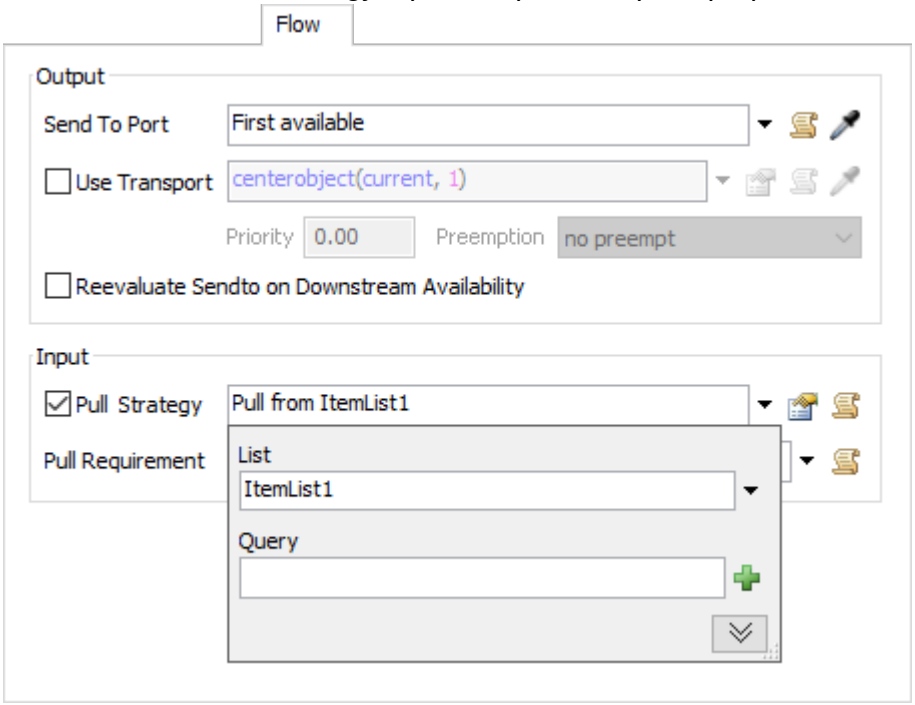
Now let's experiment with the different settings for the pull operations.

Scenario 1: Filter by Item Type

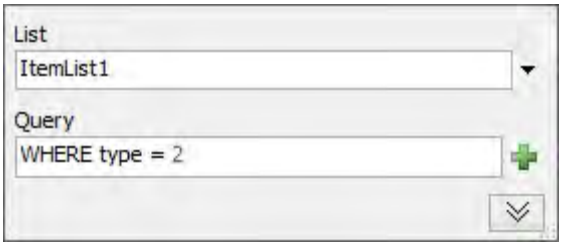
Let's say that one of the processors can only process items of type 2. We can accomplish this logic in our simulation by implementing a simple query on the processor.

1. Open a processor's properties window.

2. In the Flow tab, under Pull Strategy, open the pick list option properties.



3. Under Query enter `WHERE type = 2`. You can use the ✚ button on the right for help in building this query, or just enter it in directly.



This expression means that you only want to pull items from the list where the type column is equal to 2.

4. Reset and run the model.

Now you should notice that the processor whose pull query we changed will only pull and process items with a type of 2 (the green items).

For FlexScript Coders: The = operator

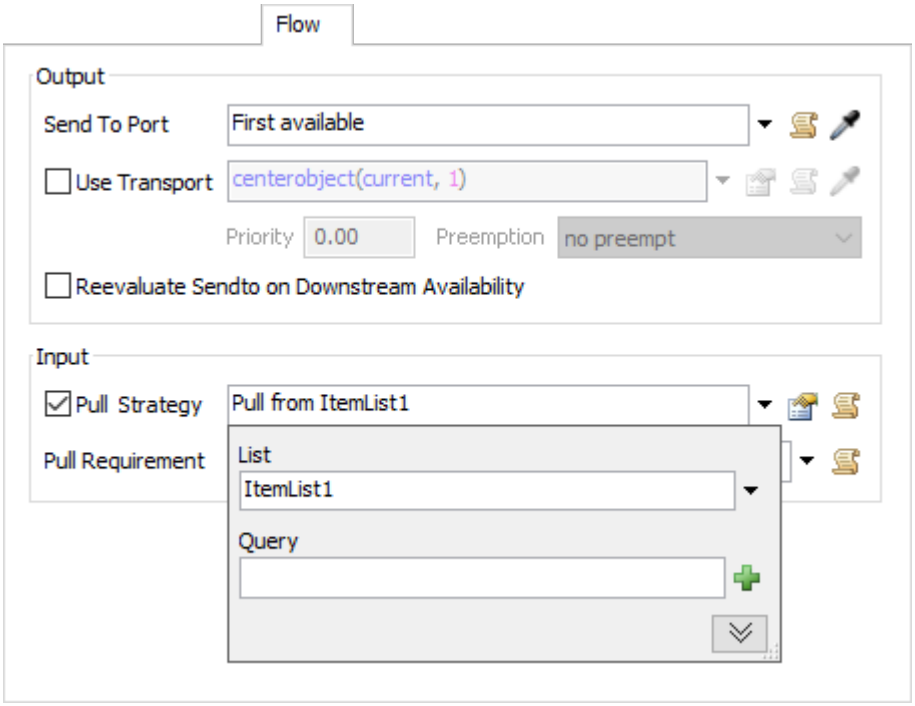
In this example we use the expression `WHERE type = 2`. The use of = as a comparison operator may be counter-intuitive for people who are familiar with FlexScript but new to SQL. In FlexScript (and in many other c-style languages), = is an assignment operator, whereas == is a comparison operator. In standard SQL, = is the comparison operator, and there is no assignment operator. To alleviate this confusion for our users, we have added a special SQL rule within FlexSim that makes both = and == a comparison operator, so that if you're used to using ==, you can do that in your list queries in FlexSim.

Scenario 2: LIFO Pulling

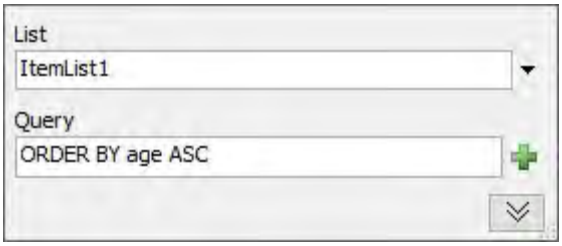
Let's say now that we want all the processors to pull in LIFO order (last-in-first-out) instead of the default FIFO, meaning that the items that were most recently added to the list should be the first to be pulled from the list. Again, we can accomplish this in the simulation by implementing a simple pull query.

For each of the processors, do the following:

1. Open the processor's properties window.
2. In the Flow tab, under Pull Strategy, open the pick list option properties.



3. Under Query enter `ORDER BY age ASC`. Again, you can use the ✚ button on the right for help in building this query, or just enter it in directly.



This expression tells the pull operation that you want to prioritize by the list's age column in ascending order, meaning items on the list with the lowest age will get the highest priority.

If you want to keep the previous scenario's logic where one processor only pulls type 2, then for that processor, you just append the ORDER BY clause at the end of the expression.

`WHERE type = 2 ORDER BY age ASC`

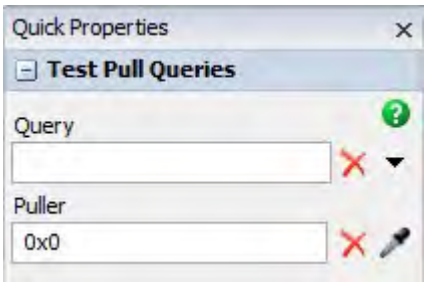
4. Reset and run the model.

You should notice now that items on the end of the list will be pulled first, and items at the front will be pulled last.

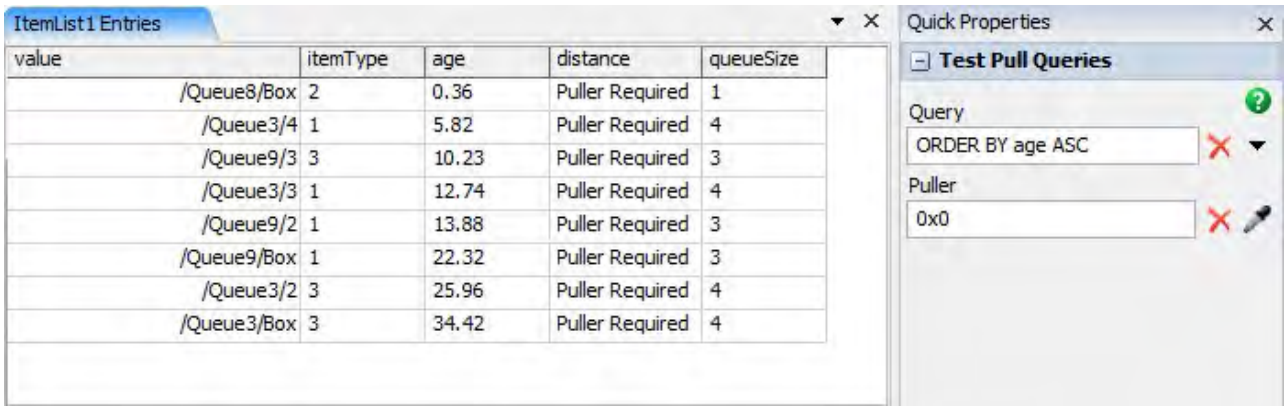
Using the Entry Viewer's Test Pull Queries

The Quick Properties for the entry viewer allow you to test out various queries. For example, what if we wanted to test out our `ORDER BY age ASC` query to make sure it does what we want before actually implementing it in the simulation logic? We can enter this in the Query field in Quick Properties.

1. Run the model until there are several items on the list.
2. With Quick Properties open, click in the entry viewer, to show the Test Pull Queries panel.



3. Test the pull query by entering `ORDER BY age ASC` in the Query field and pressing Enter.



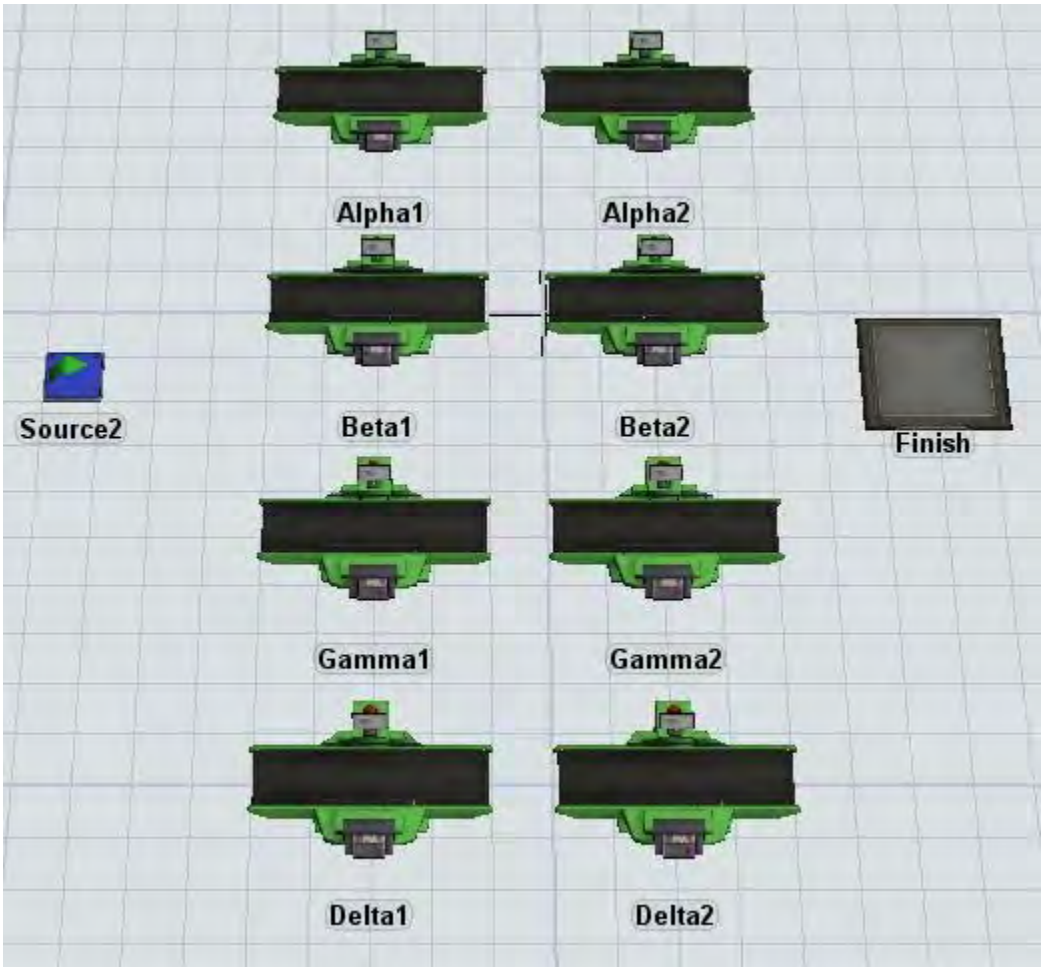
Notice that the items in the list are now reordered based on their age in ascending order. A pull operation that uses this query would pull from the top of this sorted list, namely */Queue8/Box* with type of *2* and age of *0.36*.

SQL Quick Start

If you're new to using SQL, the SQL Quick Start topic includes a lot more examples of the different pull queries that you might use in this model.

Job Shop Example

Here we will build a model that uses lists for routing items in a simple job shop model. The following image shows the basic layout, namely a source, 8 processors, and a finish queue.



As the image shows, there are basically four types of operations:

- Alpha - 2 Stations
- Beta - 2 Stations
- Gamma - 2 Stations
- Delta - 2 Stations

In this model we will have 3 different types of products, designated by item type. Each type has a unique sequence of operations that must be performed on the part before it is finished. The following table shows the manufacturing steps required for each type.

Steps

	ItemType 1	ItemType 2	ItemType 3
Step 1	Alpha	Gamma	Beta
Step 2	Beta	Delta	Alpha
Step 3	Gamma	Beta	Beta
Step 4	Delta	Gamma	Alpha
Step 5	Finish	Delta	Delta
Step 6		Beta	Gamma
Step 7		Finish	Delta
Step 8			Finish

Model Building Steps

1. Add an Item L ist - In the Toolbox , press the  button and choose Global List > Item List from the drop-down menu. This will add an Item List to the model. For now you can close the List properties window.

2. Add the Steps Table - In the Toolbox, press the  button and choose Global Table from the drop-down menu. In the table view, define the name of the table as "Steps" and give it the following data:

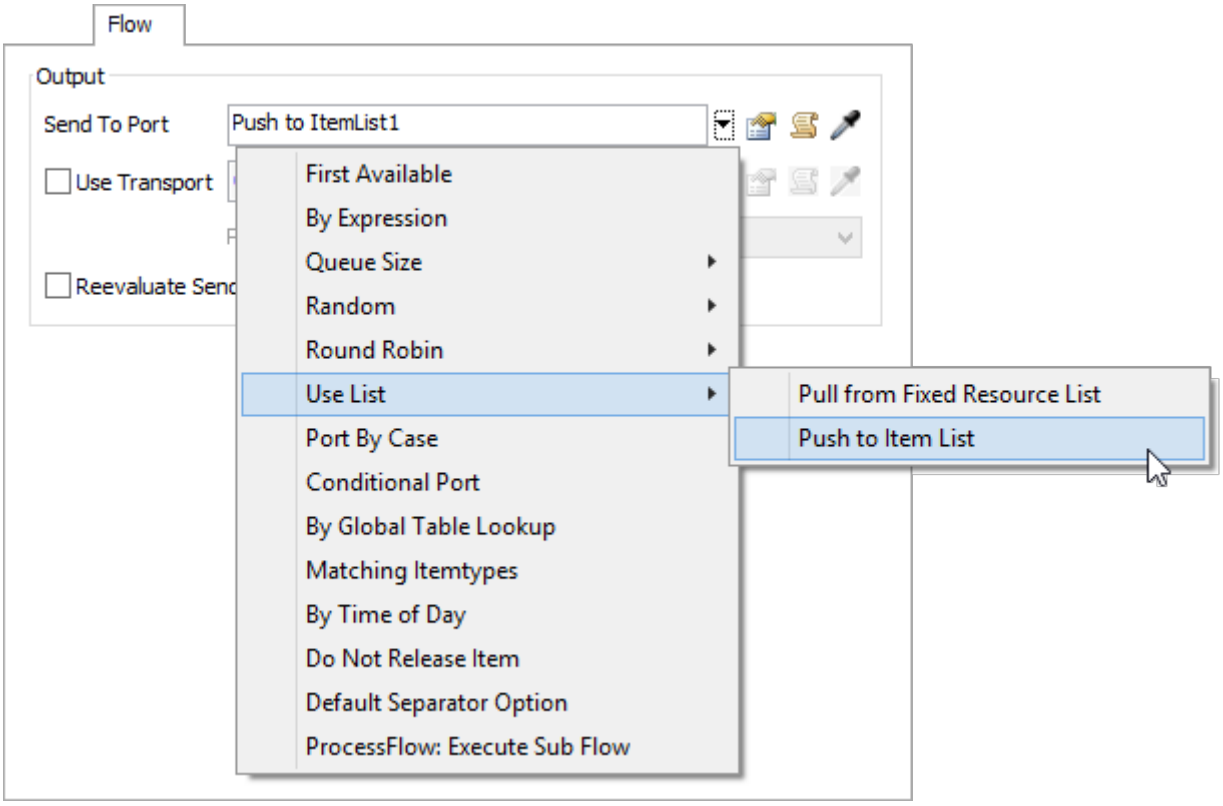
Steps

	ItemType 1	ItemType 2	ItemType 3
Step 1	Alpha	Gamma	Beta
Step 2	Beta	Delta	Alpha
Step 3	Gamma	Beta	Beta
Step 4	Delta	Gamma	Alpha
Step 5	Finish	Delta	Delta
Step 6		Beta	Gamma
Step 7		Finish	Delta
Step 8			Finish

3. Create a Source - Create a source object in your model.

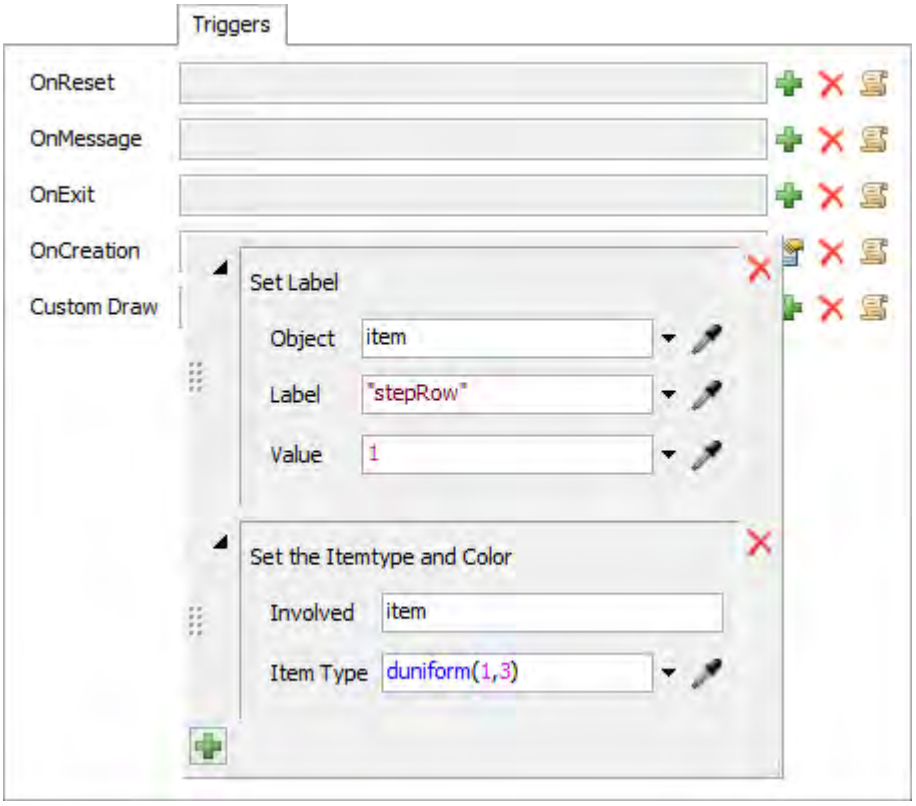
4. Define Source Properties - Double-click on the source.

    1. Give the source an Inter-Arrivaltime of exponential(0, 30, 0).
    2. Select the Flow tab. Under Send To Port, press the drop-down button and choose Use List > Push to Item List.



Make sure that the list name matches the name of the list you added.

3. Select the Triggers tab in the Source properties. In the OnCreation, add triggers to set a label named stepRow to 1, and set item type and color to duniform(1, 3).



The stepRow label will designate the row in the Steps table that the item is currently on. After each operation, the stepRow label will be incremented.

5. Define List Fields - Go back to the Item List's properties (double-click the Item List in the ToolBox). For this model we will use the following fields:

   - stepRow - This field will reference the current stepRow label on the item. We'll use this to prioritize items who are further into their process (ORDER BY stepRow DESC). o step - This field will reference the name of the current step that the item is on, e.g. Alpha, Beta, Gamma, or Delta. We will use it to match against the station that will pull from the list.
   - type - This field will not be used in any queries. Rather it will be used to better visualize the items that are on the list.

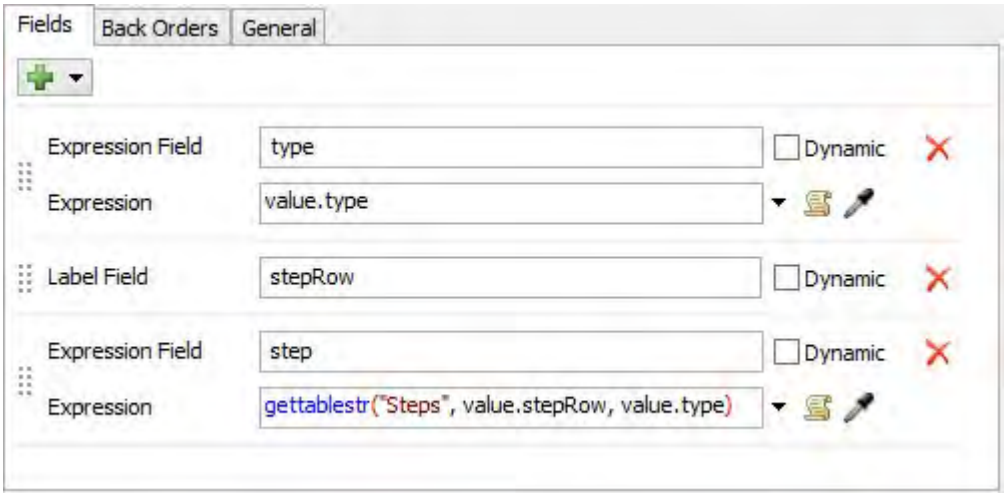
4. In the Item List's Fields tab, remove all fields but type

5. Add a Label field named stepRow.

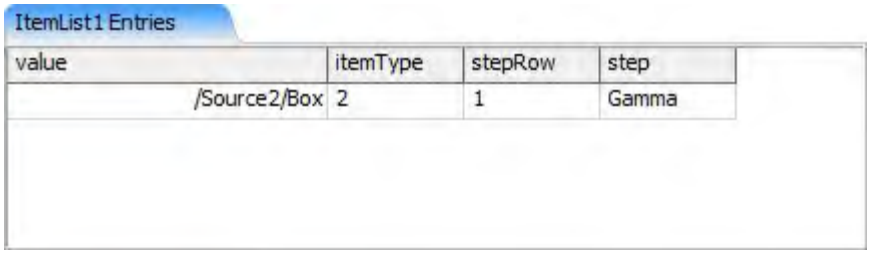
6. Add an Expression field and give it the expression gettablestr("Steps", value.stepRow, value.type).

   In expression fields you use "value" to access the primary value on the list. This expression gets a value from the Steps table (gettablestr("Steps", ...) where the row is the item's current stepRow (value.stepRow), and the column is the item's type (value.type)

7. Press Apply in the List properties window.



6. Do a Quick Test Model Run - In the General tab of the List properties, click View Entries... to open a window showing current entries on the list. Reset and run the model just to test that the source is properly assigning the data and pushing the item onto the list. After the source's first inter-arrival time has expired, the list should get an item on it.

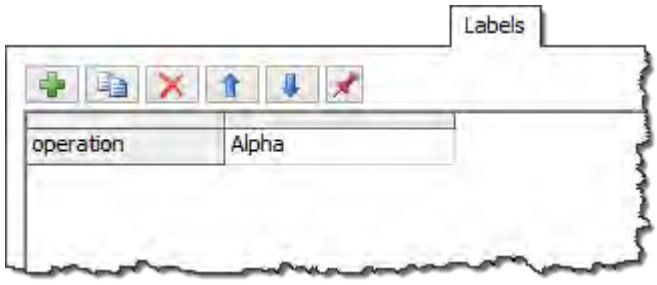


7. Create a Processor Object - Drag a processor object from the library into the model.

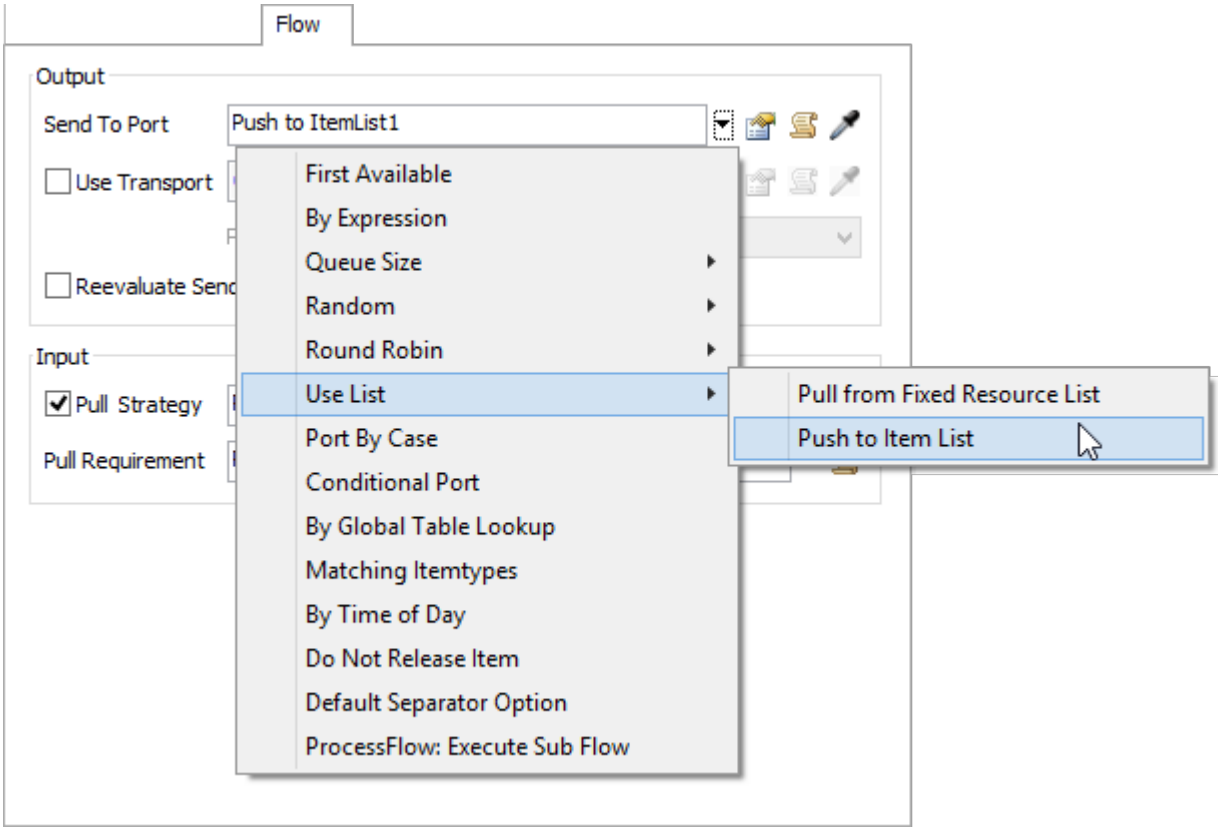
8. Define Processor Properties - Double-click on the processor.

    0. Name the process Alpha1

    1. Give the processor a string label named operation. Set the value to Alpha.

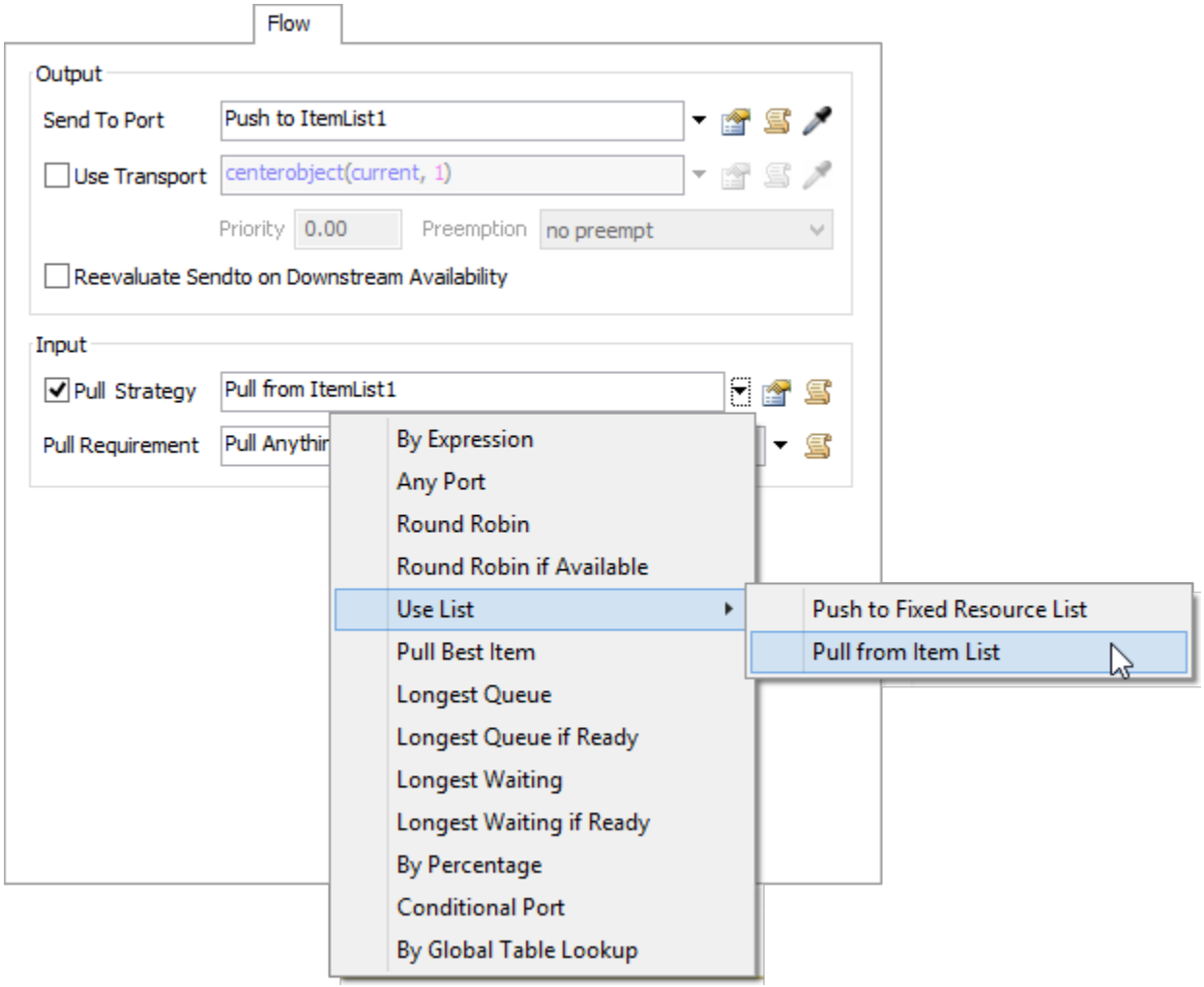


    2. Select the Flow tab. Under Send To Port, press the drop-down button and choose Use List > Push to Item List.

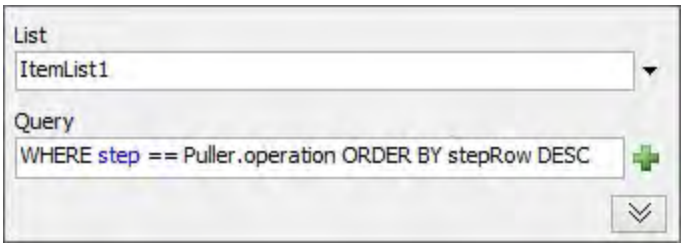


Make sure that the list name matches the name of the list you added.

3. In the Input pane, check Pull, then under Pull Strategy, press the drop-down button and choose Use List > Pull from Item List.

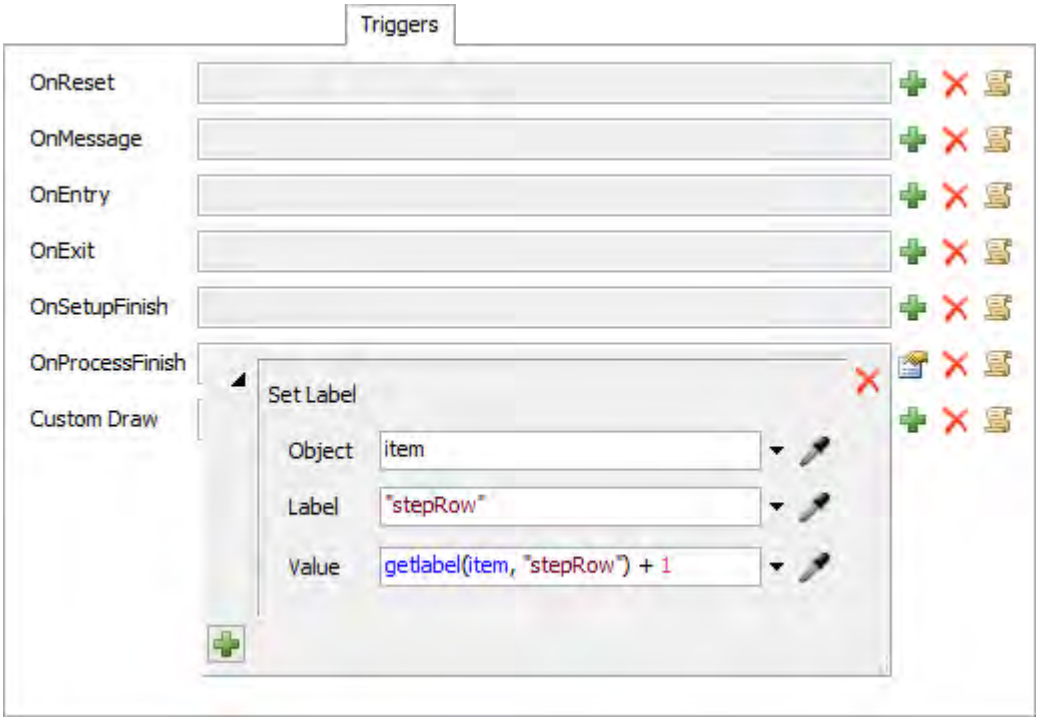


Make sure that the list name matches the name of the list you added, and define the Query as WHERE step == Puller.operation ORDER BY stepRow DESC.



This will make the processor only pull items from the list whose step field matches the processor's operation label. It will also prioritize items with the highest stepRow label, i.e. items that are further along in their steps.

4. Select the Triggers tab. In OnProcessFinish, add a trigger to set the stepRow label. In the Value field for the trigger, enter the code `getlabel(item, "stepRow") + 1`.



9. Define Remaining Processors - Now that you've defined the processor object, you can copy/paste the other processors from that object. Define a second Alpha processor (copy/paste it). Define two Betas, two Gammas, and two Deltas by copy/pasting the original processor, and change the pasted objects' operation label to its corresponding operation (Beta, Gamma or Delta).

10. Add a Finish Queue - Drag a queue from the library. The queue will have much of the same properties as the processors. It will be the queue to dump the items when they reach their Finish step.

    0. Name the queue Finish

    1. Give it an operation label with the value Finish

    2. In its Flow tab, check Pull and under the Pull Strategy choose Use List > Pull from Item List.

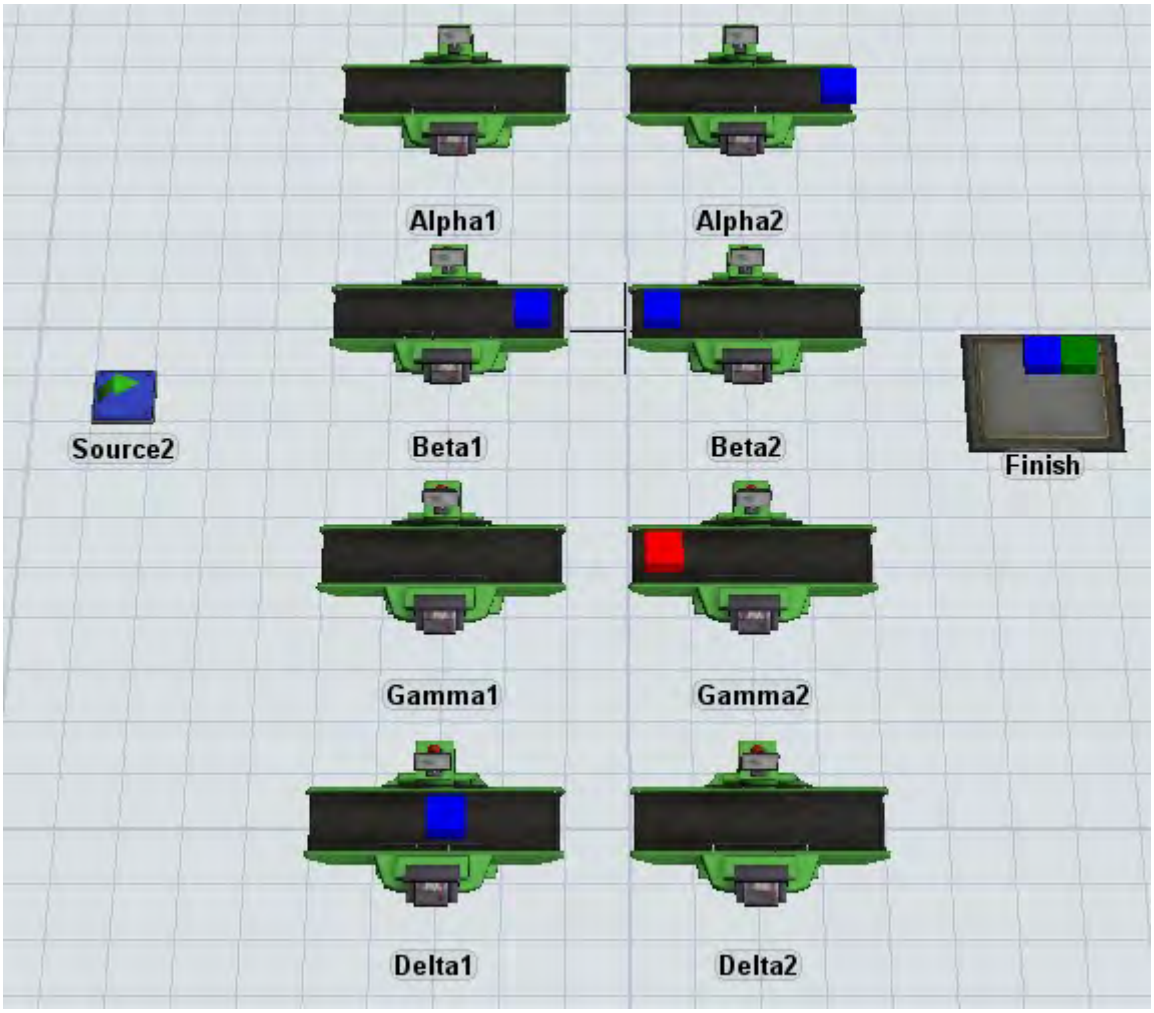
    3. Define the pull query as `WHERE step == Puller.operation`. We don't need to prioritize here because it's essentially a sink.

11. Run the Model - Now you should be able to reset and run the model. You may also want to go back to the Item List and view its back orders. Right-click on the list in the Toolbox, and select View Back Orders.

ItemList1 Back Orders

puller	query	requested	required	fulfilled
/Alpha1	WHERE step == Puller.operation ORDER BY stepRow DESC	1	1	0
/Alpha2	WHERE step == Puller.operation ORDER BY stepRow DESC	1	1	0
/Beta1	WHERE step == Puller.operation ORDER BY stepRow DESC	1	1	0
/Beta2	WHERE step == Puller.operation ORDER BY stepRow DESC	1	1	0
/Gamma1	WHERE step == Puller.operation ORDER BY stepRow DESC	1	1	0
/Gamma2	WHERE step == Puller.operation ORDER BY stepRow DESC	1	1	0
/Delta1	WHERE step == Puller.operation ORDER BY stepRow DESC	1	1	0
/Delta2	WHERE step == Puller.operation ORDER BY stepRow DESC	1	1	0
/Finish	WHERE step == Puller.operation	1	1	0

As the model runs you should see items moving through the various processors in the order defined in their associated Steps table column (red = type 1, green = type 2, blue = type 3).



SQL Quick Start

List pull logic uses SQL queries to determine what should be pulled and how to prioritized it. This topic is a quick introduction to SQL, including several example queries.

This topic uses the connectionless routing example as the base model for query examples. If you would like to test these examples, build the model and then come back to this topic.

For information about the special values of value and Puller, see Functional Reference.

SQL Clauses

In SQL, you use what are called clauses to define queries. Each clause represents a rule by which the query should perform its search. There are many different clauses available in standard SQL, but when using lists, you usually only need to know 2, namely WHERE and ORDER BY.

WHERE Clause

The WHERE clause essentially defines *what* you want to pull from a list. When you pull from a list using a query with a WHERE clause, you are essentially saying, "I want to pull something from the list *where* the entry meets this criterion." For example, one scenario in the connectionless routing example uses the following query.

```
WHERE type = 2
```

This WHERE clause defines a rule that you only want to pull items whose type is equal to 2. More specifically, the type field value of the list entry must be equal to 2.

ORDER BY Clause

The ORDER BY clause essentially defines how to prioritize what should be pulled when there are multiple potential candidates for pulling. When you pull from a list using a query with an ORDER BY clause, you are essentially saying, "I want to pull something from the list, and if there are multiple possibilities, *order* them *by* this rule." For example, one scenario in the connectionless routing example uses the following query.

```
ORDER BY age ASC
```

This ORDER BY clause says that you want to prioritize the items on the list with the lowest age. More specifically, the pull query should sort candidates by the age field value in ascending order, and pull the first item in the resulting sorted list. Here we use the optional ASC keyword to tell it to sort in ascending order (default). We could alternatively use the DESC keyword to sort in descending order.

Handling Ties

The ORDER BY clause can easily handle ties. Just comma-separate multiple expressions to define subsequent prioritization rules if there are ties.

```
ORDER BY type DESC, age ASC
```

This example says that you should first order by type in descending order, and then if there are ties (multiple items with the same type), then next order by age in ascending order.

Using Both WHERE and ORDER BY Clauses

If you define a query that uses both the WHERE and ORDER BY clauses, the WHERE should come first, and the ORDER BY can be placed immediately after it, as in the following example.

```
WHERE type = 2 ORDER BY age ASC
```

This query says that it will only pull items with type 2, and if there are multiple that meet that criterion, it should take the one with the smallest age.

Expression Operators

The following table shows operators that you can use in writing pull queries.

Operator	Definition	Example
Comparison Operators		
= or ==	Equals comparison	WHERE type = 2
<	Less-than comparison	WHERE type < 2
>	Greater-than comparison	WHERE type > 2
<=	Less-than-or-equal comparison	WHERE type <= 2

>=	Greater-than-or-equal comparison	WHERE type >= 2
<> or !=	Not-equal comparison	WHERE type <> 2
BETWEEN	Between a defined range (inclusive)	WHERE type BETWEEN 2 AND 5
IN	Value in a defined set	WHERE type IN (1, 3, 5)
Logical Operators		
AND or &&	Logical and	WHERE type = 2 AND queueSize < 5
OR or ||	Logical or	WHERE type = 2 OR queueSize < 5
Math Operators		

+	Addition	ORDER BY distance + age
-	Subtraction	ORDER BY distance - age
*	Multiplication	ORDER BY distance * age
/	Division	ORDER BY distance / age
Expression Grouping		
()	Grouping Parentheses	WHERE (type < 2 OR type > 8) AND queueSize < 5

Examples

The following examples are based on the connectionless routing example. Again, if you would like to test these examples out, build the model then come back to this topic.

Multi-Rule Prioritization

If you combined scenario 1 with scenario 2, you will likely have noticed that the queues would often accumulate with type 1 and 3 more than type 2. This is because one of the processors was dedicated to type 2, while the others would take any of the items. Hence, type 2 happened to be pulled more often because both the dedicated processor and the other processors were pulling it, to the detriment of the other type.

Let's say that we need to keep the same requirements, namely one dedicated processor with the rest undedicated, but we want to try to better alleviate the "unfair" queueing of type 1 and 3. One option is for the

undedicated processors to prioritize type 1 and 3. Only if there are neither of these type available would the undedicated processors pull type 2. Nevertheless, we still want to pull in LIFO order when there are multiple possibilities within the first division.

To implement this logic in the model, define the pull query on the undedicated processors as follows:

```
ORDER BY type <> 2 DESC, age ASC
```

This example defines `type <> 2 DESC` as the highest order priority, and `age ASC` as the second order priority. Here we use the rule that a logical expression that is true will give the value 1, whereas a logical expression that is false will give the value 0. So, the expression `type <> 2` gives a 1 when type is not equal to 2, and will thus take priority over 0 (type is equal to 2) in a descending order sort. The second order priority takes effect if there are ties in the first, namely if it found multiple items of type 1 or 3, or if it couldn't find any 1's or 3's, but it found multiple 2's. In those cases it will sort that valid set by age in ascending order, effecting a LIFO behavior.

Dedicated Processor, Multiple Types

You may have a processor that is dedicated to more that one type. For example, one processor only processes type 1 and 3. This can be acheived with either of the following pull queries.

```
WHERE type = 1 OR type = 3
```

```
WHERE type IN (1, 3)
```

FlexScript Functions

You can also use FlexScript function calls in your queries. This is often used to do math operations that are not in the standard set. For example, you may want to sort primarily by distance (find the item closest to where the pulling processor currently is), but only up to some threshold, and then sort by some other rule. Specifically, maybe you want to separate distance into "buckets" of 10 meters each. Items within the same "bucket" would be pulled in LIFO order (age ASC). Here you would use the round() function to round the distance to every 10 meters, and then use age as the second order priority.

```
ORDER BY round(distance / 10) ASC, age ASC
```

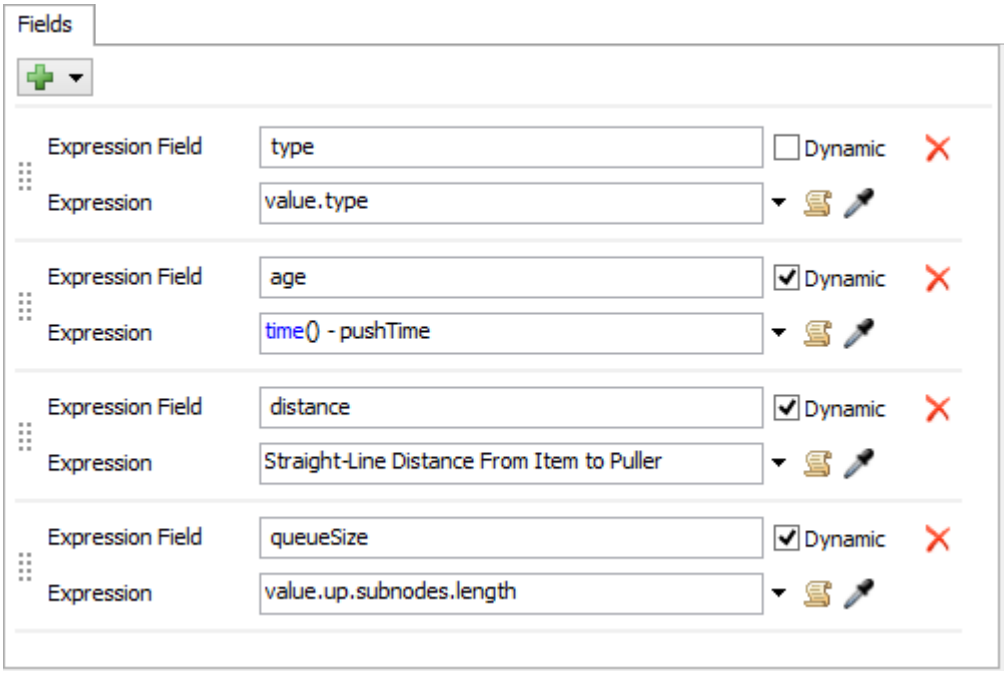
SELECT Clause

In some special circumstances you might also use the SELECT clause. See the Functional Reference for more information.

Fields Tab

The Fields tab is where you define the set of fields that can be used in a pull query or back order queue strategy. To get to the Fields tab of a List, double-click on the desired List in the Toolbox (if you haven't added a List you can add if from the  button in the Toolbox). Then select the Fields tab.

Depending on the type of list you create, a default set of fields will be added. You can easily remove these fields by pressing the  next to the field, and/or add your own if needed. The following image shows the default set of fields available for an item list.



Adding Fields

Click on the  button for a list of available fields to add. The available add options will include, at a minimum, the following set of options.

- Expression - An expression field will evaluate a FlexScript function to retrieve the field's value. Enter the name you want to call the field in Expression Field. Then enter the expression directly in Expression, or use the buttons to the right of the edit to choose various options, sample for the desired value, or edit the code in a code window.
- Label - A label field will retrieve its value from the same-named label on the entry's value object. Enter the name of the label in Label Field. Note that this will only work if a list entry's value is an object reference, not a number or string.
- Push Argument - A push argument field is only used if you are manually pushing values onto the list using the listpush() command. The listpush() command allows you define additional arguments after the partition ID argument. In the Fields tab you add a Push Argument field, define its name and the number of the additional argument associated with the field. For example, if your listpush() call were: listpush("ItemList1", item, partitionID, up(item));, you could then add a push argument field, in Push Argument Field give it the name parentObject, and under Argument, enter 1, meaning the first additional argument after the partition ID of the listpush() command.
- age - The age field is a dynamic customized expression field that returns the amount of simulation time that has elapsed since the entry was added to the list.
- pushTime - The pushTime field is a customized expression field that returns the simulation time when the entry was added to the list.

Fields for Specific List Types

In addition to the standard set of fields, pressing the  button will provide options specific to the type of list you are using. Note that all of these are just customized Expression fields. This means that if the given set of automatic fields don't fit your needs, you can easily view and customize the code associated with each field. Available add options may include the following:

Item List Fields

- type - The item type of the item.
- distance - This is the straight-line distance from the item to the puller object. Using in the case of item lists, the puller will be a FixedResource that is pulling from the list, trying to determine which item to receive next.
- queueSize - The size of the object that containers the item. For example, if the item is in a queue, this is the total number of items currently in the queue.
- ageInQueue - The time that has elapsed since the item first entered the object that currently contains it.
- totalAge - The time that has elapsed since the item was first created.
- meetsPullRequirement - Used in FixedResource pulling. This evaluates to 1 if the item satisfies the puller object's pull requirement.

Fixed Resource List Fields

- queueSize - The number of items currently in the Fixed Resource.
- distance - The straight-line distance from the Fixed Resource value to a puller object (usually an item).
- throughput - The throughput of the Fixed Resource.
- isIdle - 1 if the Fixed Resource's state is STATE_IDLE, 0 otherwise. Note that you may need to adjust this field based on the state profile of the objects that will be on the list. For example, Queues don't use STATE_IDLE as part of their state profile, so if Queues will be on the list, the expression may need to be adjusted.

Task Sequence List Fields

- priority - The priority value of the Task Sequence.
- preempt - The preempt value of the Task Sequence.
- distance - The travel distance from the puller Task Executer to the Task Sequence's first travel destination.
- isTransportTS - 1 if the Task Sequence is a standard transportation Task Sequence to move an item from one Fixed Resource to another, 0 otherwise.
- isUtilizeTS - 1 if the Task Sequence is a utilization Task Sequence, i.e. a Task Sequence to go and be utilized at a Processor object, 0 otherwise.
- loadToUnloadDistance - For transport Task Sequences, the straight-line distance from the point where item will be picked up to the point where the item will be dropped off.

- loadStation - For transport Task Sequences, the object that the item will be loaded from.
- unloadStation - For transport Task Sequences, the object that the item will be unloaded to.
- transportItem - For transport Task Sequences, the item that will transported.

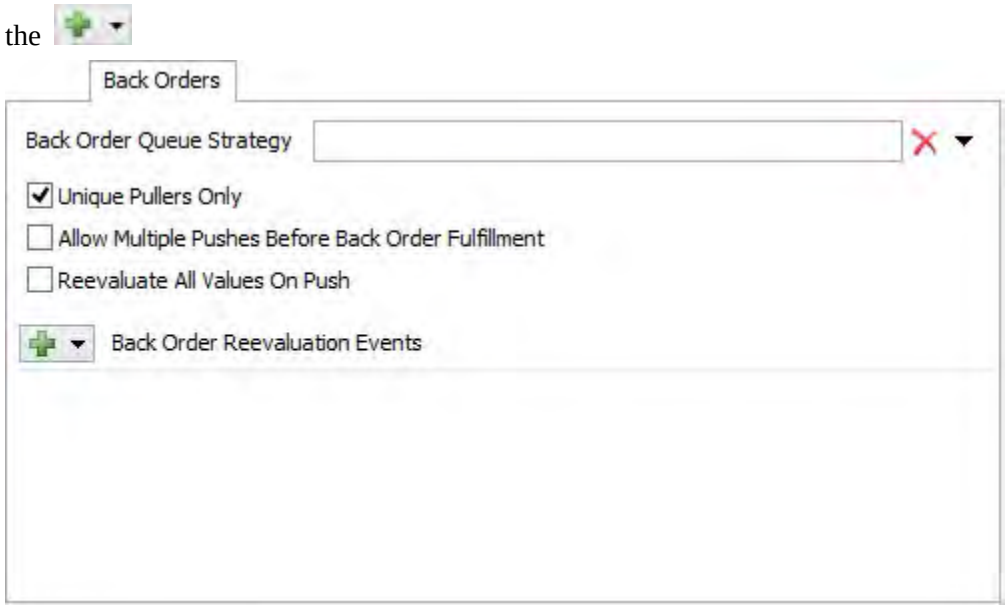
Task Executer List Fields

- utilization - The non-idle state percentage of the Task Executer.
- totalTravel - The total travel distance of the Task Executer.
- distance - Assuming the puller is a Task Sequence, the travel distance from the Task Executer to the Task Sequence's first travel destination.
- queueSize - The number of items currently in the Task Executer.
- throughput - The Task Executer's current output.
- isIdle - 1 if the Task Executer's current state is idle, 0 otherwise.
- isPullerPreempting - Assuming the puller is a Task Sequence, 1 if the Task Sequence is preempting, 0 otherwise.
- pullerPriority - Assuming the puller is a Task Sequence, the priority value of the Task Sequence.

Dynamic Fields

Each field you add (except push arguments) can be a dynamic field by checking the Dynamic checkbox. A dynamic field is a field whose source value may change while an entry is on the List. While a non-dynamic field's value will be evaluated only once when the entry is added to the List, a dynamic field's value will be re-evaluated every time the list is queried with a pull request. For this reason, non-dynamic fields are faster in execution, but dynamic fields offer more power in querying lists whose source values may change. Also, all puller-dependent fields must be designated as dynamic.

The Back Orders tab is where you define List properties associated with back orders. To get to the Fields tab of a List, double-click on the desired List in the Toolbox (if you haven't added a List you can add if from button in the Toolbox). Then select the Back Orders tab.



- Back Order Queue Strategy The Back Order Queue Strategy defines how back orders will be prioritized for a given value. When a value is pushed onto the List, outstanding back orders are processed for that value in the order defined by this queue strategy. It should be an SQL ORDER BY statement that references at least one puller-dependent field (see the puller term in the List Concepts topic for more information). This is because the back order queue strategy is used to prioritize back orders against a single entry. An ORDER BY statement that gets data only associated with the entry value would consequently have the same result value for all back orders, and would thus be unable to properly prioritize the back orders.

  Examples

  ORDER BY Puller.priority DESC - This will prioritize the pullers who have the "priority" label with the highest value.

  ORDER BY pullerPriority DESC - This example is essentially the same as the previous example, except that here you would need to explicitly add a dynamic puller-dependent expression field named pullerPriority, and give it the expression: puller.priority.

  ORDER BY distance ASC - This example will prioritize the back orders with the lowest distance value, which usually defines a distance between the value and the puller.

- Unique Pullers Only Check this box to disallow multiple back orders with the same puller. If this is checked and a pull request is made with the same puller as an existing back order, then that pull request will replace the previous pull request.

  Overwriting Back Orders

  If a back order is replaced by a new pull request using the same puller, the query, request number and require number will also be replaced by the new pull request. Objects that were previously listening for the back order to be fulfilled will continue listening.

- Allow Multiple Pushes Before Back Order Fulfillment If this box is checked, the list will wait effectively zero seconds after a push before processing back orders for the pushed value(s). This means that if multiple values are pushed onto the list at the same simulation time, all those values will get on the list before back orders are fulfilled.

  This setting is useful in several cases. First, if you are pulling from the list with an ORDER BY clause in your pull query, then you will likely want to allow all values onto the list before evaluating the query, so it can consider all values pushed, instead of just pulling as soon as the first one gets on. Second, if you are pulling from the list with a required number less than the requested number, then allowing multiple pushed values onto the list before fulfilling the back orders can get you closer to your requested number.

  To allow multiple pushes, the list creates an event in effectively zero time after a value is pushed to the list to process back orders. This is "effectively" zero time because the event is not created at exactly the same time as the time the value is pushed. Rather, the event's time will be incremented by the smallest amount representable by a double precision floating point value. This means that all events created at the exact same time as the original push will be executed before the back order fulfillment event, even if those events are created after the original value is pushed. And while the back order fulfillment time is not exactly the same as the push time, it is effectively the same.

  When this box is checked, the listpush() command will always return null. In other words, since back orders are no longer fulfilled synchronously with pushes, they will never be fulfilled in time for the listpush() command to return a valid value telling who pulled the pushed value.

- Reevaluate All Values On Pushes By default, back orders on the list will only evaluate new values that are pushed to the list. Check this box to cause all values (those already on the list and those being pushed to the list) to be evaluated for every back order whenever a new value is pushed to the list. This can be important if there are dynamic fields that may change and cause a value already on the list to be pulled when another value is pushed onto the list.

- Back Order Reevaluation Events This pane lets you define when back orders should be reevaluated. This is usually only needed if in your pull request you have a WHERE clause whose result may change while the back order is active. For example, consider the following query: `WHERE queueSize <= 5`. If the pull request is made at a point when queueSize is greater than 5, then you may want to reevaluate that pull request when queueSize becomes less than 5. FlexSim doesn't automatically know when this condition may change, so you need to define events that will trigger reevaluation.

  To add a reevaluation event, click on the  button, and choose an event to add. Then choose the Event Type you want to listen for.

Events

There are several categories of reevaluation events, as follows:

Puller Event

A puller event is an event that happens on the puller. Here the puller must be an object (not a number or string). When a back order is created, the List will listen for the defined event on the puller object, and when the event fires, the back order will be reprocessed for all values in the list.

Value Event

A value event is an event that happens on the value, i.e. the primary value of an entry. Here the value must be an object (not a number or string). When the value is pushed to the List, the List will listen for the defined event on the value object, and when the event fires, all back orders will be reprocessed for that entry.

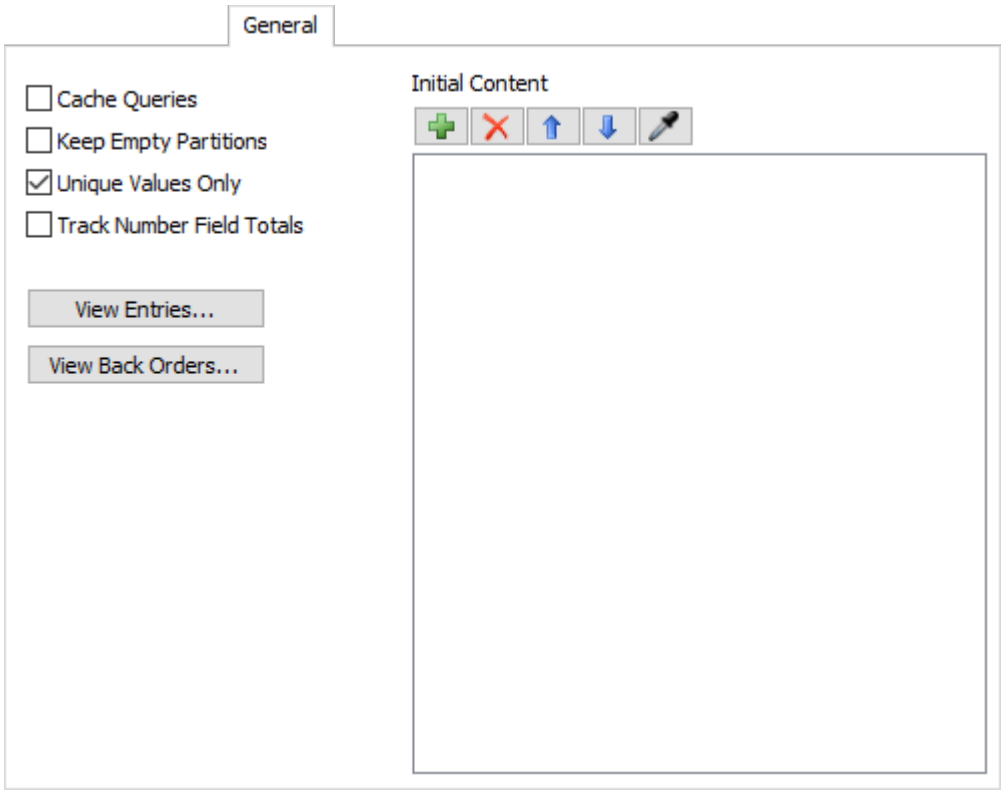
Global Event

A global event is some event that is not associated with either values or back orders. The List will listen for this event and will reprocess all back orders whenever the event happens.

Time Interval

A timer interval event will cause the List to repeatedly reevaluate back orders at a set time interval. When the first back order is added, the List starts the timer loop and reevaluates back orders after the given time elapses. The timer loop ends when the back order queue is emptied.

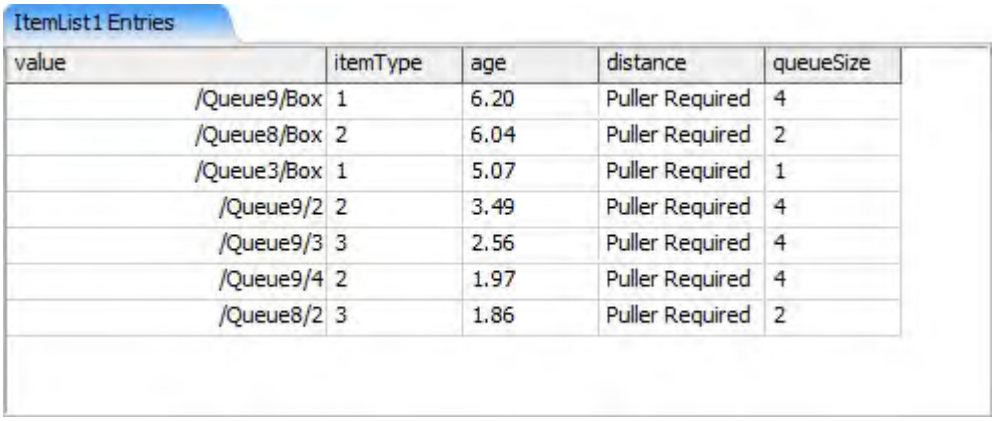
The General tab is where you define general properties for a List. To get to the General tab, double-click on the desired List in the Toolbox (if you haven't added a List you can add if from the  button in the Toolbox). Then select the General tab.



- Cache Queries If checked, the List will cache parsed queries. This means that if you use the same query multiple times, the query will only be parsed once instead of each time the pull request is made. This can increase simulation run speed, but does require some small memory overhead, depending on how many unique queries are made in a simulation.
- Keep Empty Partitions If checked, the list will never remove partitions, even if they are empty. This is useful if you want to keep statistics on a specific partition. However, depending on how many unique partitions are created on the list, this can use significant amounts of memory.
- Unique Values Only If checked, the List will disallow multiple entries with the same value. If a value is pushed when there is already an entry with that
- Track Number Field Totals If checked, the list will keep statistics on the total (sum) field values of all entries for non-dynamic number fields. This can be used if you want to either listen for changes to a field's total, usually using Process Flow, or if you want to use it to gather model statistical results.
- View Entries... Press View Entries to open list's entry viewer.

- View Back Orders... Press View Back Orders to open list's back order viewer.
- Intial Content The set of controls under Initial Content allow you to define a set of values to be pushed onto the List when the simulation starts. Use the buttons above to add, remove and reorder these values. You can add objects individually, or you can associate the initial content with a model group, so that the List's initial content will always be synced with the objects you specify in the model group.

The Entry Viewer shows you a live list of the entries currently on a list. To open this window, go to the General tab of a list's properties, and press View Entries...

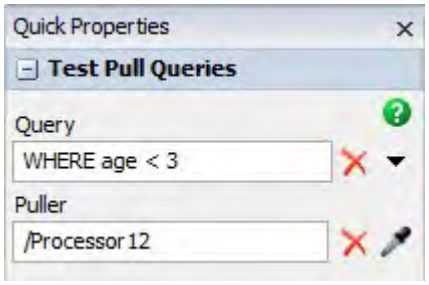

ItemList1 Entries

value	itemType	age	distance	queueSize
/Queue9/Box	1	6.20	Puller Required	4
/Queue8/Box	2	6.04	Puller Required	2
/Queue3/Box	1	5.07	Puller Required	1
/Queue9/2	2	3.49	Puller Required	4
/Queue9/3	3	2.56	Puller Required	4
/Queue9/4	2	1.97	Puller Required	4
/Queue8/2	3	1.86	Puller Required	2

Columns

The table's first column shows the value for each entry, and additional columns show the field value for each defined field in the list.

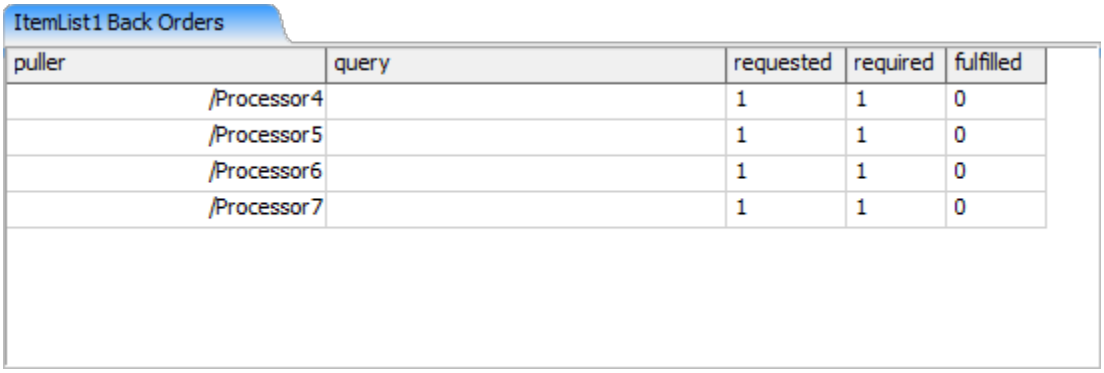
Test Pull Queries



The Quick Properties for the Entry Viewer allow you to test out pull queries before implementing them in the simulation logic. The connectionless routing example gives an example of how you might use these tools.

- Query This field allows you to test out various queries on the list. Enter the query you want to use and press Enter to update the list view. Use the button on the right for help in forming the query. The table view will update based on the query you define, filtering and prioritizing the entries as dictated by the query.
- Puller Use this field to define who the puller is. Some of the fields in the view will show *Puller Required*, meaning that the value for that field is dependent on who is pulling from the list. Press the button on the right, then click on an object in the model to define the puller. Puller-dependent fields in the list will update to be based on the puller you define.

The Back Order Viewer shows you a live list of the back orders currently on a list. To open this window, go to the General tab of a list's properties, and press View Back Orders...



Columns

For each back order, the table shows the puller, the query, the number of requested entries, the number of required entries, and the number fulfilled so far. Usually objects pull one at a time, so requested, required, and fulfilled will often be 1, 1, and 0 respectively.

Commands

The following are commands that are used with Lists. Refer to the command documentation for more information on using these commands.

- listpull
- listpush
- listentries
- listbackorders
- listremove

Query Syntax

In a pull request you may include an SQL query that filters and prioritizes what you want to pull from the List. The query usually will include a WHERE, and/or ORDER BY clause. In some special cases you may use the SELECT clause. It has a special meaning that will be explained later in this topic.

```
WHERE queueSize < 5 ORDER BY age ASC
```

This reference documents only information specific to FlexSim's SQL implementation. Refer to commonly available documentation for more general information on the language.

Using List Fields

By default, when you reference a field in a WHERE or ORDER BY clause, it will refer to one of the defined fields on the List.

```
ORDER BY pushTime
```

Here, the ORDER BY clause will prioritize by the List's pushTime field in ascending order (in SQL the default is ascending), assuming the pushTime field is a defined field on the List.

Accessing Labels

If a field that you define in your SQL query is not a defined field on the List, then the query will retrieve the label value of the same name on the entry's value object.

```
WHERE SKU = "05692AQD"
```

In this example, if SKU has not been defined as a field on the List, and let's say it is an Item List, the query will retrieve the label named "SKU" on the items that have been added to the List, and compare it against "05692AQD".

Puller and Value Table Specifiers

You can also access labels on the puller object using a keyword "Puller" table specifier, followed by a dot.

```
WHERE step = Puller.step
```

Here the step field (either a defined List field or a label on the value object) is compared with the step label on the puller object. Here we use "table specifier" in reference to SQL's syntax for accessing a database table. You might conceptualize the List as a database table with the name "Value", and you are kind of doing an inner join with a table with one row in it called Puller, where the labels on the puller object are like fields of the Puller table.

By default if you leave a table specifier out, it assumes it is a defined field on the List or a label on the value object. However you can use a keyword "Value" table specifier explicitly for better readability.

```
WHERE Value.step = Puller.step
```

Here you can either use lower case value or upper case Value, and lower case puller or upper case Puller, according to your preference. Here at FlexSim we usually capitalize the table and lower-camel-case the fields.

Using FlexScript Commands in SQL Queries

You can also use FlexScript Commands in your SQL queries.

```
ORDER BY round(distance / 100) ASC, age DESC
```

This ORDER BY clause takes the distance field and categorizes it into distance ranges 100 units wide, so that a distance of 130 would be tied with a distance of 95 (round(130 / 100) == round(95 / 100), but would get higher priority than a distance of 200 (round(130 / 100) < round(200 / 100)). Ties would then use the second priority clause and be prioritized by highest age.

```
ORDER BY value.type DESC
```

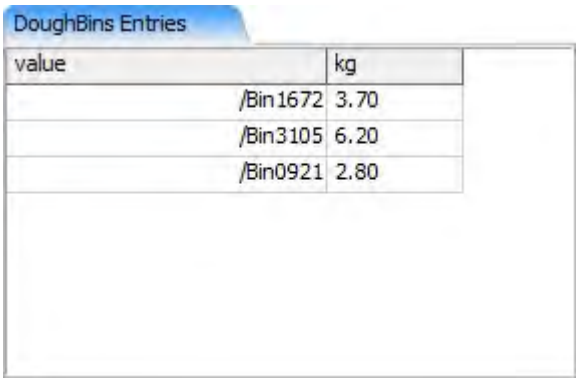
This ORDER BY clause gets the item type on the entry value object. Note that this ORDER BY clause would be exactly the same as if you were to use the standard Item Type field on an Item List. This example just shows a different way of doing it.

Using the SELECT clause

When pulling from lists with a query, the SELECT clause takes on a special meaning. It allows you to pull things from the list using fluid-like request quantities, instead of discrete entries on the list. Take for example the following listpull() command usage.

```
listpull("DoughBins", "SELECT kg", 4.5, 4.5)
```

And assume that the entries on the DoughBins list look like the following when pulled.



Here, instead of trying to pull a number of discrete entries from the list, the listpull() command is trying to pull 4.5 total kg units from the list, no matter how many discrete entries it requires. In this case, the pull operation will pull the full 3.7 kg from Bin1672. Next, it will pull the remaining 0.8 kg from Bin3105. The listpull() command will then return back an array consisting of both Bin1672 and Bin3105.

Overflow Management

This example naturally leads to the question of how the list manages left over amounts, namely the remaining 5.4 kg on Bin3105. And what does the list do about removing entries from the list? Obviously it should remove Bin1672 from the list because it has pulled its full kg amount. But what should it do about Bin3105, which technically has some "overflow" that hasn't been pulled off the list. Should it remove Bin3105 from the list or leave it on? The answer to that depends on the nature of the fields that you are querying in the SELECT clause.

Trackable Overflow

The list will track overflow amounts if the field defined in the SELECT clause meets one of the following requirements.

1. The clause references a dynamic label field.
2. The clause references a label that is not explicitly defined as a field on the list.
3. The clause references a non-dynamic field.

If one of these requirements is met, the list will track the values of the fields, decreasing them until they are 0, at which point it will remove the entries from the list. If labels are used (options 1 or 2), then as the pull operation finds valid entries to pull, it will decrement the value of the label by the amount that is pulled. So in this example, the kg label on Bin1672 will be decremented to 0 and Bin1672 will thus be removed from the list. Then the kg label on Bin3105 will be decremented to 5.4, and since it is still greater than 0, it will remain on the list. Subsequent pull operations will pull from the list with Bin3105's kg quantity properly decreased.

When non-dynamic fields are used (option 3), the list stores its own cached value for the field, so the pull operation will decrement its own cached value as quantities are pulled from the list. Admittedly this makes the term "non-dynamic" a bit of a misnomer because the values do change while on the list. In this case, think of the fields instead as "cached" or "non-reevaluated" fields.

Untrackable Overflow

SELECT clauses where the list does not manage overflow amounts include the following:

1. The clause references a dynamic non-label field.
2. The clause is an SQL expression that does not reduce to a single field, i.e. `SELECT length * width`.

When SELECT clauses with untrackable overflow are used, any entries containing overflow amounts will be removed from the list when they are returned. This means that in the above example, Bin3105 will be removed from the list, even though it technically has 5.4 kg left over.

SELECT Clause Constraints

In list pull operations, the SELECT clause should reference only one "column". In other words, it should not be a comma-separated list of columns like normal SQL queries. Additional columns will be ignored.

Using Lists in the query() Command

You can also integrate global lists in sql queries in the query() command. Just include the List as a table named in the FROM clause, and then access its fields normally. Note that here you should not use pullerdependent fields. If it is a partitioned list, then you should encode the table name as ListName.$1, and then pass the partition ID as the first additional parameter to query(). Or, if the partition ID is a string,

you can encode the string directly into the table name. For example, the table named ListName.Partition1 means it will use the global list named "ListName", and the partition with ID "Partition1".

List Statistics

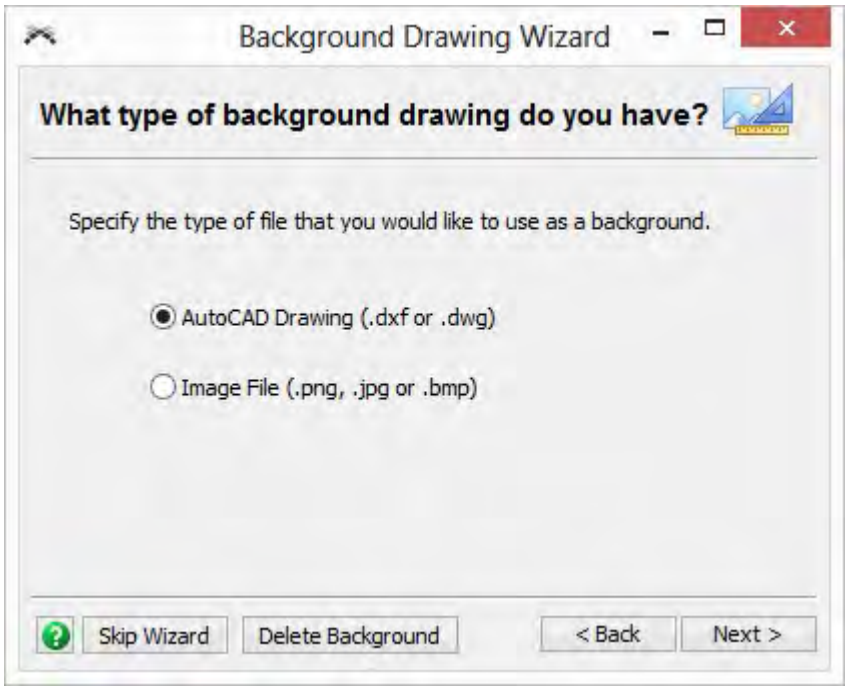
Lists provide several statistics. You can use these statistics for gathering data about a model, using the getstat() command, or you can subscribe to these statistics as events, listening for when they change. Below are the set of statistics provided by lists:

- Input The number of values that have been pushed onto the list.
- Output The number of values that have been pulled from the list.
- Content The number of values currently on the list.
- Staytime Records the time that entries stay on the list.
- Back Order Input The number of back orders that have been created for the list.
- Back Order Output The number of back orders that have been fulfilled on the list.
- Back Order Content The number of back orders currently on the list.
- Back Order Staytime Records the time that back orders wait to be fulfilled.

Additionally, lists provide per-partition statistics. These statistics have one requirement (p1 in the getstat() command), namely the partition ID. If you only use the list's default partition, these statistics will be the same as the list's main statistics.

- Partition Input The number of values that have been pushed onto the partition.
- Partition Output The number of values that have been pulled from the partition.
- Partition Content The number of values currently on the partition.
- Partition Staytime Records the time that entries stay on the partition.
- Partition Back Order Input The number of back orders that have been created for the partition.
- Partition Back Order Output The number of back orders that have been fulfilled on the partition.
- Partition Back Order Content The number of back orders currently on the partition.
- Partition Back Order Staytime Records the time that back orders wait to be fulfilled on the partition.

Model Background



The Model Background can be accessed from the Toolbox. (View menu > Toolbox > Add > Visual > Model Background). or by dragging and dropping a Background object from the Visual section of the Library Icon Grid.

Note: Dragging and dropping a background object into a model already containing a background will open the previously created background's properties window.

The Model Background is just a Visual Tool object that displays either an image or an AutoCAD drawing on the model floor.

Image Files

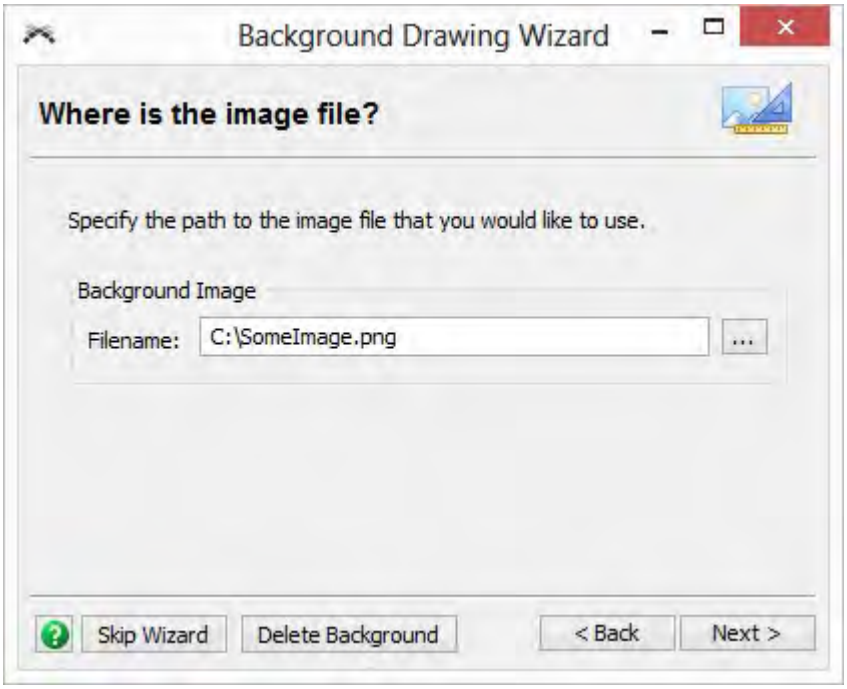
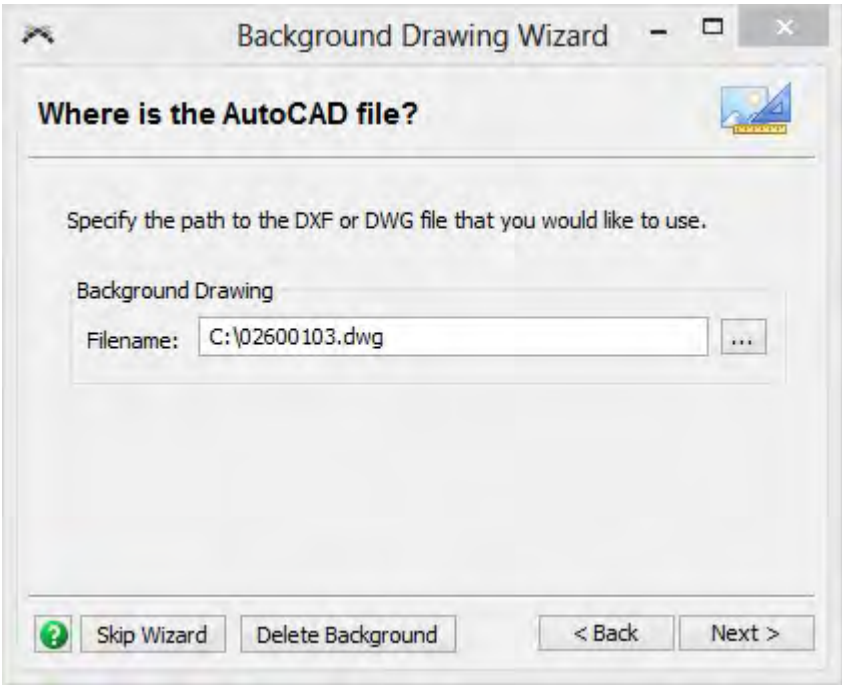


Image files may be png, jpg, or bmp files.

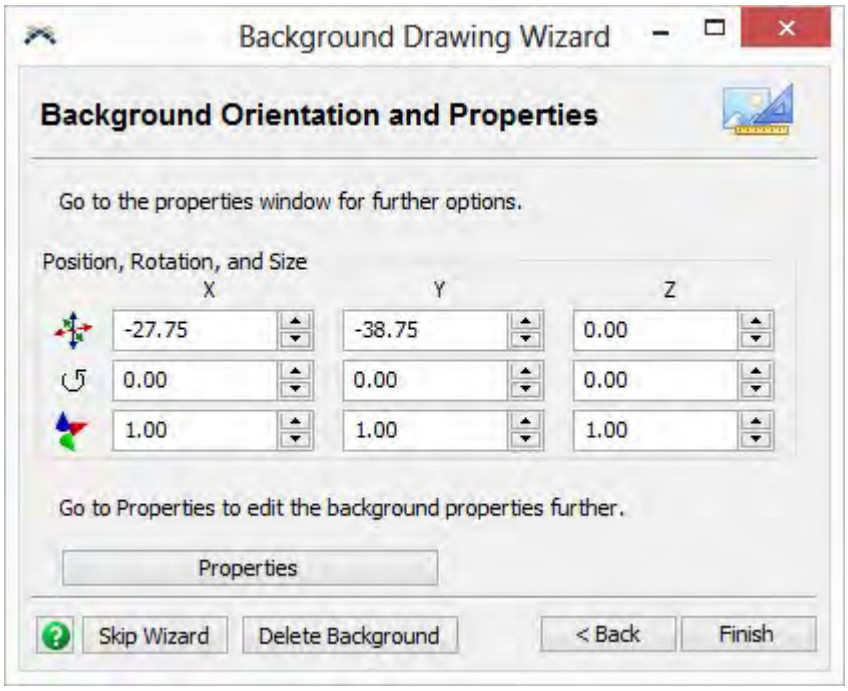
AutoCAD Files



AutoCAD files may be dxf or dwg. DWG files will load and render faster, and you can customize their layer visibility and colors.

Note:
The DWG renderer is a custom third-party renderer that controls its own view frustum and occlusion clipping planes using deprecated OpenGL functions, which causes issues when trying to render in stereoscopic 3D or VR, and it does not work with a Modern (Core Profile) OpenGL Context.

Setting the Position, Size and Rotation

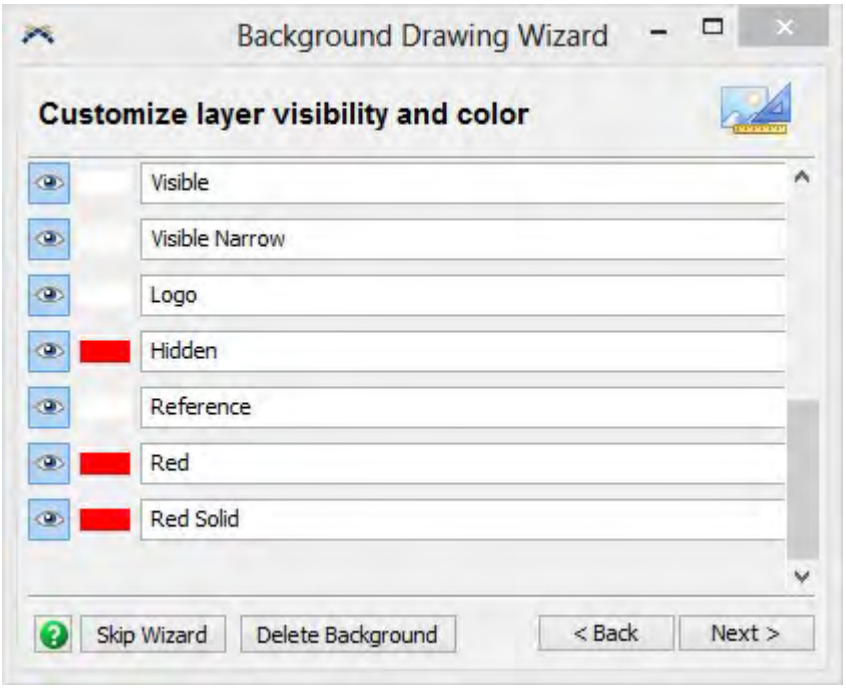


Some files may not import with the correct position, size or rotation. This can easily be fixed through this final page.

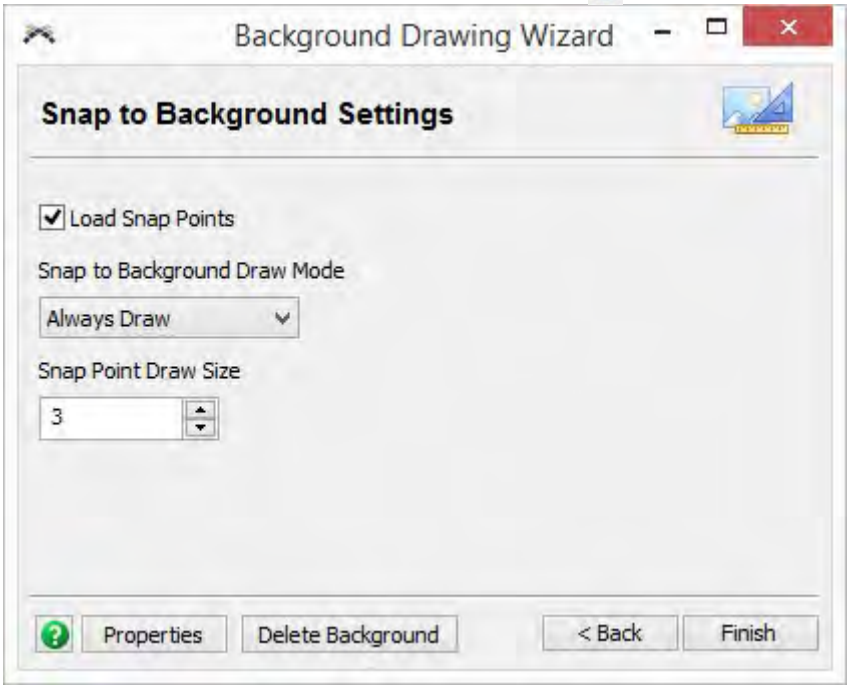
DWG and DXF files both have length units associated with the file. When you import these into FlexSim, you want those units to match your model's units. Let's say you are importing a DWG or DXF file that is the layout of your company's factor floor. If your FlexSim model units are set to meters and the DWG or DXF file you were given was set to feet, the image will be scaled too large. This is because FlexSim imports the model as a 1-1 unit ratio. So 1 foot is equal to 1 meter. To scale your DWG or DXF file to the correct size, for this example, you would set the x, y and z scale to 0.3048.

DWG Layers and Snap Settings

DWG files contain multiple layers which may be customized within FlexSim.



You can turn layers on an off by clicking on the button. Next to the eye is a color well where you can define that layer's color.

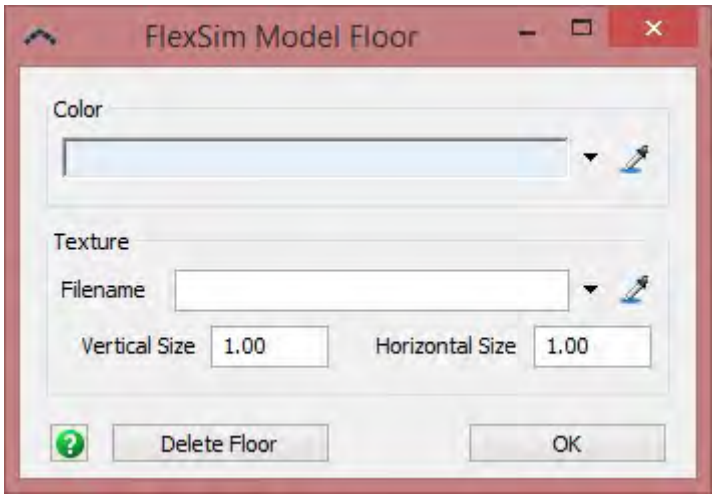


Load Snap Points - Check this box to load data into the tree containing points from the DWG file. You must load snap points to enable Snap to Background in the 3D view. If you change layer visibility, clear this box and check it again to load updated snap points.

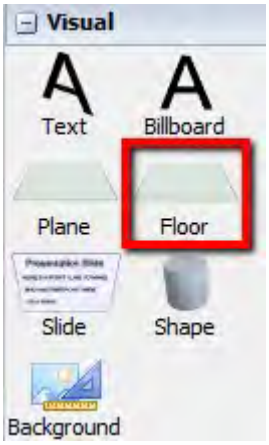
Snap to Background Draw Mode - Select how the snap points should be drawn when Snap to Background is enabled.

Snap Point Draw Size - The size that the snap points should be drawn.

Model Floor

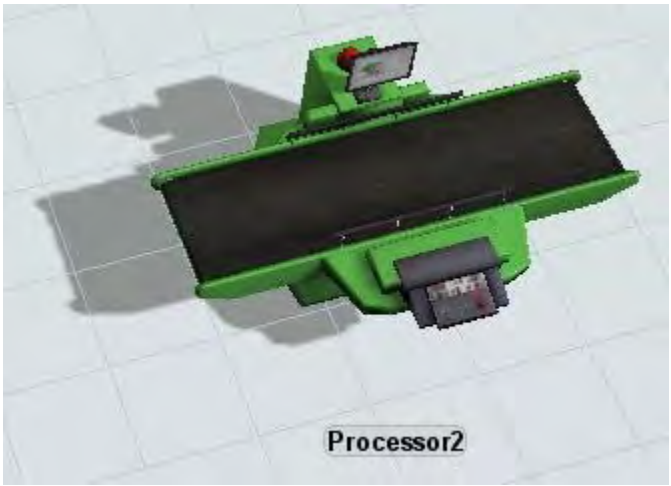


The Model Floor can be accessed from the Toolbox. (View menu > Toolbox > Add > Visual > Model Floor). or by dragging and dropping a Floor object from the Visual section of the Library Icon Grid.



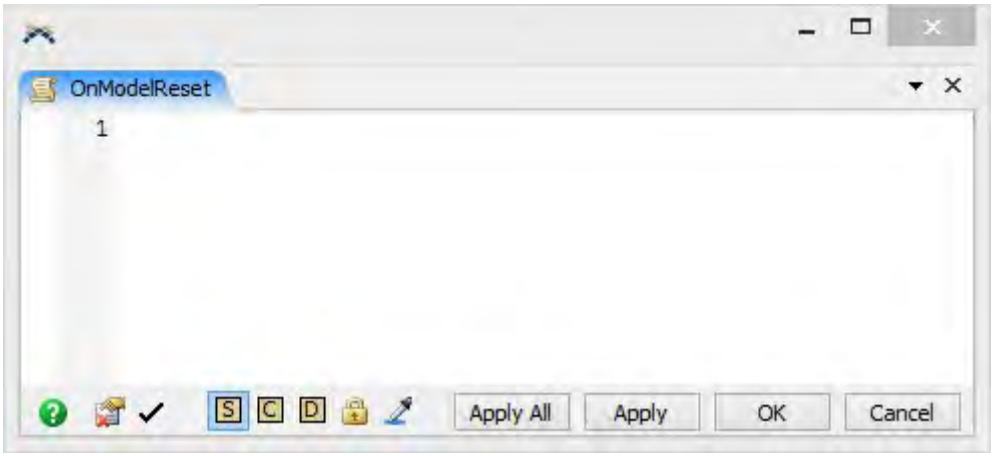
Note: Only one floor object may be added to a model. Dragging and dropping a floor object into a model already containing a floor will open the model floor's properties window.

The Model Floor is a Visual Tool object that displays a color and/or image texture as the "floor" of the model. The floor extends out from all directions as far as the eye can see. It allows you to see shadows from model objects.



If no texture is defined, only the color will be used to draw the floor. If an image texture is defined, the vertical and horizontal repeat size of the image may be defined.

Model Triggers



Model Triggers are accessed from the Toolbox. (View menu > Toolbox > Add > Modeling Logic > Model Triggers).

Model triggers allow you to execute code at different points between model runs. The following triggers are available:

On Model Open - This trigger is fired when the model is opened from the file.

On Reset - This trigger is fired when the model is reset.

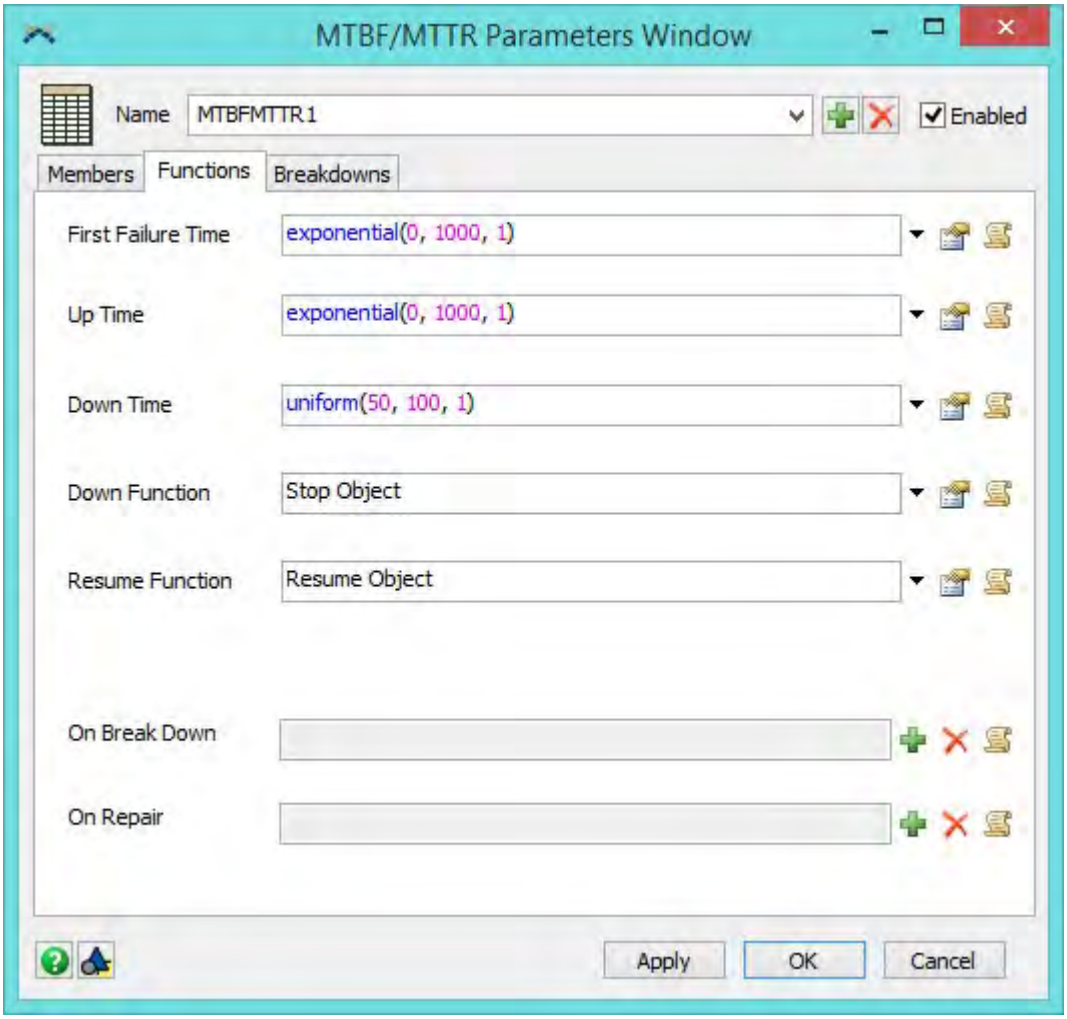
On Run Start - This trigger is fired whenever the model changes from a stopped or paused state to a running state.

On Run Stop - This trigger is fired whenever the model changes from a running state to a stopped or paused state.

On Post-Compile - This trigger is fired after the completion of a compile (see Build menu and When to Compile).

For more available triggers, see the triggers section of the Model Libraries Node page.

MTBF/MTTR



MTBF/MTTR objects are accessed from the Toolbox. (View menu > Toolbox > Add > MTBF MTTR).

MTBF/MTTR objects are used to set random breakdown and recovery times for groups of objects in the model. Each MTBF/MTTR object can have any number of object members 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 and what behaviour they should perform. A model may contain any number of MTBF MTTR objects.

Though similar to the Time Table, the MTBF/MTTR object uses picklists to determine dynamically when the connected members will break down and how long they will be broken down for. You can also specify more specific information about the breakdowns. This includes specifying if all connected members will breakdown together, or if the breakdown times will be individually calculated for each object (firing the First Failure Time, Up Time, and Down Time picklists once per object). If you only want the connected members to count specific states towards their Up Time, for example a Processor when it is in the *processing* state, this can be specified as well. This means when the Processor is *idle*, elapsing time won't count towards the Processor's Up Time until it enters the *processing* state.

Pages

- Members
- Functions
- Breakdowns

Name - The name of the MTBF/MTTR. The combobox has a list of all MTBF/MTTR objects in the model, allowing you to quickly jump to different MTBF/MTTR objects.

- Create a new MTBF/MTTR object.

- Delete the current MTBF/MTTR.

Enabled - Specifies whether the MTBF/MTTR should execute it's down times for all of the members of the MTBF/MTTR. The Experimenter allows you to enable and disable MTBF/MTTRs for different scenarios.

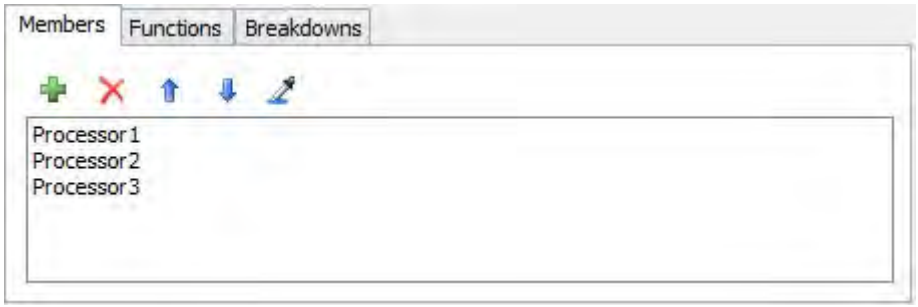
- Adds the MTBF/MTTR to a User Library as either a Draggable Icon or an Auto-Install Component.

Apply - Saves all changes to the MTBF/MTTR.

OK - Saves all changes to the MTBF/MTTR and closes the window.

Cancel - Cancels any unsaved changes made to the MTBF/MTTR and closes the window.

Members Page



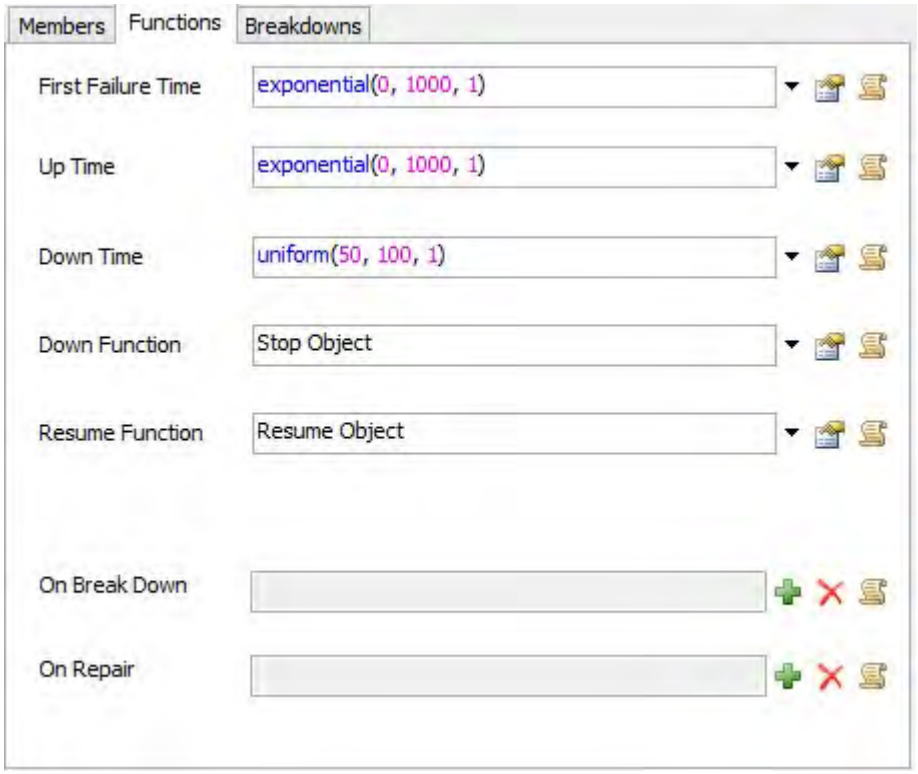
- This will open an object selection GUI where you can select multiple objects in the model.

- Removes the selected member(s) from the list.

- Reorder's members Up or Down in the list.

- Click to enter "Sample" mode, then click on any object in the model to add it as a member.

Functions Page



The following picklists can be fired individually for each object, or for all the objects together depending on the checked state of Break down members individually from the Breakdowns Page.

First Failure Time - This picklist returns the time of the first failure. Returning a negative number will cause the first failure to be ignored.

Up Time - This function determines how long the objects controlled by this MTBF MTTR object will run before they go into a broken-down state. The Up 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.

Down Time - This picklist 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.

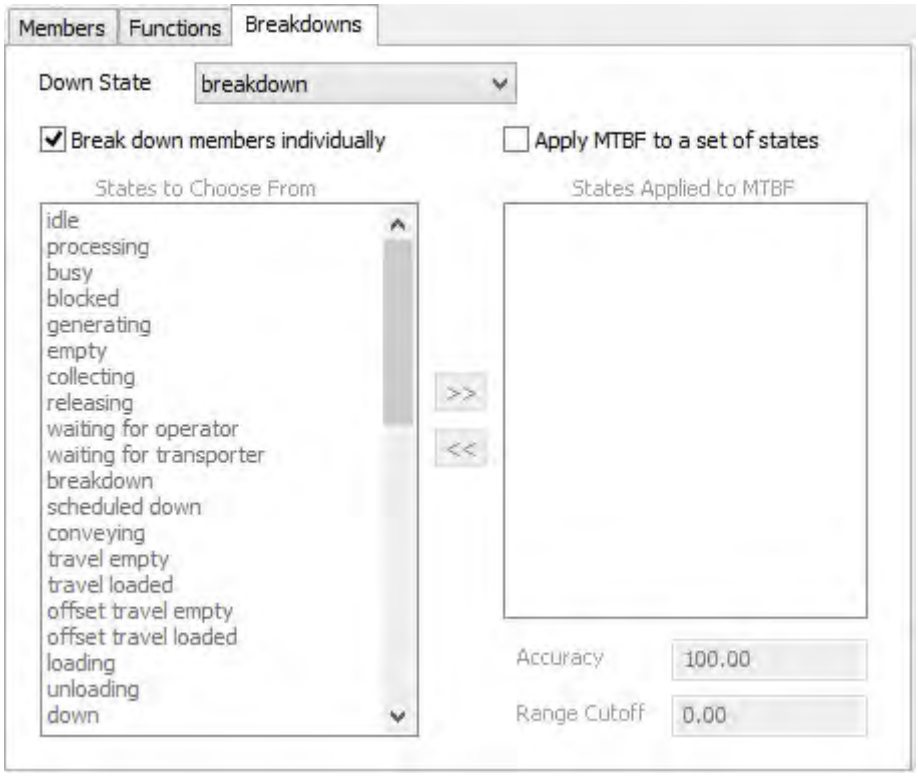
Down Function - This picklist is executed when the objects in the member list go down. It is executed once for each object in the member list. Here is where you specify what to do to stop the object.

Resume Function - this picklist 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 picklist is fired immediately after the Down Function, but it is only executed once, instead of once for each object. See Down/Resume Trigger.

OnRepair - This picklist is fired immediately after the Resume Function, but it is only executed once, instead of once for each object. See Down/Resume Trigger.

Breakdowns Page



Down State - This specifies the state that the object will go into when it goes down.

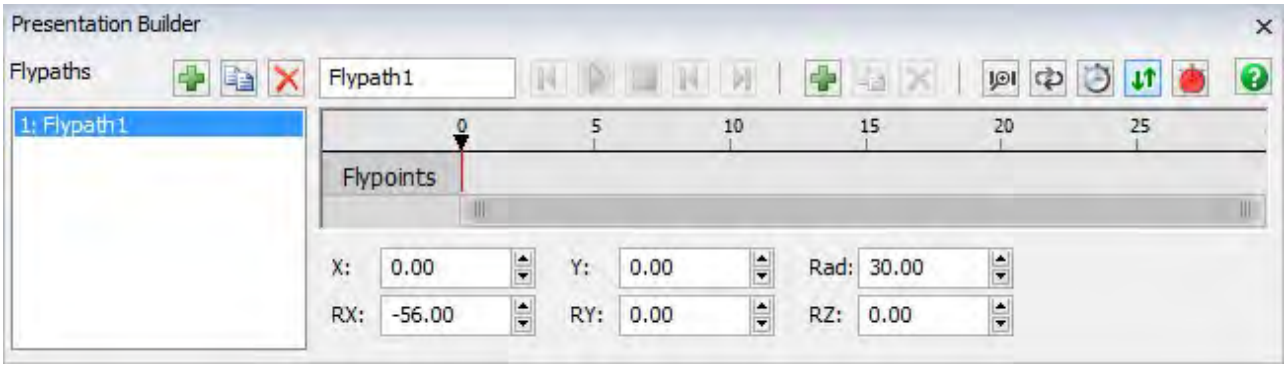
Break down members individually - If this box is checked, the MTBF/MTTR object will create a separate thread of down and resume events for each member object. If it is not checked, all member objects will go down and resume at the same time.

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 object's states for its Up 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 change the accuracy value to 5, then the MTBF will do much fewer checks before bringing the object down.

Range Cutoff - This field only applies if the MTBF/MTTR uses a subset of the object's states for its Up 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 object's 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.

Presentation Builder



The Presentation Builder is accessed from the Toolbox. (View menu > Toolbox > Add > Visual > Fly Path). The Presentation Builder that will assist you in developing a fly-thru presentation of the model. You can create multiple flypaths each with their own set of flypoints. When run, the 3D view will sequentially fly or move to each flypoint in the flypath. When used with the presentation slide option of the visual tool the presentation builder can develop PowerPoint™ style presentations in 3D. If you do not have a 3D view active, the Presentation Builder will be grayed out.

By default, flypaths are not associated with simulation speed, instead flypaths travel in real time. That setting can be changed by toggling the button. Flypaths also do not start/stop when the model is started/stopped. The Presentation Builder has its own set of Start and Stop buttons. However, when using the Video Recorder, you can specify one your flypaths to be run while recording your video file.

Creating a Flypath

Creating a flypath is easy in FlexSim. Once you have the Presentation Builder open, move/rotate the 3D view to the position you would like the flypath to start at. Press the to add a flypoint. This is similar to a keyframe in the Animation Creator. Notice that the Presentation Builder automatically moved the flypath time cursor two seconds ahead of your created flypoint. This makes it easy to add multiple flypoints very quickly. Move to your next desired position/rotation and click the again. Continue this process until you have created a complete flypath.

You can always insert new flypoints by moving the time cursor to any spot on the time line and hitting the . Delete flypoints by selecting them and pressing the or hitting the Delete key.

You can change the zoom of the timeline by either using the mouse scroll wheel or dragging the ends of the bottom scrollbar.

Note when using the scroll wheel: The timeline must be the active view in order to receive mouse scroll events. You may need to click on the timeline to make it the active view if zooming is not occurring.

Updating Flypoints

Once a flypoint has been created, the time cursor must be directly over the flypoint in order to update it. When the time cursor is directly over a flypoint, the flypoint diamond will change to a hollow diamond and the position and rotation boxes below will change to red text. The position/rotation boxes display that flypoints current position. You can edit those fields directly to change the position of your flypoint, or you can reposition the 3D view to update your flypoint.

You can move flypoints anywhere along the timeline by clicking and dragging the flypoint to a new position.

Selecting Flypoints

Hold the CTRL key down and click on flypoints to select multiple flypoints.

Hold the ALT key down and click a flypoint to select all flypoints from the clicked flypoint to the end of the flypath.

Running Fly Paths in the 3D View

Aside from the controls in the Presentation Builder, you can also fly through flypaths by pressing keys on the keyboard. This is useful when you don't want to have the Presentation Builder open. Press one of the numbers: 1-9 to run the associated flypath (plays the flypath from the list at the given rank number). Press the space bar or the 'N' key, and the view will run the next flypath. Press the 'B' key and the view will go back to the previous flypath.

Reference

Flypaths

- Adds a new flypath.

- Duplicate the selected flypath.

- Removes the selected flypath.

Flypath List - The list of the model's flypaths. Select a flypath to view its properties in the Flypoint Editor.

Flypoint Editor

- Moves the time cursor to the first flypoint in the flypath.

- Runs the flypath from the current time.

- Stop the current flypath.

- Moves the time cursor to the previous or next flypoint.

- Adds a new flypoint at the time cursor's current position.

- Duplicates the selected flypoints.

- Removes the selected flypoints (same as Delete key).

- Zooms the timeline in or out to make all flypoints fit on the screen.

- Causes the flypath to loop back to the beginning once it hits the last flypoint.

 - If toggled, the flypath will use the model's run speed instead of real time. This can be useful when working with the Video Recorder. By default this option is off and the time displayed in the timeline is in real seconds.

 - Sync the 3D View. If toggled, the 3D view will update it's position and rotation as the current time cursor is moved along the timeline.

 - If toggled, moving or rotating the 3D perspective view will cause a new flypoint to be created if not flypoint exists at the current time.

Timeline - The timeline displays a list of times at the top in seconds. Below that are all the flypoints, or keyframes. You can add any number of flypoints to a flypath.

X - This field specifies the x location of focus point of the camera.

Y - This field specifies the y location of focus point of the camera.

Rad - The radius field specifies the distance the camera is away from its focus or rotation point. RX - This field specifies the pitch of the camera.

RY - This field specifies the roll of the camera.

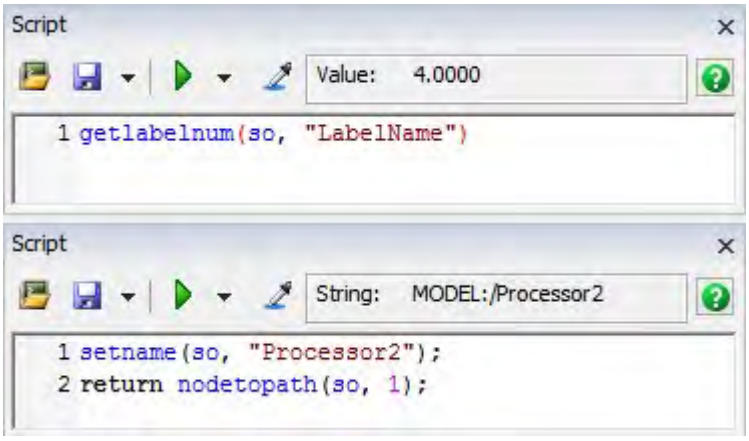
RZ - This field specifies the yaw of the camera.

Script Console

A Script Concsole can be accessed through either the Debug menu > Script Console, or through the FlexSim Toolbar.

The script console allows you to execute flexscript commands on the fly without needing to run your model. This can be useful for getting information from your model as well as configuring your model. Type the flexscript code in the main field at the bottom of the window and press the . If your code has a return value, this value will be displayed in the results field. If you are executing a command with a return value like, model().find("Processor1") and your code consists of only one line, you can ommit the return and the semi-colon at the end of the line to see the return value in the results field.

You can use the Sampler button to reference objects and paths in your model. For more information, see the Sampler page.



If you need to debug your code, you can enter debugging mode by clicking the of the but and select Debug. This will place a debug point at line 1 of your code, open it in a Code Editor and execute the code.

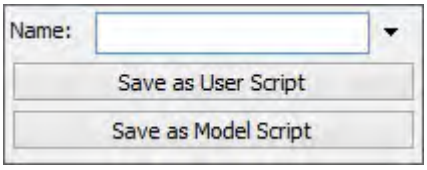
Saving and Loading Scripts

Saving

Code entered into the Script Console can be saved as either a Model Script or a User Script by pressing .

- Saves the current script. If the script has not yet been saved, opens the Save As popup.

- Opens the Save As popup.



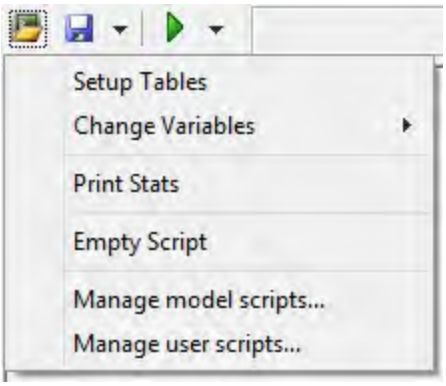
A list of previously saved scripts is available by pressing the ▾.

Save As User Script - These scripts are saved in the user preferences folder (VIEW:/enviornment). These scripts are available for all models while FlexSim is open under your user.

Save As Model Script - These scripts are saved in the model's /Tools/Scripts folder. They are only available for the current model.

Loading

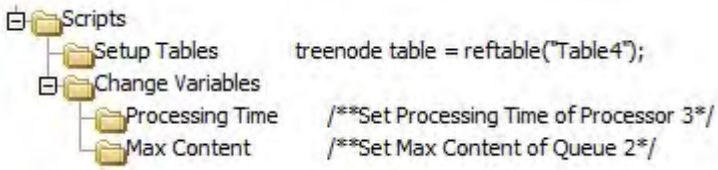
Saved scripts can be loaded by pressing the button. A menu will appear with a list of all Model Scripts, separated by a line, and then all your User Scripts.



Empy Script - Closes any currently open script and clears the code field.

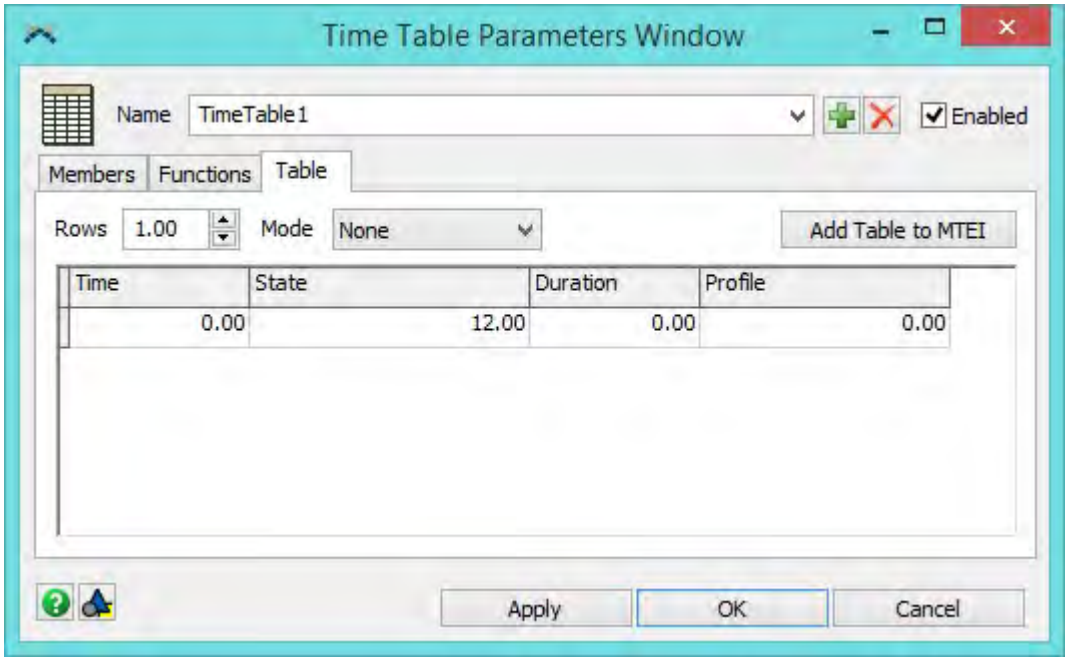
Managing Saved Scripts

As seen in the above image, Change Variables contains a sub-menu of further Model Scripts. You can organize your scripts through any number of sub-menus. To manage your scripts, click the button and select Manage model scripts... or Manage user scripts.... A Tree Window will appear.



From here, you can organize your scripts by creating sub-nodes and renaming any of your scripts. For more information on using the Tree, see the FlexSim Concepts - Model Tree View or the Tree Structure page.

Time Tables Concepts



Topics

- Functions
- Down State and State Profiles
- Modes o None o Daily Repeat o Weekly Repeat o Custom Repeat o Date Based

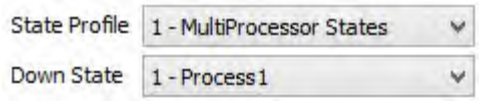
Time Tables are accessed from the Toolbox. (View menu > Toolbox > Add > Time Table).

Time tables 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.

Functions

When the time table hits a down time, two sets of functions are called. First, the Down Function is fired. This happens once for each member of the Time Table. This allows you to stop the associated object, send a TaskExecuter to some specified location, etc. After all of the Down Functions have fired, the On Down function will fire once. The On Down function passes in the list of members and the table row associated with the down time.

Down State and State Profiles



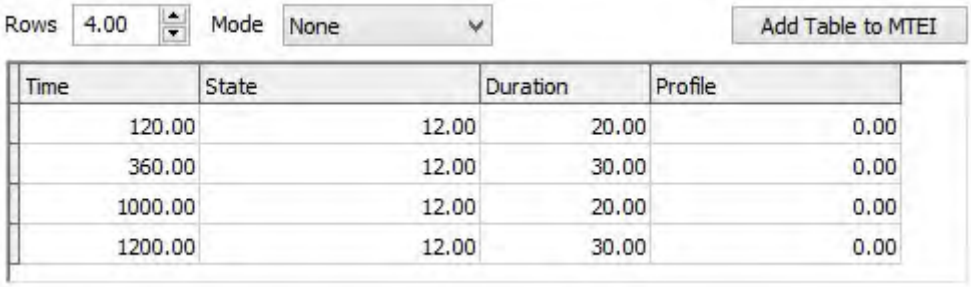
For each down time you can specify the state and state profile that the members will go to during that down period. An object can be tied to multiple Time Tables and be stopped multiple times. Each object stores it's set of states each time it is stopped so the object can then be resumed and move back through the subsequent set of states.

State Profiles: When sending objects to a down state using state profiles, all members of the Time Table should have the associated state profile.

Modes

The TimeTable can be set up in different modes allowing for non-repeating/repeating schedules or schedules based upon date and time. The Daily Repeat, Weekly Repeat, and Date Based modes all utilize the Model Start Time and Date as defined in the Model Settings.

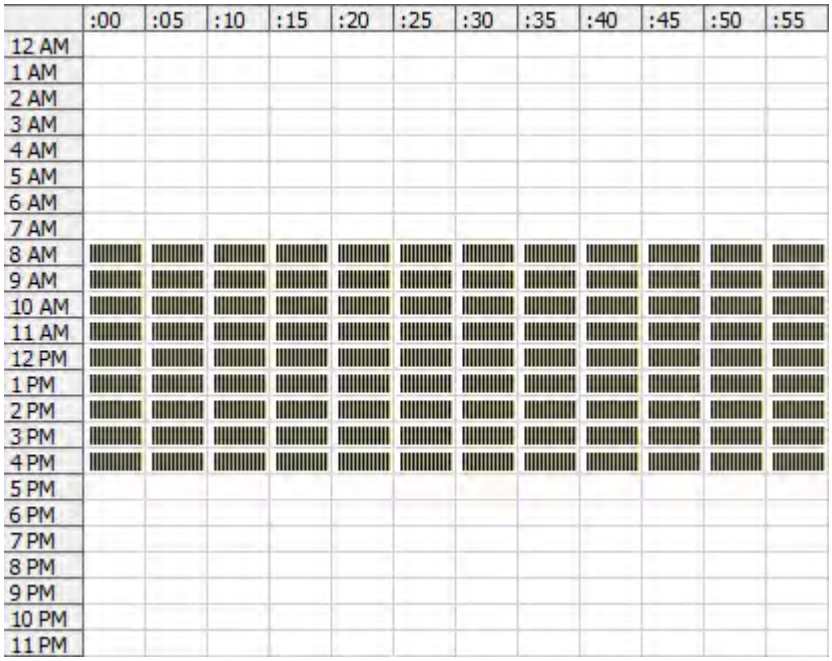
None



Time	State	Duration	Profile
120.00	12.00	20.00	0.00
360.00	12.00	30.00	0.00
1000.00	12.00	20.00	0.00
1200.00	12.00	30.00	0.00

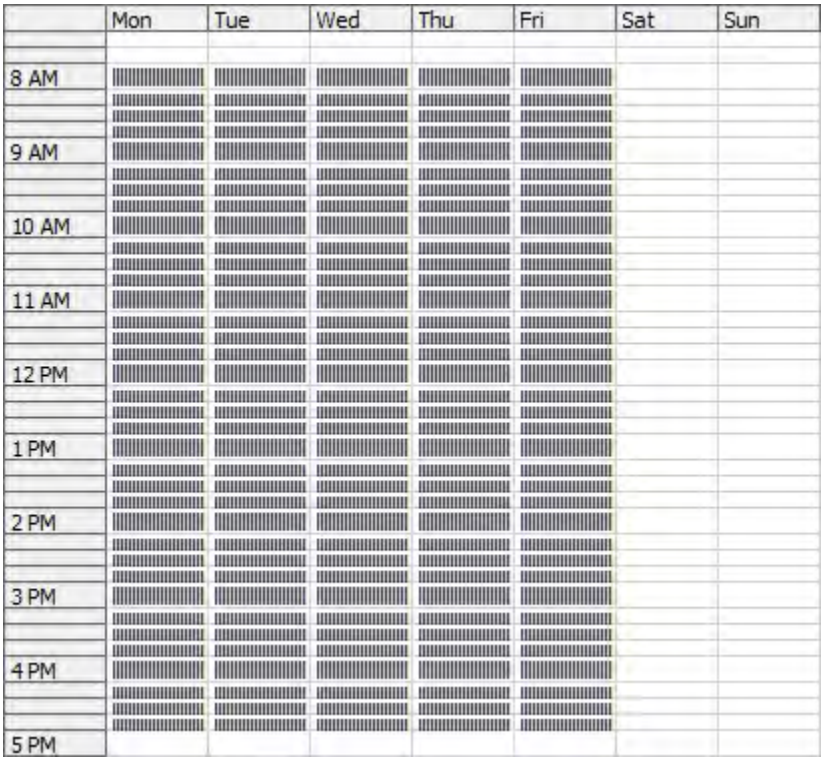
When the mode is set to none, the times listed in the table are absolute times based upon the model time units. The schedule will not repeat.

Daily Repeat



This graphical view allows you to specify the *Operation Time* and *Down Time* If you set your Time Table to repeat daily, the Model Start Time (as defined in the Model Settings). The Model Start Date does not come into effect. For example, if your Model Start Time is set to 08:00:00 AM and your Graphical Table looks like the above table, the members of the Time Table will begin the Model in an Operational state, and no functions will be fired. If however, you change the Model Start Time to 07:00:00 AM, when you reset and run your model, the Down functions will fire and the members will begin in a Down state. One hour later (based on the model time units, so 3600 seconds if the model time is set to seconds), the Resume functions will be fired and the members will begin their Operational Time.

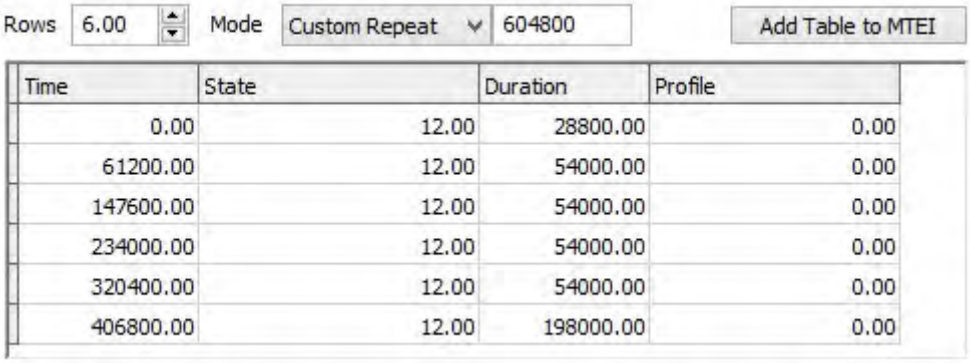
Weekly Repeat



Setting the Time Table to repeat weekly will behave similarly to the Repeat Daily, except that the Time Table will also take into account the Model start *day of the week*. If the Model Start Time begins on a Tuesday at 08:00:00 AM and our Time Table is set to the above values, then the Time Table will skip all of Monday and jump to Tuesday at 08:00:00 AM with the Time Table's members being Operational. When the Model Time hits Friday at 05:00:00 PM, the members will go Down and remain down until Monday at 08:00:00 AM where the Time Table will start over.

Note: If you want to use the Graphical Time Table to build your Time Table, but you don't want to tie into the Model Start Time, you can set the Time Table's repeat time to *Daily* or *Weekly*, make your necessary changes, hit Apply, then set the Repeat time to *Custom*. This will auto fill the numerical table with the correct values associated with the Daily or Weekly table.

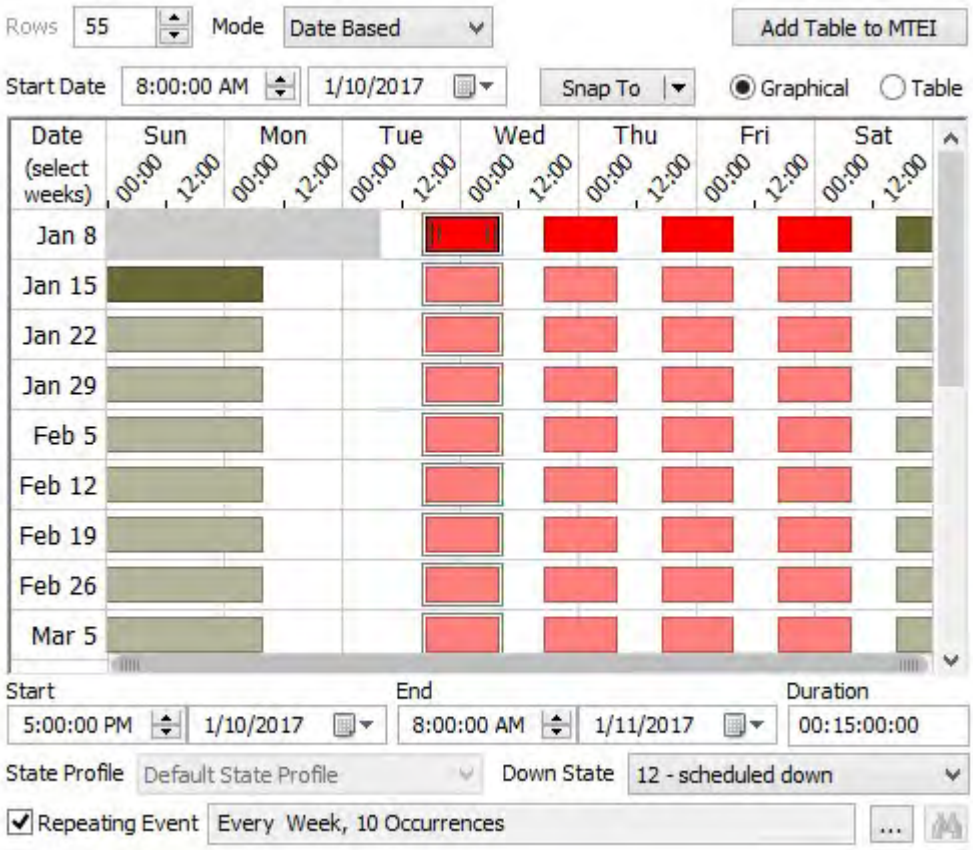
Custom Repeat



Time	State	Duration	Profile
0.00	12.00	28800.00	0.00
61200.00	12.00	54000.00	0.00
147600.00	12.00	54000.00	0.00
234000.00	12.00	54000.00	0.00
320400.00	12.00	54000.00	0.00
406800.00	12.00	198000.00	0.00

The custom repeat mode is similar to the None mode, except you now have the option of specifying the repeat time for the table.

Date Based



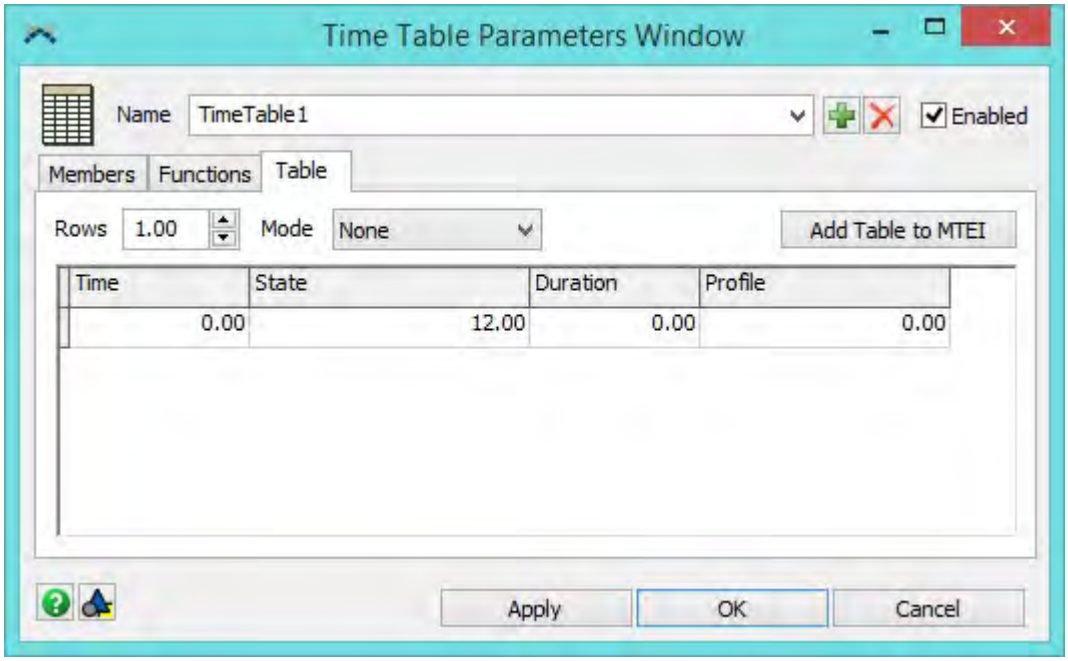
The date based mode allows you to set up down times where each down time is associated with a specific date. These dates are based upon the Model Start Date and Time (as defined in the Model Settings) as well as a Time Table start date and time. This mode does not repeat, however, you can create repeating events.

In the above image, this Time Table has been set to start at 8:00 AM on 1/10/17. The first down time is set for 5:00 PM. If the Model Start Date and Time is set to the same start date and time as the Time Table, the associated members will start operational and go down at 5:00 PM. Changing the Model Start Date and Time from the Model Settings window will allow you to jump into the Time Table's scheduled at the specified time. For example, setting the Model Start Time to 5:00 PM on 1/10/17 will cause the members to start in the Schedule Down state.

Each of the events pictured above are repeating events that repeat weekly for 10 weeks. The lighter colored events indicate that the event is part of a repeated event. You can modify the repeated event all at once, or individual events may be modified.

See the Reference page for more information on how to use the date based interface.

Time Tables Reference



Pages

- Members
- Functions
- Table o Table Editor o Graphical Table o Date Based
    - Date Based Editor View

Name - The name of the TimeTable. The combobox has a list of all TimeTables in the model, allowing you to quickly jump to different Time Tables.

- Create a new Time Table object.

- Delete the current Time Table.

Enabled - Specifies whether the time table should execute it's down times for all of the members of the time table. The Experimenter allows you to enable and disable time tables for different scenarios.

- Adds the Time Table to a User Library as either a Draggable Icon or an Auto-Install Component.

Apply - Saves all changes to the Time Table.

OK - Saves all changes to the Time Table and closes the window.

Cancel - Cancels any unsaved changes made to the Time Table and closes the window.

Members

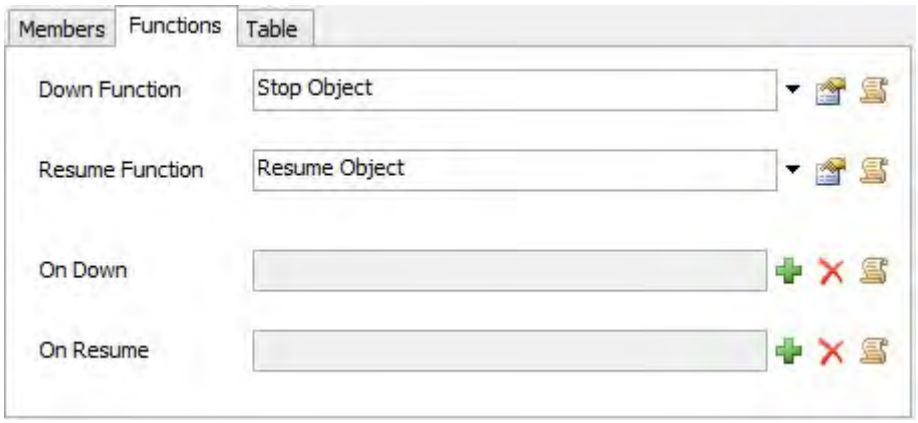
- This will open an object selection GUI where you can selec t multiple objects in the model.

- Removes the selected member(s) from the list.

- Reorder's members Up or Down in the list.

- Click to enter "Sample" mode, then click on any object in the model to add it as a member.

Functions



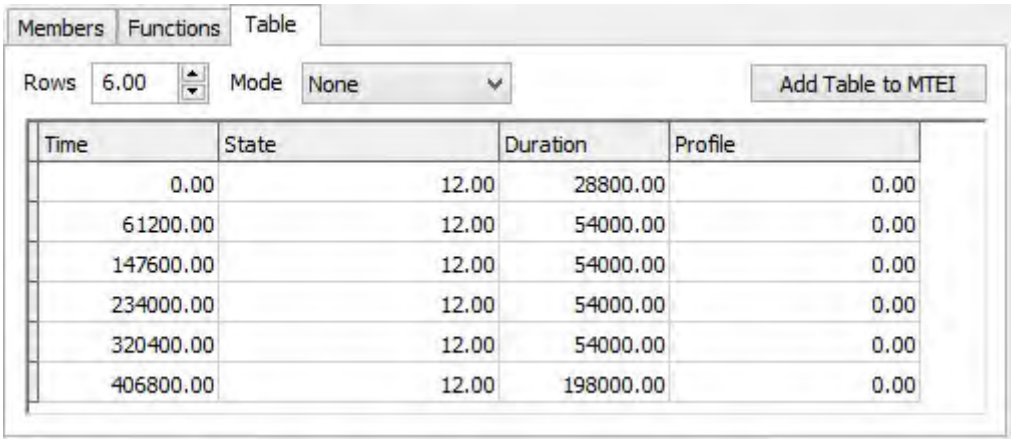
Down Function - This picklist is executed when the objects in the member list go down. It is executed once for each object in the member list. This is where you specify what to do to stop the object.

Resume Function - This picklist 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.

On Down - This picklist is fired immediately after the Down Function has been fired for all objects, but it is only executed once, instead of once for each object in the member list. See Down/Resume Trigger. On Resume - This picklist is fired immediately after the Resume Function has been fired for all objects, but it is only executed once, instead of once for each object in the member list. See Down/Resume Trigger.

Table

Table Editor

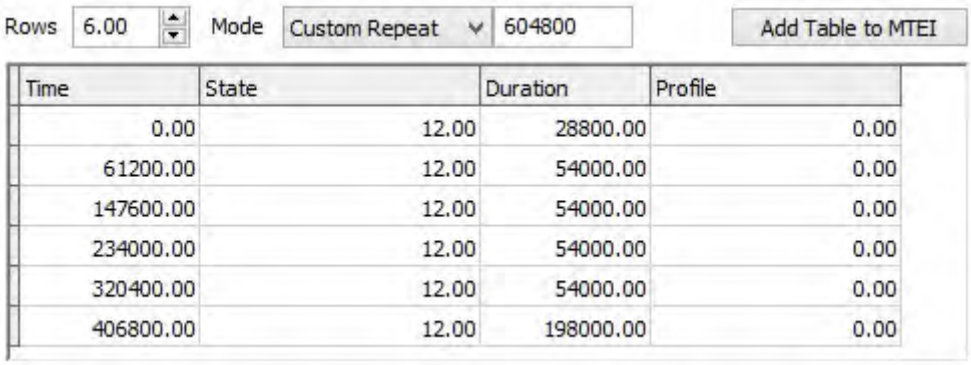


Rows - This is the number of rows in the table.

Mode - This specifies the mode of the Time Table, None, Daily Repeat, Weekly Repeat, Custom Repeat or Date Based.

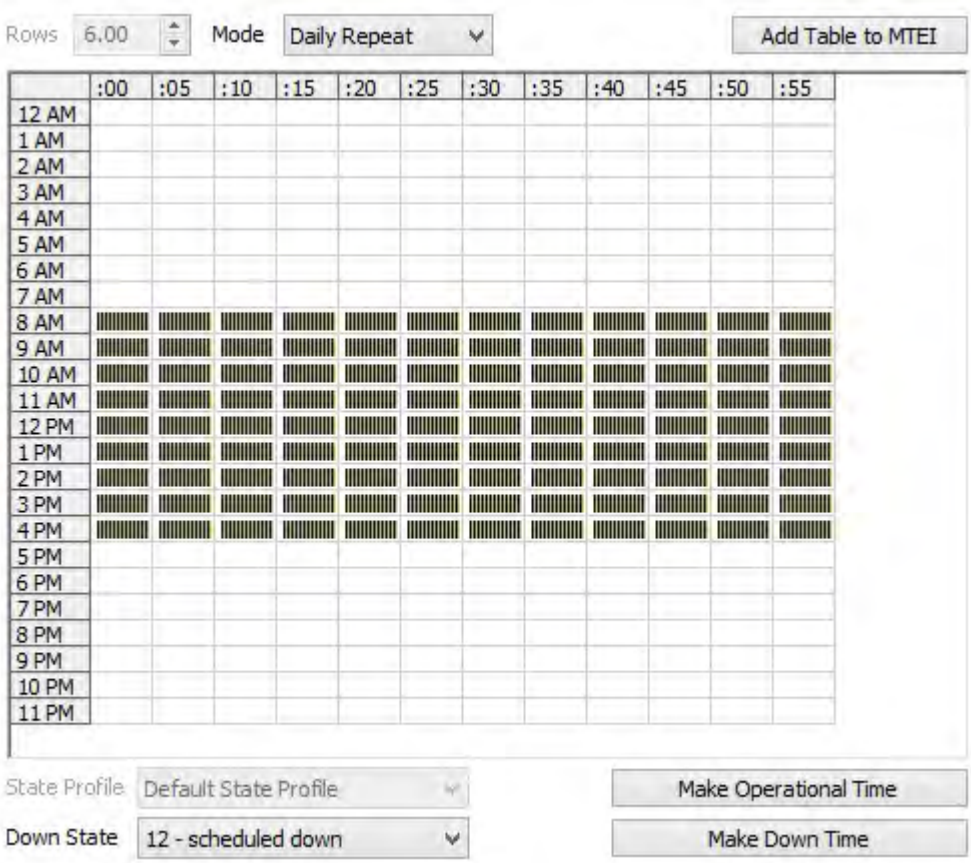
Add Table to MTEI - This button will add the table to the model's multiple table import accessed through the Excel Interface. When used with modes None, Custom Repeat and Date Based, the imported table takes the form of the numeric table (Time, State, Duration, Profile). Table - Each row records the following:

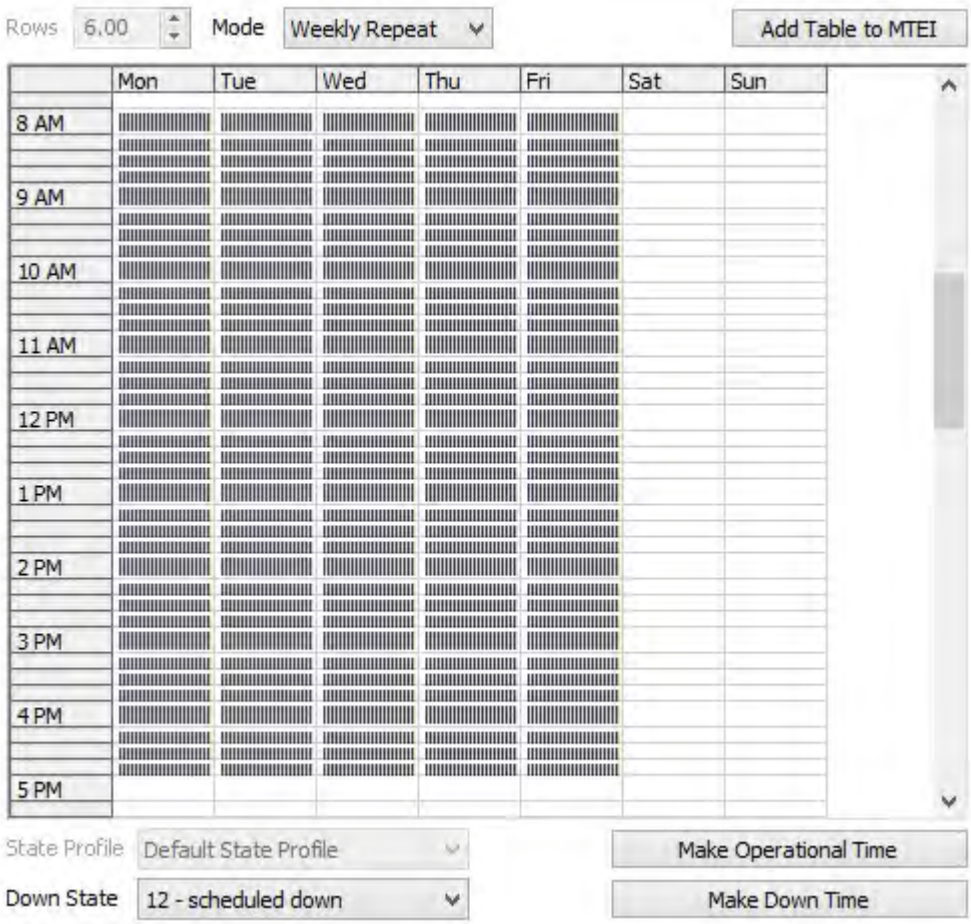
- Time - This is the time since the table began that the state change should occur. If the mode is set to None, the table does not repeat, so the times listed in the Time column are absolute.
- State - This is the state that the objects controlled by this table will change into when the time table tells 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.
- Duration - This is how long the objects will stay in the new state before changing back to their original state.
- Profile - This is the state profile that the Down State is associated with. If you click on this column, a drop-down box will appear at the top, giving you a list of possible state profiles. Changing the State Profile will update the State column drop-down.



Repeat Time - Specify the time, in model time units, to repeat the Time Table.

Graphical Table





For more information on editing the Graphical Table, see the Concepts page.

Add Table to MTEI - This button will add the table to the model's multiple table import accessed through the Excel Interface. This will add the Graphical Table to the MTEI (not the numeric table).

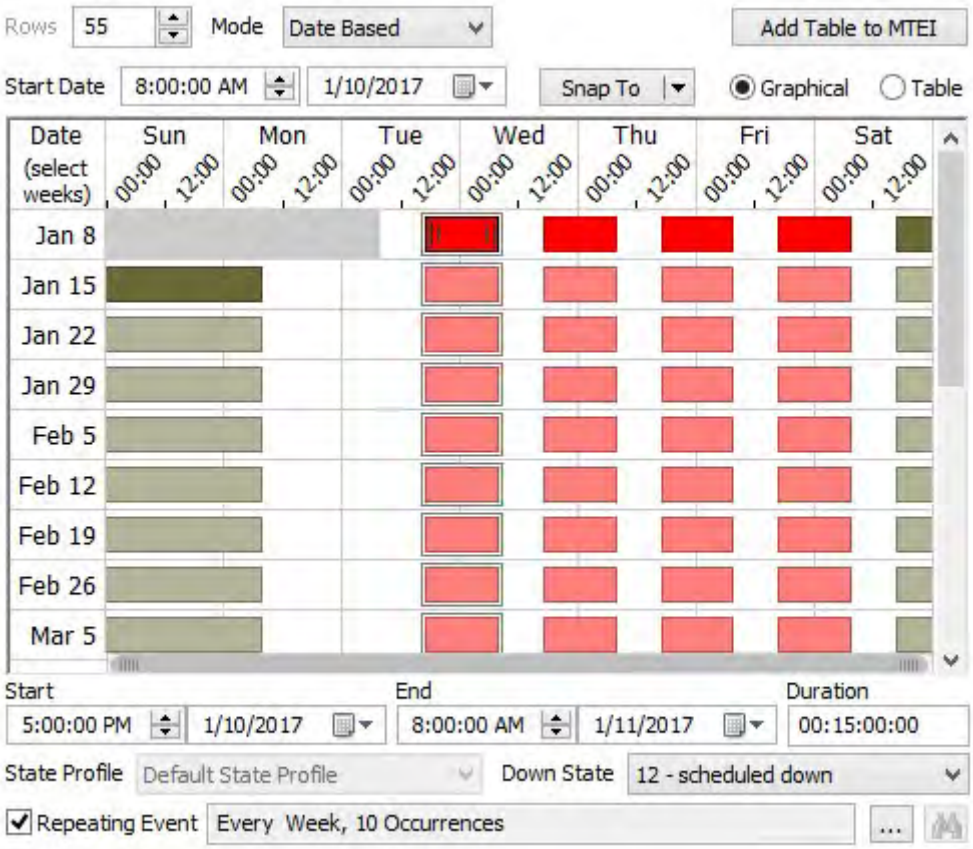
State Profile - Specifies the state profile for the down state.

Down State - Specifies what state the members should go to when they enter their *Down Time*. Make Operational Time - Select a set of cells from the table and press this button to make those times Operational (||||||).

Make Down Time - Select a set of cells from the table and press this button to make those times Down (empty cells).

Setting operational time: When the Time Table builds the numeric down time table based upon the graphical table, it looks at the text length of each cell. If the text length equals 0, the cell is referencing a down time. This allows you to put any string value into a cell to make that time an operational time.

Date Based



Add Table to MTEI - This button will add the table to the model's multiple table import accessed through the Excel Interface. The imported table takes the form of the numeric table (Time, State, Duration, Profile). Start Date - Specifies the start date and time of the Time Table. Events that occur before the start time will not fire, however, the data will remain. This allows you to temporarily turn off parts of your Time Table if needed.

Snap To - Specifies the time to snap entries to in the view. By default there is no snap to. You can specify the preset 10 min, 15 min, 30 min and 1 hour, or set your own custom snap to time. The custom time is in model time units.

Graphical / Table - Switch between the graphical view and the numeric table view.

Start - The start date and time of the selected entry.

End - The end date and time of the selected entry.

Duration - The duration of the selected entry. Of form DD:HH:MM:SS.

State Profile - Specifies the state profile for the down state of the selected entry.

Down State - Specifies what state the members should go to for the selected entry.

Repeating Event - If checked, the event will become a repeating event. Here you can define how often the repeat occurs and either how many times it repeats or until which date and time it repeats to. ... - Displays the properties of the repeating event.

- If the selected event is part of a repeating event but is not the parent event, this will center the parent event in the view and select it.

Date Based Editor View

Creating entries - Click and drag anywhere in the white space to create a new down time. Snaps to the Snap To time.

Modifying entries - Click and drag in the middle of an entry to change the start time of the entry. Drag left or right to change the time along the week, or drag up or down to change weeks. Click and drag at the left or right edge of an entry to modify the duration. Snaps to the Snap To time. Start and end times as well as duration can also be modified through the GUI below the editor view.

Selecting weeks - Click and drag along the left side of the view under the Date column to select a week of entries. Any entries with their start time in the selected week will be selected.

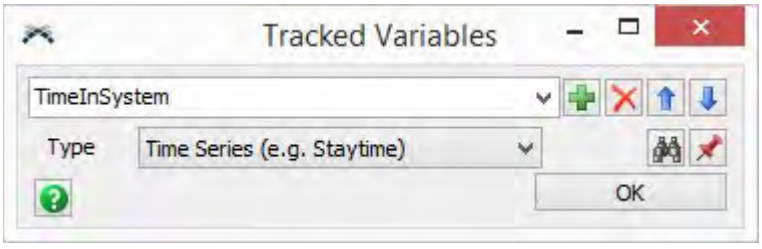
Keyboard shortcuts

- *Ctrl* + *C* - Copy selected entries.
- *Ctrl* + *X* - Cut selected entries.
- *Ctrl* + *V* - Paste selected entries. If a single entry was copied, the pasted entry will be pasted below the cursor position. If an entire week was selected, the entries are pasted based upon the week that the cursor is positioned over, keeping start times relative to the start of the week.
- *Ctrl* - Hold the control key and click and drag an entry to create a copy.
- *Arrow Keys* - Select an entry or a week of entries and use the up, down, left and right arrow keys to adjust the entry start times based upon the Snap To time.

Controlling the View

- Hold the Ctrl key while using the scroll wheel to zoom in on the mouse location.
- Hold the Shift key while using the scroll wheel to scroll left and right.
- Use the scroll wheel to scroll up and down in the view.
- Click and drag along the bottom scroll bar to zoom in/out and scroll left and right.

Tracked Variables



Tracked Variables are accessed from the Toolbox. (View menu > Toolbox > Tracked Variables).

Global Tracked Variables

Tracked Variables accessed through the toolbox are stored in the MODEL:/Tools/TrackedVariables folder. These tracked variables are global for the model. Work In Progress and Time In System are the two default Tracked Variables. When a Flowitem is created, an associated value is added to both of these default variables. These values remain in the model until the model is reset. Additional Tracked Variables may be added to your model through this window.

- Work In Progress - This is a count of all flowitems in the model at any given time. It is incremented when a flowitem is created and decremented when a flowitem is destroyed.
- Time In System - This records the time at which an object leaves the model and the total time it spent in the model.

Local Tracked Variables

Tracked Variables can be stored on other nodes in the model. These include, but are not limited to, Global Tables and object labels. You can also create Tracked Variables dynamically using FlexScript.

Properties

Tracked Variable Drop Down - Shows the current Tracked Variables. Enter text to rename the Tracked Variable.


- Adds a new Tracked Variable.


- Removes the selected Tracked Variable.


- Reorder's the selected Tracked Variable Up or Down in the list.


- Pins the tracked variable to a Dashboard as either a histogram or line graph.

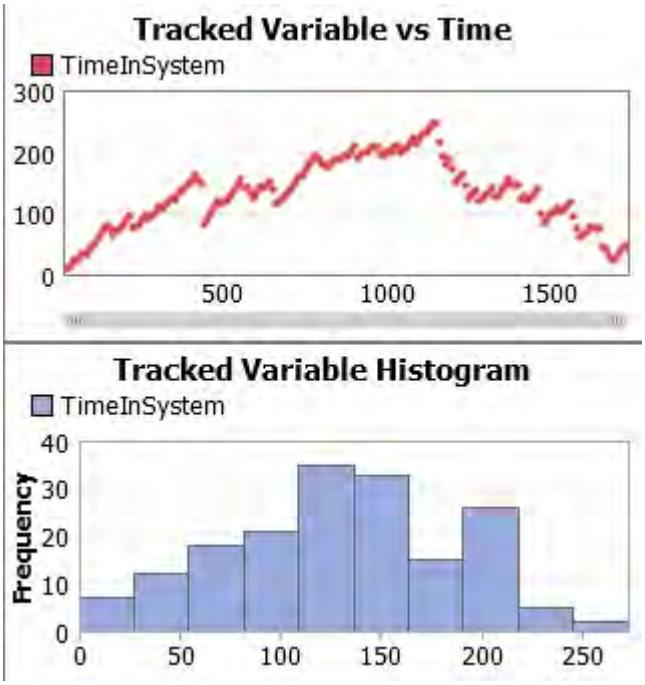
Type - The type affects what metadata will be collected and how it will be calculated. Type can be one of the following values:

- Level. Records the variable as a level that can go up and down, such as content. The average is a time-weighted average.
- Cumulative. Records the variable as a value that only accumulates over time, such as input or output. Average is not tracked.
- Time Series. Records the variable as a series of independent values. The average is a non-timeweighted average. An example of this type is staytime tracking.
- Categorical. Records the variable as a set of values, where values do not have a mathematical relationship to each other. Average is not tracked. An example of this is state tracking.

- Kinetic Level. This is like the Level type, except that the level can also have a rate of change, such as for a battery level that is constantly depleting/recharging at a defined amperage, or a liquid level that increases or decreases based on a flow rate.

Dashboard

Tracked Variables can be displayed in the Dashboard in multiple graphs:



Object Statistics using Tracked Variables

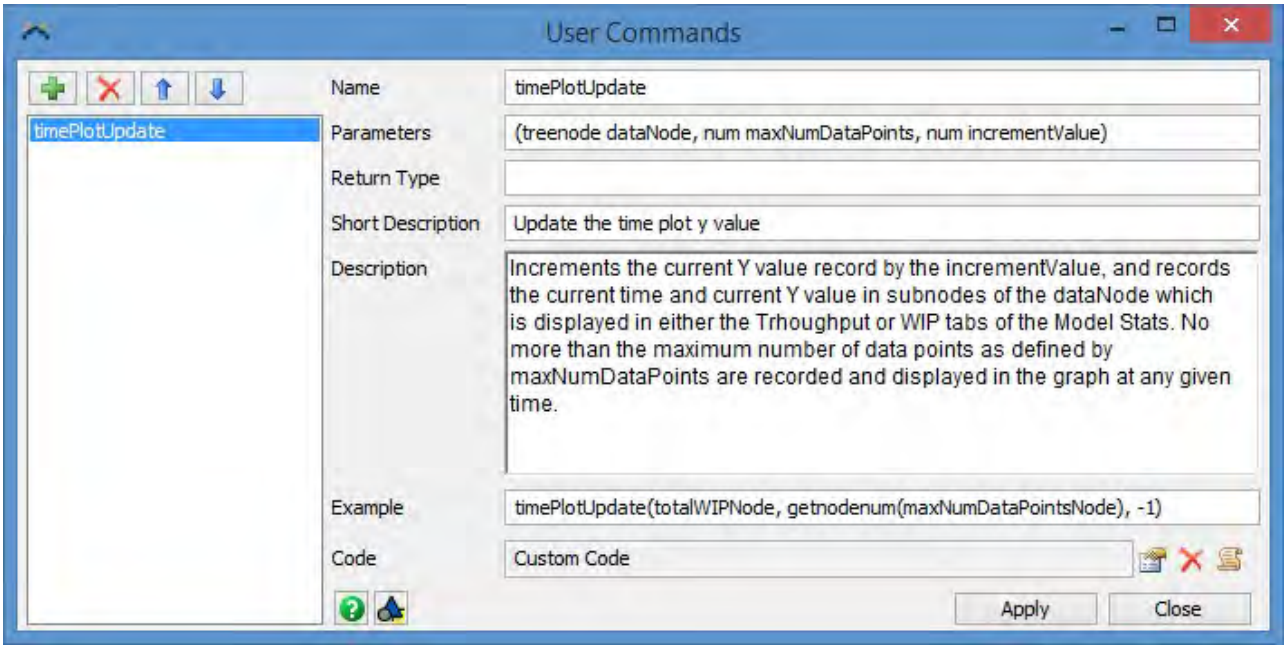
FlexSim's simulation objects also use tracked variables to record many of their own object-specific statistics. Each object will usually track the standard set of statistics, which are Content, Input, Output, State, and Staytime. Each object can also publish additional statistics depending on the type of object. To see the list of available statistics for an object, you can add a Tracked Variable vs Time chart to a dashboard, then use the button in its properties window to hover over the object of interest. This will list out the object's available statistics.

You can also manually get various statistic values of an object using the Object's `stats` property which will give you a reference to each Tracked Variable the object stores.

Listening for Object Statistic Changes

Using the process flow module's Wait For Event or Event-Triggered Source activities, you can also execute logic when an object's statistic changes. Use the button in the activity's properties pane and hover over the desired object to get a list of statistics that can be listened to. Statistic change events will have the format On<*StatisticName*>Change.

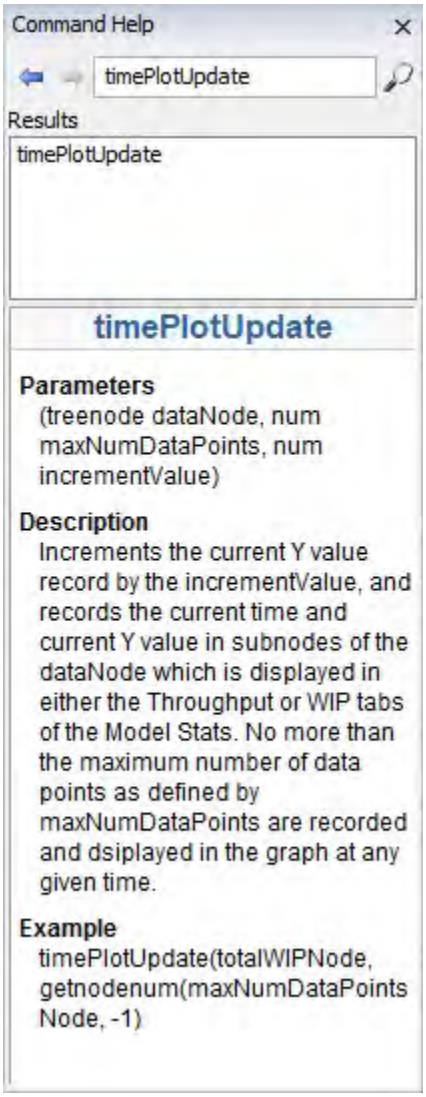
User Commands Concepts



User Commands are accessed from the Toolbox. (View menu > Toolbox > Add > Modeling Logic > User Command).

The User Commands tool lets you add, delete, and edit custom commands in your model.

Once you have created your command you can call it like any other FlexSim command. It will appear in blue when typing code and it will appear in the Command Summary. You can also hover the mouse over the command and press F1 to display the Command Helper.



Note on calling user commands from C++: When calling a user command from C++, all parameters passed will need to be converted to a number, or else you may get compiler errors. To convert a parameter to a number, use the tonum() command.

Parameters

The following are some of the valid parameters you can enter in the Parameters and Return Type fields:

Object

treenode

node Array

obj string

str num

array var

The following are valid parameters, but not return types (use num for the return type):

int double

The var parameter with accept any of the datatypes. To specify multiple datatypes but not include all datatypes you can concatenate parameters together using a forward slash: str/node

In the Parameters field you can have optional parameters by enclosing the additional parameters in square brackets at the end of the list:
(var object, num type [, int flags])

Accessing Parameters

If your command specifies any of the above data types as parameters that will be passed in to the command, you can access those parameters through the param command: param(num)

Where num is the rank from the parameter list. For example, if the parameters was specified as:
(treenode dataNode, num maxNumDataPoints, num incrementValue) You would

access those values in your code by:

treenode dataNode = param(1); double

maxNumDataPoints = param(2); int

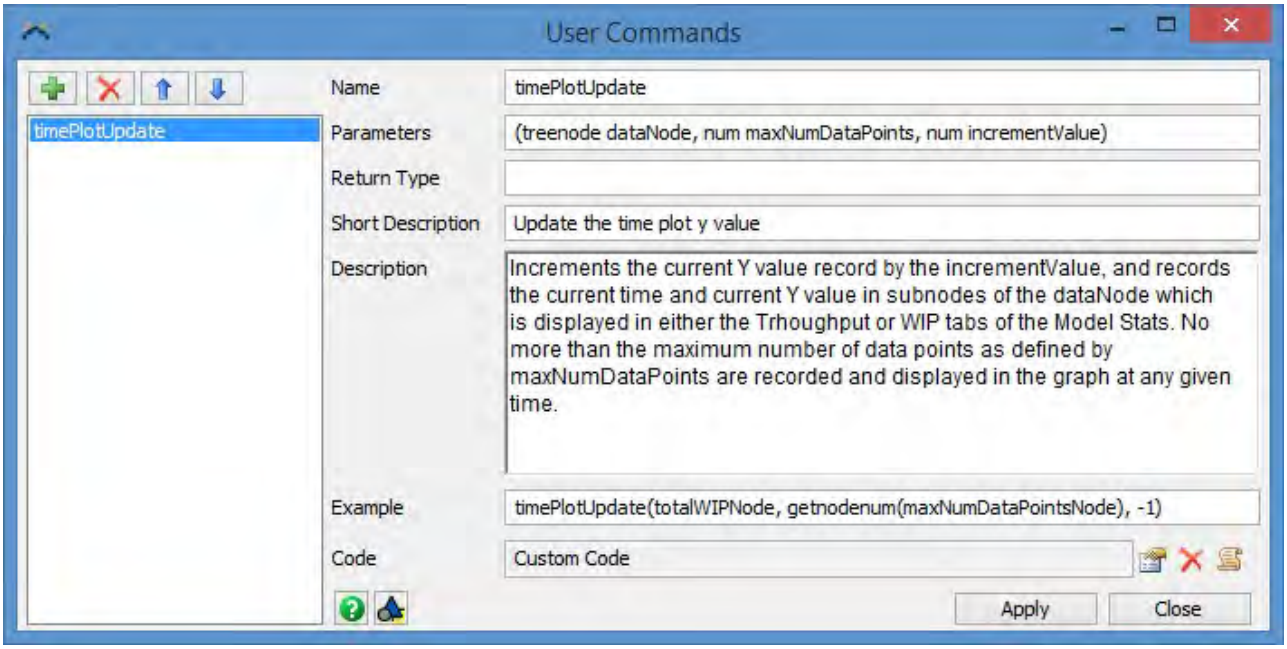
incrementValue = param(3);

Note on Parameters and Return Type: The values specified here will be used by the FlexScript parser. The number of parameters and their datatypes will give FlexScript warnings if not passed in as documented in the Parameters section here. If the return type is incorrecly used in code, it will return a FlexScript error and not build correctly. You can specify the parameters list as (...) and leave the return type blank for the FlexScript parser to ignore them. For examples of parameter lists and return types, view the Command Documentation.

Description

When you open the command documentation from the menu Help > Commands, the user commands will be shown in the command documentation in addition to all of the built-in FlexScript commands. The description field shown above will be displayed as html so you can use html markup for formatting, such as <b> and <\b> for bold. Because the description allows for html markup, some standard characters do not work properly, such as <, >, &, and new line characters. To display these types of characters, you can use special markup codes, such as ">" (without the quotes) for a > symbol and <br/> for a line break. If any of the commands' descriptions container invalid html markup, FlexSim may throw an exception when you try to open the Users Manual and your commands will not be properly documented.

User Commands Reference



- Adds a new blank User Command.

- Removes the selected User Command.

- Reorder's User Commands Up or Down in the list.

User Commands List - Displays a list of all User Commands. Select a command from the list to edit it.
Name - The name of the User Command. In writing code, this name is the name of the command, ie myUserCommand().

Parameters - Specifies the parameters that can be passed into the command. Each parameter should be separated by a comma. To access a parameter in code, use the param(paramNum)command.

Return Type - Specifies the value the command will return. This can be values like, num, treenode, string etc.

Short Description - Specifies the short description of the command. The short description displays in the tooltip when the command is typed or you hover the mouse over the command in a code edit window.

Description - The description displays in the Command Summary.

Example - The example displays in the Command Summary

Code - Here is where you specify the Flexscript, C++ or DLL code for the command.

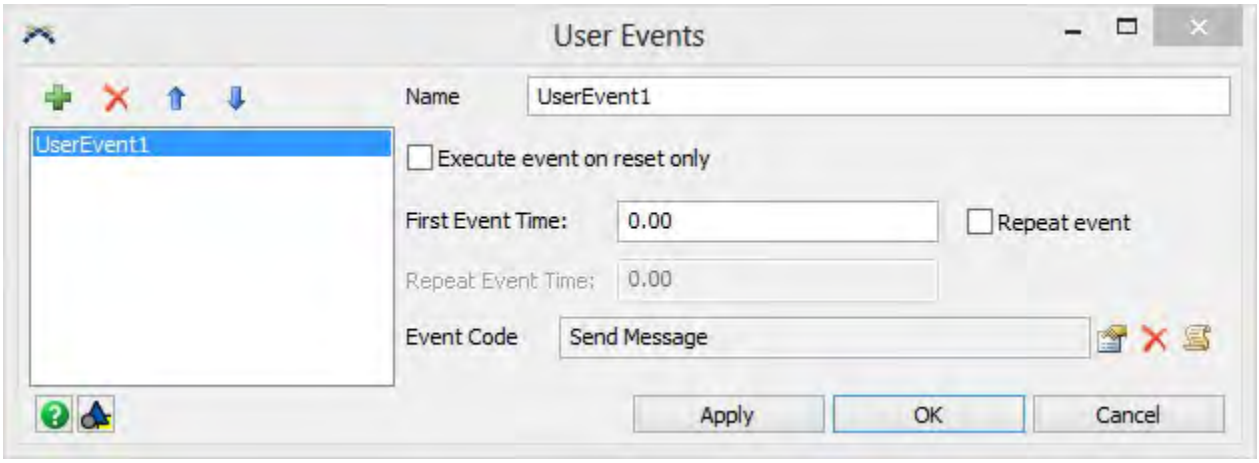
- Displays this help page.

- Adds the User Command or All User Commands to a User Library as either a Draggable Icon or an Auto-Install Component.

Apply - Saves all changes to the User Command.

Close - Closes the User Command window without saving changes.

User Events



User Events are accessed from the Toolbox. (View menu > Toolbox > Add > Modeling Logic > User Event).

User Events are FlexScript functions that execute at set times during the model's run, but are not connected with any specific, visible object. They are created and stored in model's */Tools/UserEvents* folder. A model can have any number of user events.

- Adds a new blank User Event.

- Removes the selected User Event.

- Reorder's User Events Up or Down in the list.

User Events List - Displays all of the Model's User Events. Changing from one User Event to another will apply any changes made to the previous User Event before displaying the newly selected User Event.

Name - This is the name of the user event. This is purely for the modeler's convenience and has no affect on the model. It is helpful to 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 in model units that the User Event will occur.

Repeat Event - If this box is checked, as soon as the user event executes, it begins counting towards the next execution time as defined by the Repeat Event Time.

Repeat Event Time - If the Repeat Event box is checked, this field will be enabled. Once the first User Event executes, the User Event will repeat in regular intervals defined by this time (in model units). Event Code - This is where the FlexScript code for the event is written. Any valid FlexScript statements can be used in this picklist.

- Adds the User Event to a User Library as either a Draggable Icon or an Auto-Install Component.

Apply - Saves all changes to the User Event.

OK - Saves all changes to the User Event and closes the window.

Cancel - Cancels any changes made to the User Event and closes the window.

Video Recorder Concepts

The Video Recorder can be accessed from the Toolbox. (View menu > Toolbox > Add > Visual > Video Recorder).

The Video Recorder is an object that is added to the Tools folder of the model tree when the Video Recorder option is selected in the Tools menu. The Video Recorder is based on the FFmpeg video solution, and enables the user to record high quality videos of their model, utilizing camera flypaths, layers, and animations if desired. This functionality is obtained by use of output files, sequences, and layers. Windows Display Scaling and FFmpeg: The video recorder uses FFmpeg. In order for FFmpeg to properly record videos, Windows display scaling level (DPI) must be set to 100%. If your computer is not at 100%, you can change the setting through the Windows control panel under the Display settings..

Output Files

The Video Recorder allows the user to record multiple output files at one time. Each output file can have a different size, framerate, codec, and file type. This can be especially useful for long recordings with complicated models, where it can take hours to record, since you can record all of the views you want in one run. By default, output files are encoded using a variant on MPEG-4 and written to the .mp4 file format.

Sequences

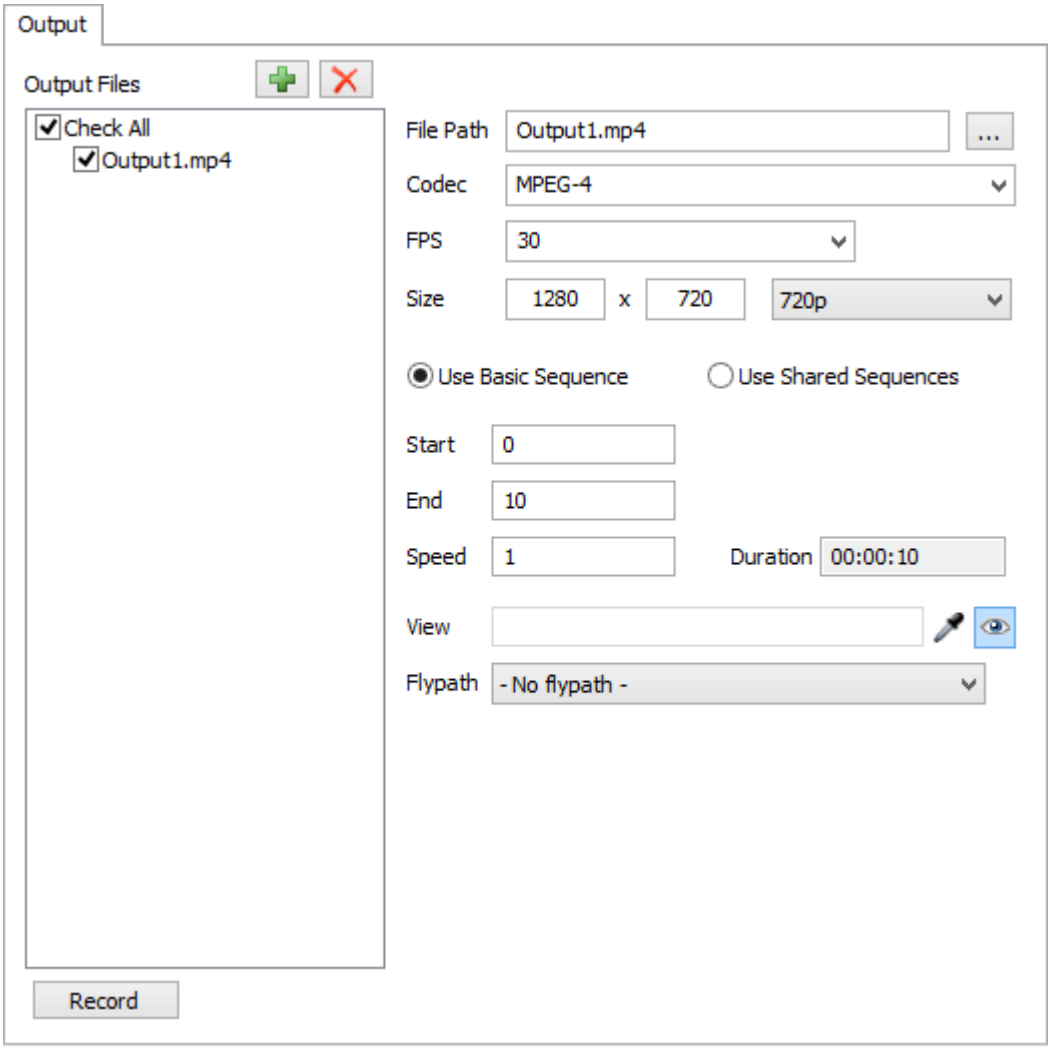
Each output is made up of sequences. Sequences are what defines when to start and stop recording during your model run. A single output can have as many different sequences as you want, so you can capture only the relevant points of your simulation. However, only one sequence is required to make one file, and by default only one sequence is used per output. If you want more than one sequence in a single output, select the Use Shared Sequences option.

Layers

Layers allow you to record multiple views of a model in a single video. You can customize the size and position of each layer to arrange the views of your model. By default, there is only one layer, and it is sized so that it matches the output size.

Output

The following image shows the Output tab of the Video Recorder dialog:



The following sections explain the properties found on this tab:

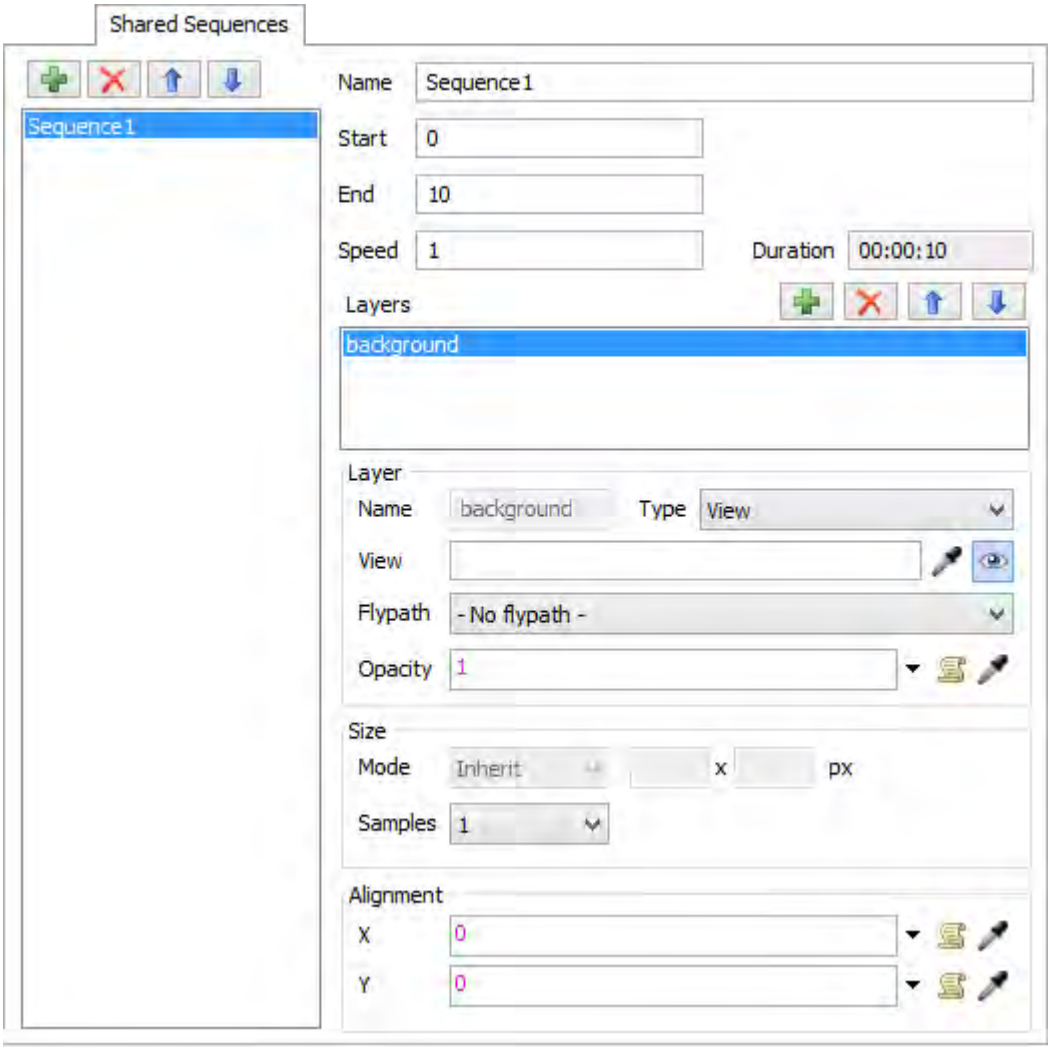
- Output FilesThis view lists the names of the output files that will be created when recording. To add another output file, click the button. To remove an output file, select the desired output in the list and press the button. When an output is selected, details about that output are displayed in the right portion of the tab. If the box next to the output name is cleared, that file will not be produced during the next recording.
- Record Click this button to create the output files. The model will reset and run, allowing the Video Recorder to create the output files.
- File Path This specifies where the output file will be saved. The path can be set manually or by clicking the browse button. If you want to use a video container other than .mp4, you can specify the extension here. You can specify any container supported by FFmpeg.
- Codec This is the codec the video will record in. MPEG-4 is the default codec, and will record in high quality. H.264 is a widely accepted codec for recording high quality video in low file sizes. If you want to use a different codec, you can enter in the standard FFmpeg command line syntax for codec options here.

  Note: Windows Media Player normally does not support the playback of video files encoded in the H.264 format, but virtually any other media player (such as the VLC Player) does. There are also extensions to Windows Media Player that allow H.264 to be played that can be downloaded and installed. If you want to guarantee that your video can be played in Windows Media Player, record your video as an MPEG-4..

- FPS This field indicates how many frames per second will be recorded. You can save rendering time and output file size by lowering this number. However, values below 30 will cause your videos to look choppy.
- Size This is the size of the video, in pixels. Use the dropdown to select one of the presets, which are standard video sizes. You can also enter custom values.
- Use Basic Sequence Use this option to create simple videos. When you use this option, the following properties are available:
    - Start The start time (in model time) of the video o End The stop time (in model time) of the video o Speed The model speed o Duration The total length of the output video. This is calculated from the previous three properties.
    - View The view used to record the video. Please note that only some views may be recorded by the Video Recorder.
    - Enable/Disable Preview When selected, you will see your model running as the video is recorded. Disable this option to improve recording time.
    - Flypath The flypath to use. None by default.
- Use Shared Sequences This option allows you to create more complex videos, including videos with multiple layers. When you choose this option, you can add and remove sequences to the selected output file. You can create and edit sequences on the Shared Sequences tab.

Shared Sequences

The following image shows the Shared Sequences tab of the Video Recorder dialog.

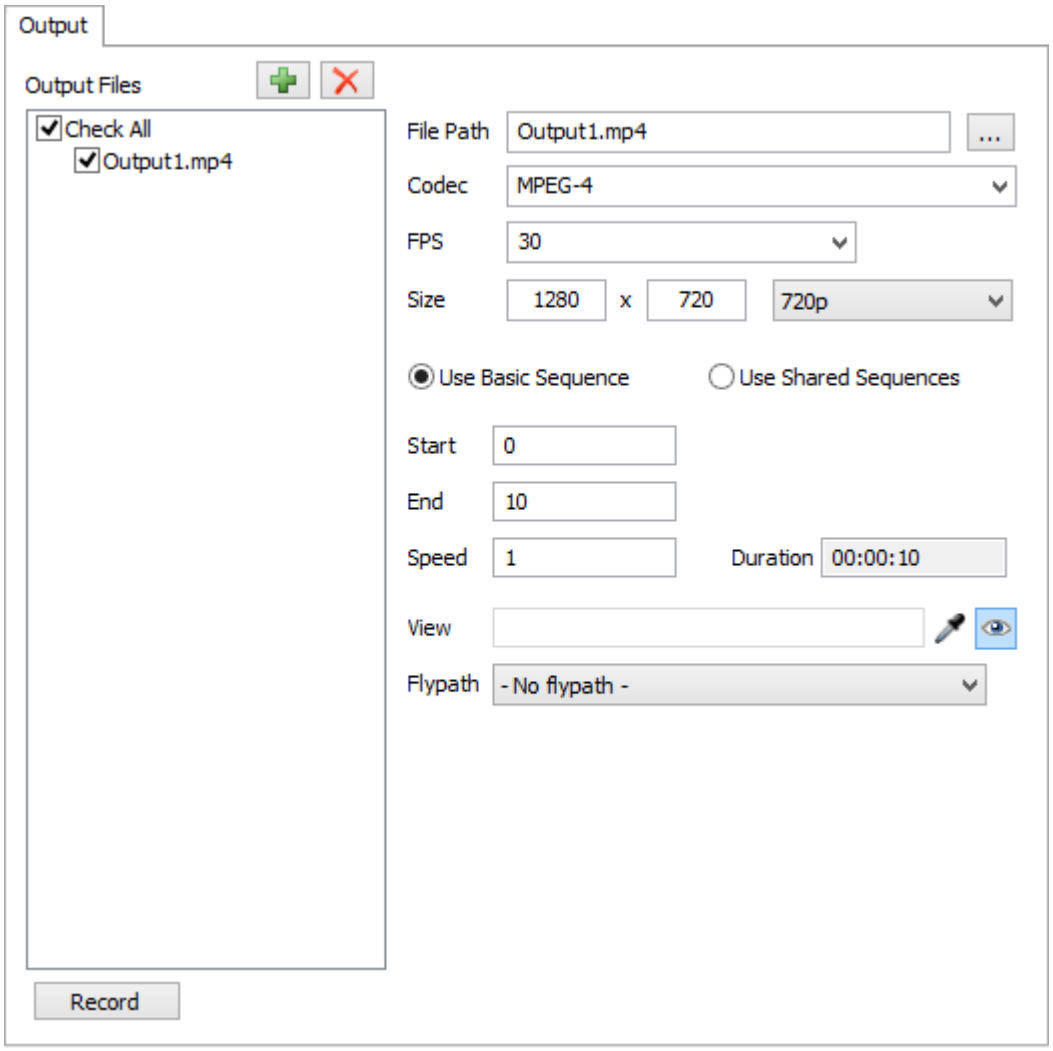


The following section describes the properties available on this tab:

- Sequences List This list shows all the sequences that are available for you to use in a video. You can add, remove or reorder the sequences in this list. The following properties apply to the sequences as a whole:
    - Name The name of the sequence o Start The start time (in model units) for the sequence o End The end time (in model units) for the sequence o Speed The model speed o Duration The total video length. This value is calculated from the previous three properties. As you create sequences, they become available to use on the Output tab.
- Layers This list shows you all the layers that are part of the selected sequence. You can add, remove, or reorder the layers in this list. Layers are drawn so that the each layer is drawn over the layer below it in the list. Thus the layer on the bottom of the list is the background that all other layers are drawn on top of. The following properties apply to the selected layer:
    - Name The name of the layer
    - Type The type of layer. View implies that this layer will record a view of your model. Color means that this layer will display a solid color. You can use the second option to set the color of unused space in your video.
    - View The view used to record the video. Please note that only some views may be recorded by the Video Recorder.

- Flypath The flypath used by the current layer. Keep in mind that due to how flypaths operate, trying to have multiple layers use different flypaths on the same view will not work.
- Opacity The opacity of the current layer. You can choose the Static Value option to specify a fixed opacity. You can also use these fields to animate the opacity. Use the Keyframe Animation option to choose times and values for the layer, and choose Slide In/Slide Out to have the layer transition towards that value. Finally, you can enter in your own custom FlexScript to define the opacity of the layer.
- Mode Set this option to Inherit to have this layer fill the output size. Use Custom to specify a custom size for this layer.
- Samples If this option is set higher than one, then your video will have high quality anti-aliasing. For example, if you are recording a 1920x1080 layer and you set this value to 2, the Video Recorder will capture a 3846x2160 layer, and use it to smooth out your image. This is called supersampling, and will significantly increase recording time.
- Alignment The X and Y value specify the offset for this layer from the bottom left of the output area. You can choose the Static Position option to specify an offset with respect to other locations. You can also use these fields to animate your layer. Use the Keyframe Animation option to choose times and locations for the layer, and choose Slide In/Slide Out to have the layer transition into the output area. Finally, you can enter in your own custom FlexScript to define the position and motion of the layer.

Video Recorder Example



Recording a simple video

The Video Recorder is designed to make recording simple videos of your model as easy as possible. For example:

1. Open the Video Recorder window (add an Video Recorder object to your model) by going to the Toolbox > Visual > Video Recorder.
2. Specify your output file path by clicking the browse button.
3. Click the sampler button next to View and click on your model.
4. Your video is now ready to record. If desired, set the size, FPS, codec, start and end times, speed, and fly path options to fit your needs.
5. Press the Record button.
6. Your model will reset and run, and you will see a popup in the top left corner displaying recording progress.
7. When finished, the popup will disappear and your video file will appear at your specified path.

A more complex video

In this example we will utilize Shared Sequences and some of the advanced layer options.

1. Open the Video Recorder window and specify the output file path as above.
2. Click the Shared Sequences tab.
3. There should be one Sequence available to edit. Set the End field to 30 and the Speed to 2.
4. The background of this video will be a solid color. Change the Type to Color and pick a dark blue.
5. Add another layer by clicking the button above the Layers list.
6. Make sure the Type of the new layer is View. Set the View to your model view by using the sampler button as above.
7. Change the mode in Size to Custom, and set the length and width to be 20 pixels less than your output length and width.
8. From the picklist in the X field, choose Slide In/Out. Change the first popup field to be 10, and have it slide in for 10 seconds. This will make it so that the model view will start off the screen, slide in for 10 seconds, and stop when the left edge of the model view is 10 pixels from the left edge of the video.
9. Set the Y field to be 10.
10. Switch back to the Output tab. Change the mode from Use Basic Sequence to Use Shared Sequences.
11. Using the button, select your sequence from the drop down list.
12. Press the Record button.

The recorded video should display the model view sliding in from the left, stopping centered in the blue background.

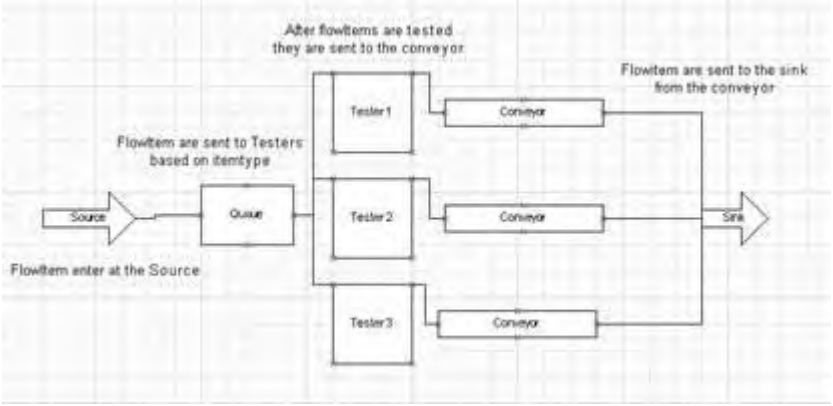
Microsoft Visio™ Importer

The Visio Importer can be accessed from the Toolbox. (View menu > Toolbox > Add > Import > Visio Import).

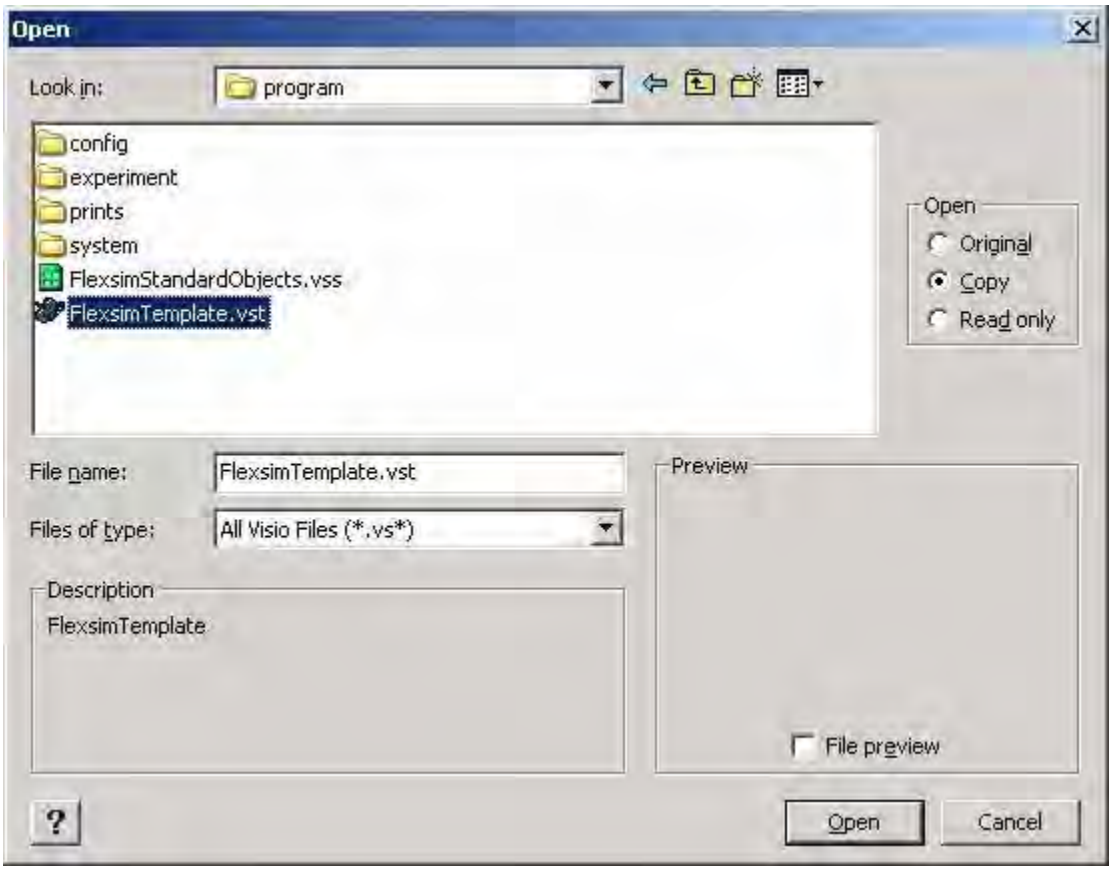
The Visio import tool allows you to build models in Microsoft Visio™ and import them into FlexSim.

Tutorial

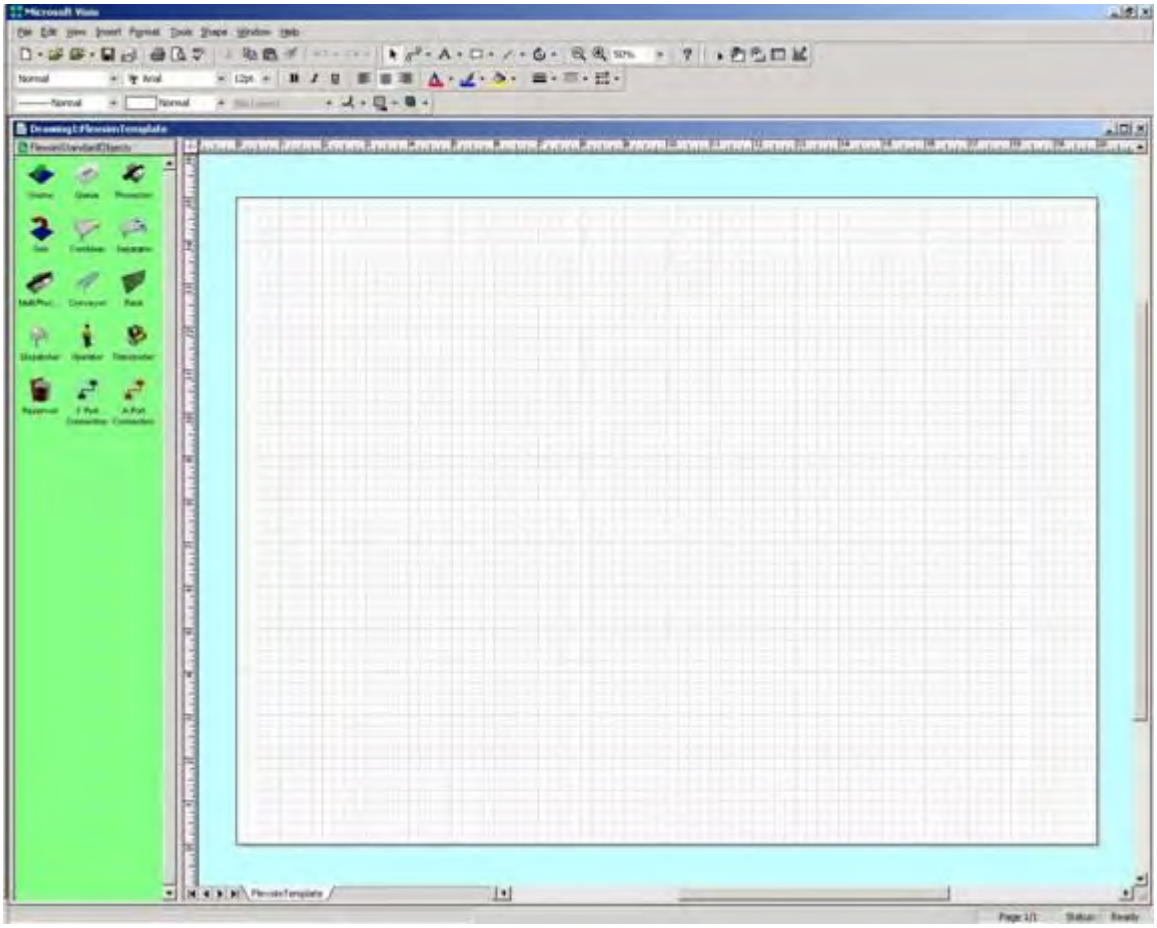
Start Visio



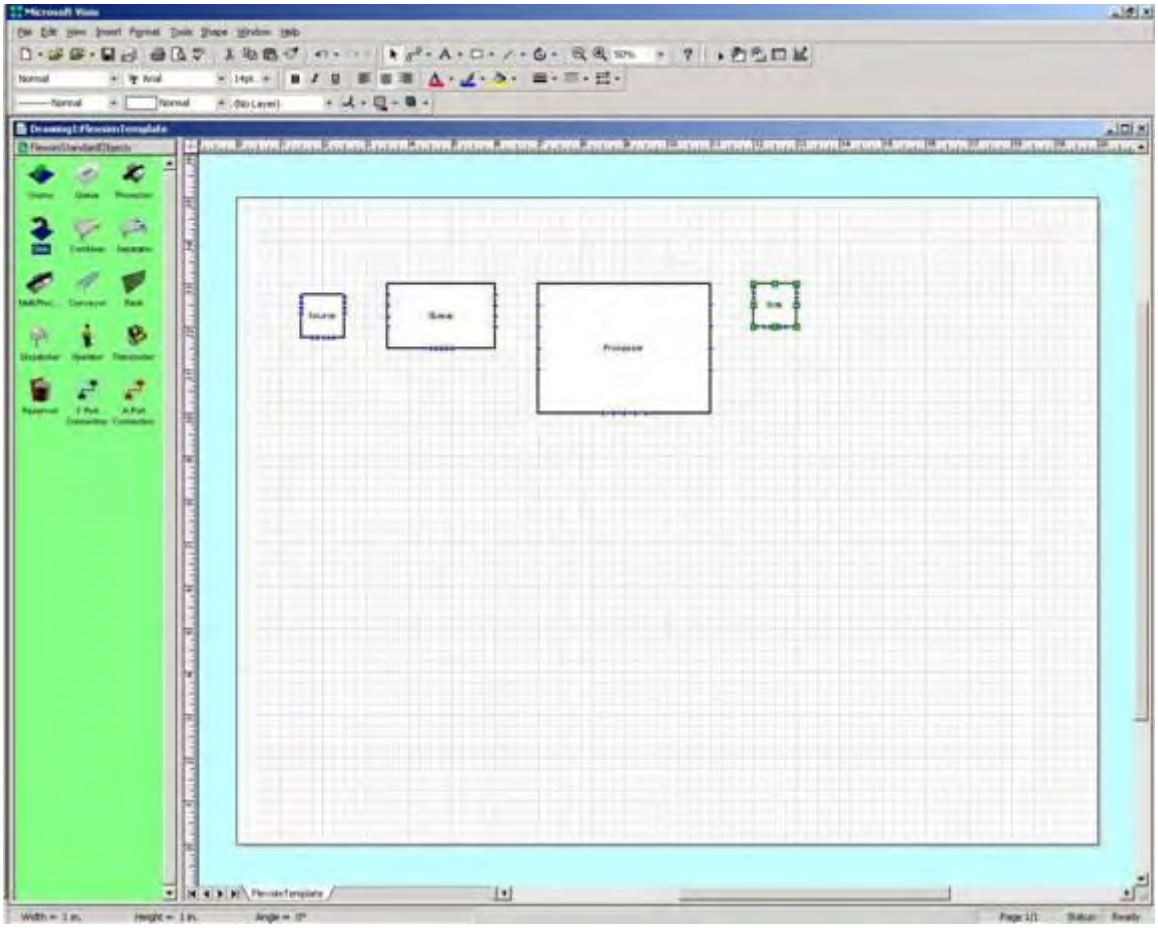
Open the FlexsimTemplate.vst



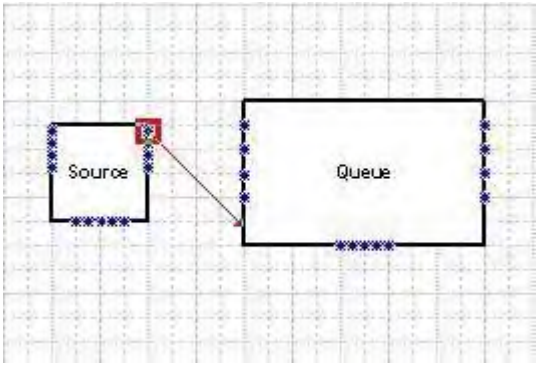
Visio will look like this.



Drag objects from the Stencil into the model space to arrange their layout.

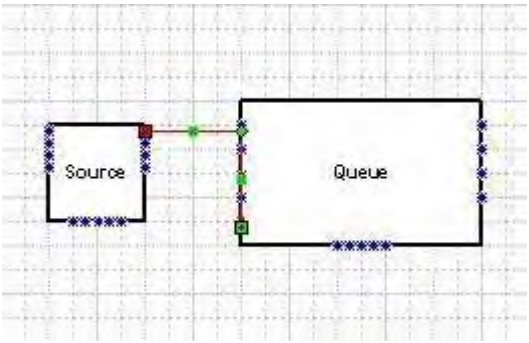


Drag the A Port Connection object into the model to create a connection.

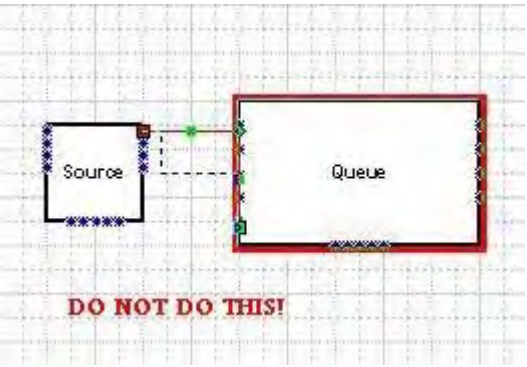


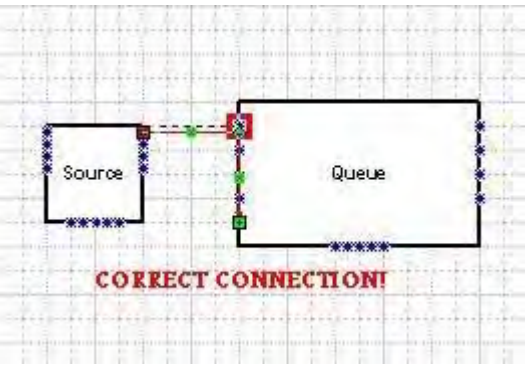
The connection is made from the upper left point to the bottom right point. If you hook the connection backwards, the connection will be backwards in FlexSim.

This is a connection that is only connected to the upstream object.



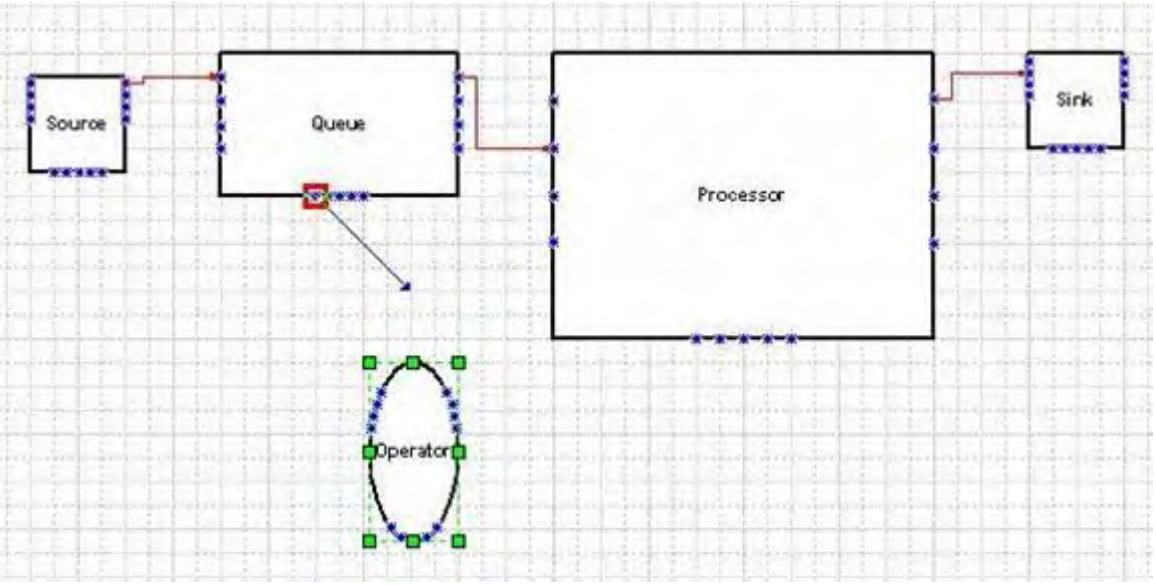
Note: Do NOT connect the connection directly to the object. It must be connected to one of the blue points.

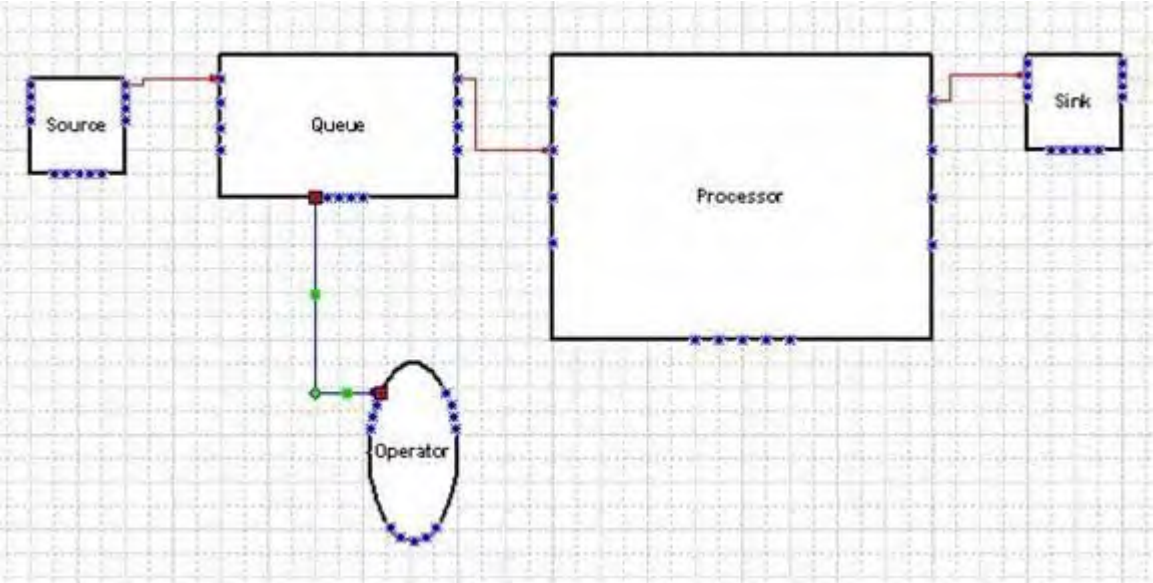




When dragging out a connection, it is acceptable to have both ends touching blue connection points. The connection will be created correctly. Also, the connection point that you connect them to in Visio does not matter. FlexSim connections will be ordered according to the order in which they were dragged 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.

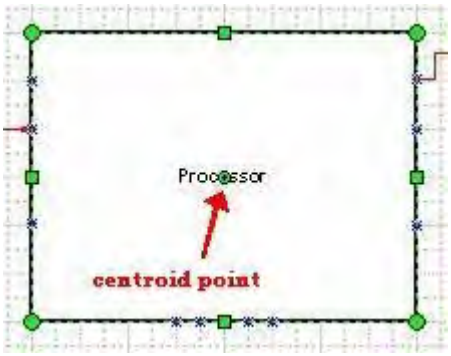




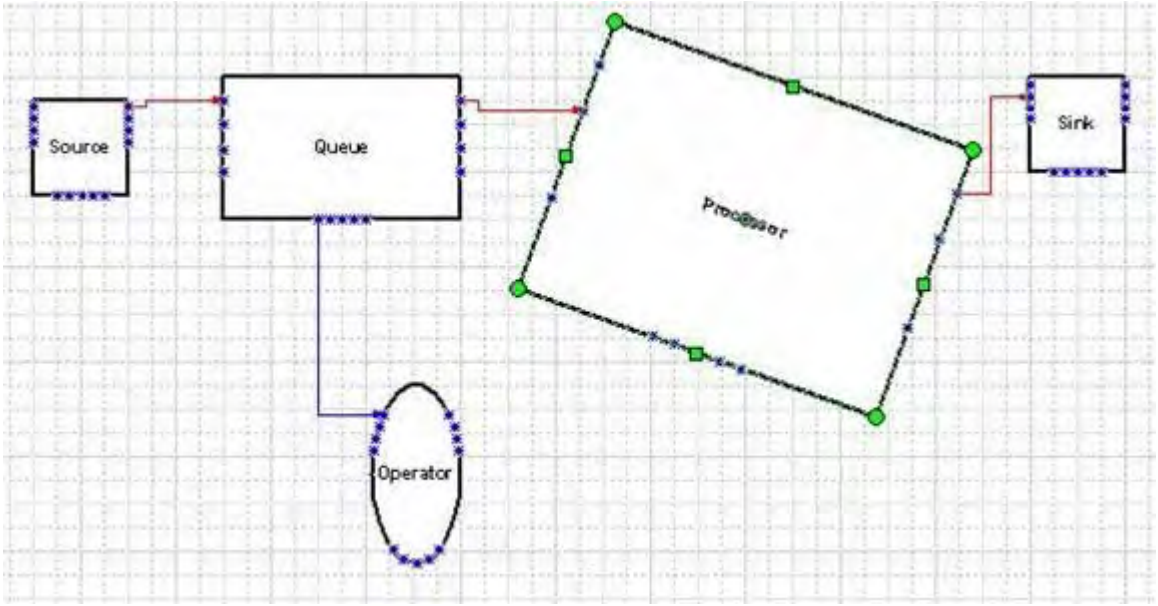
Objects can be rotated and resized from Visio to resize and rotate objects in FlexSim.

The rotation tool in Visio looks like this. 

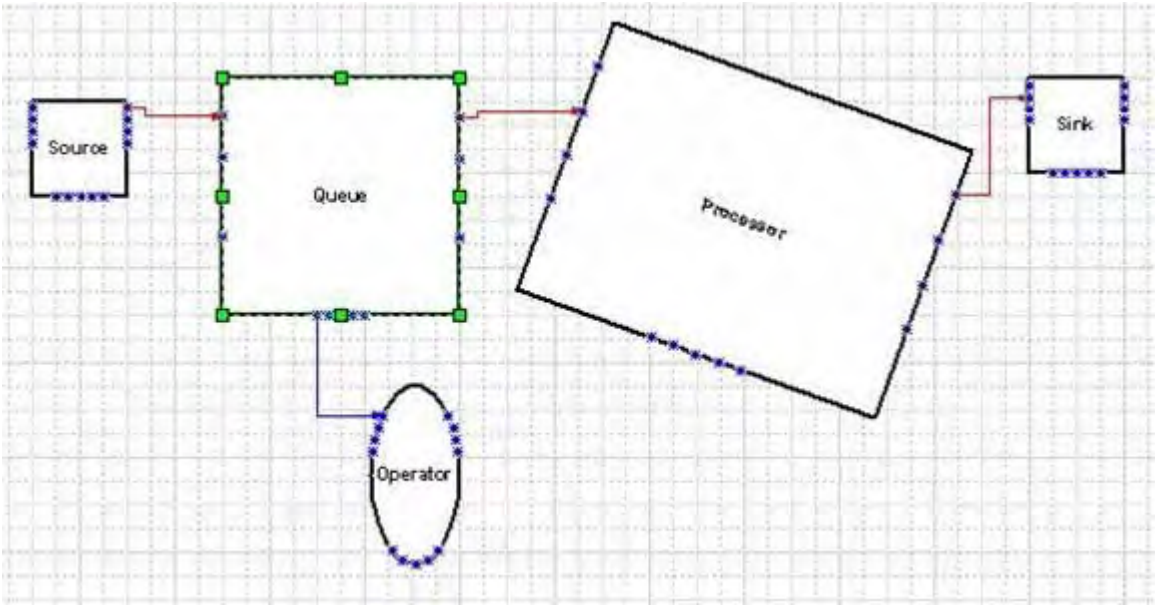
Do not move the centroid point of the object when resizing. The centroid point is the green circle with a black point in its middle. It is used to define where the center of rotation is for that object. If you move it in Visio, the object will not be located correctly in FlexSim.



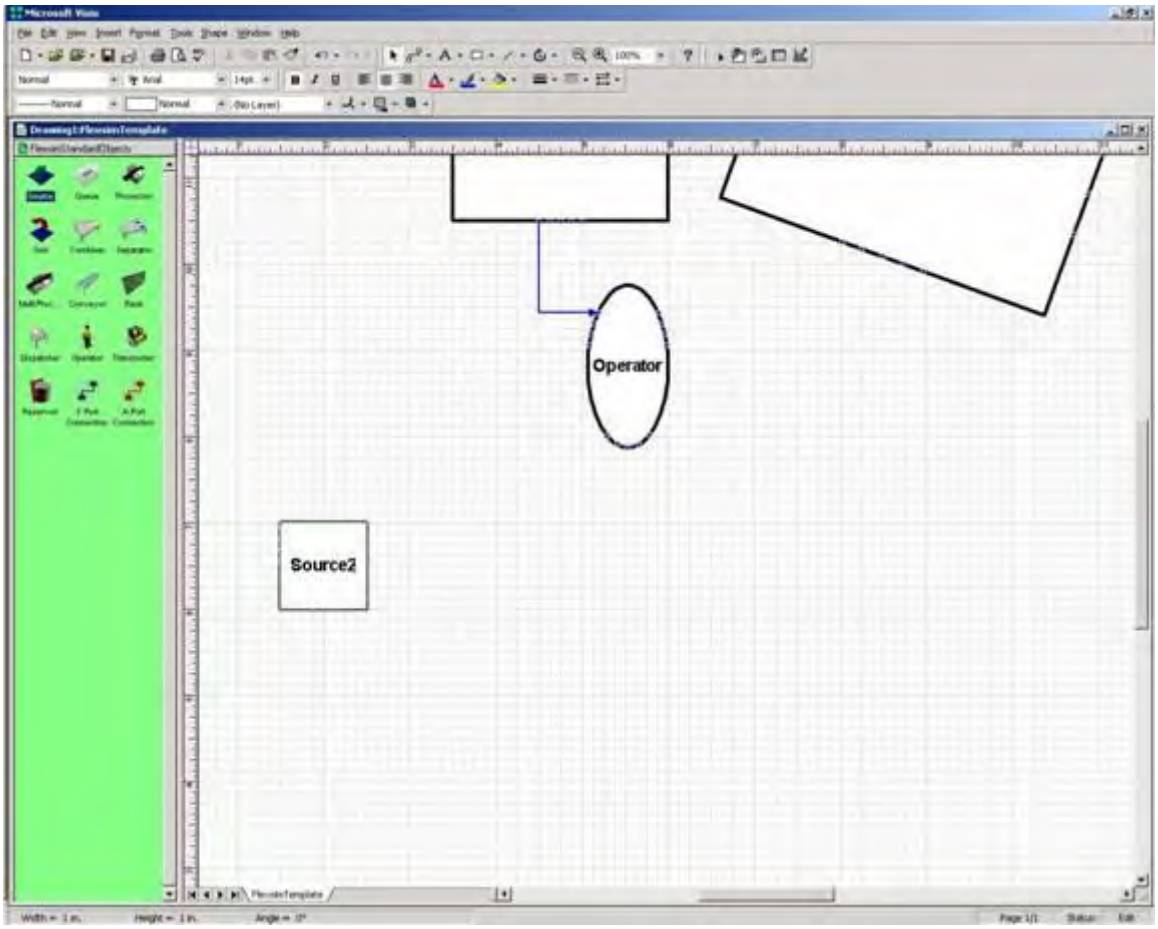
This processor was rotated by clicking a nd dragging the rotation tool around one of its corner points.



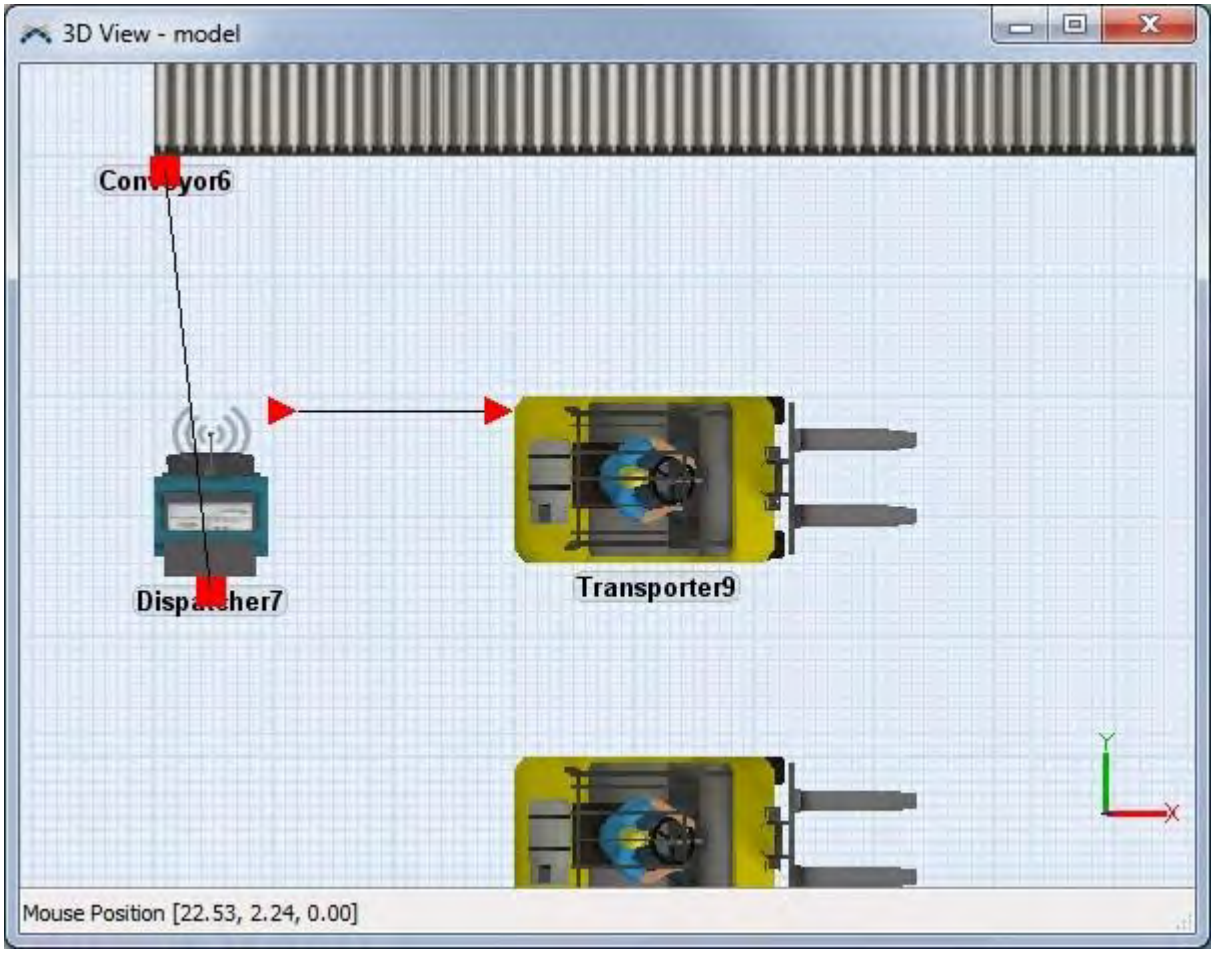
This queue has been resized by dragging its side point down using the selection tool .



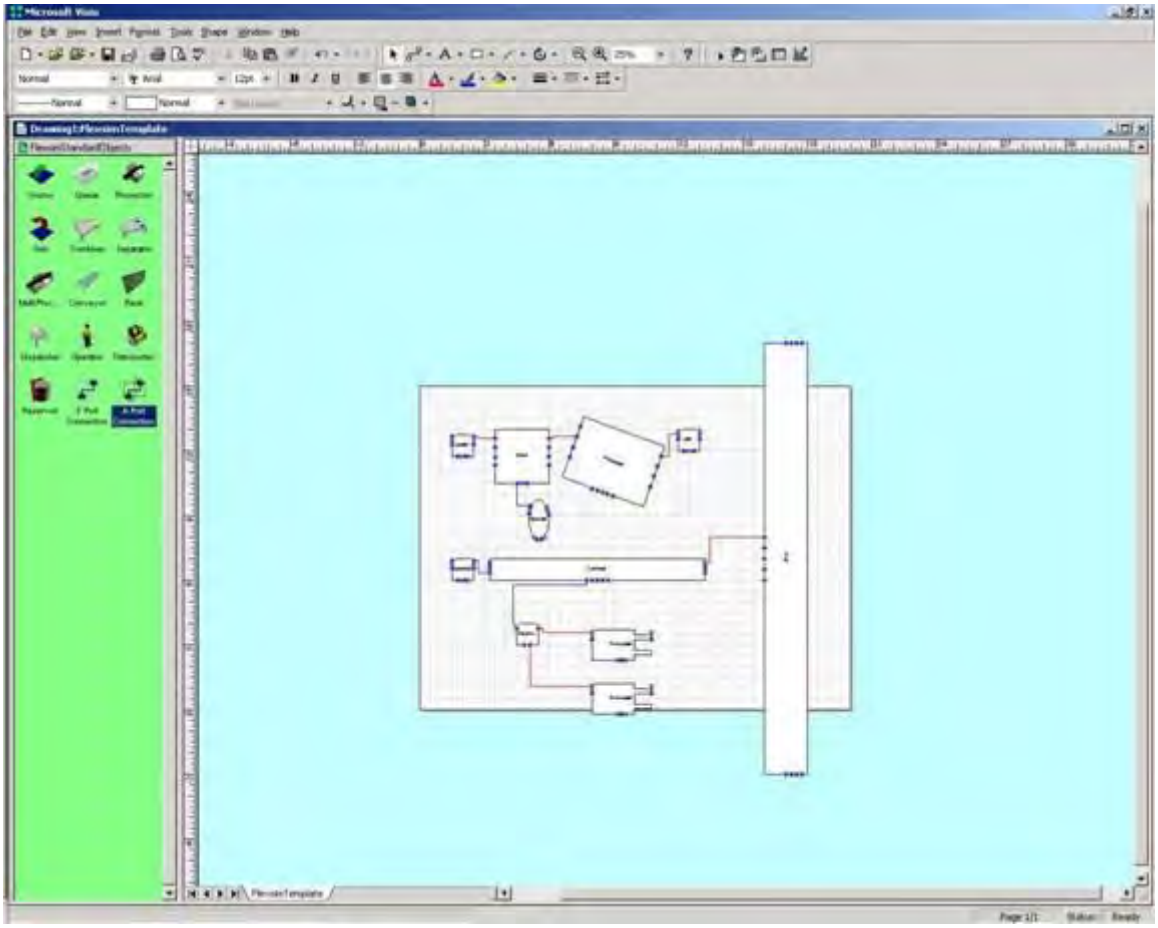
Objects can be renamed by double clicking on the object and typing in a new name.



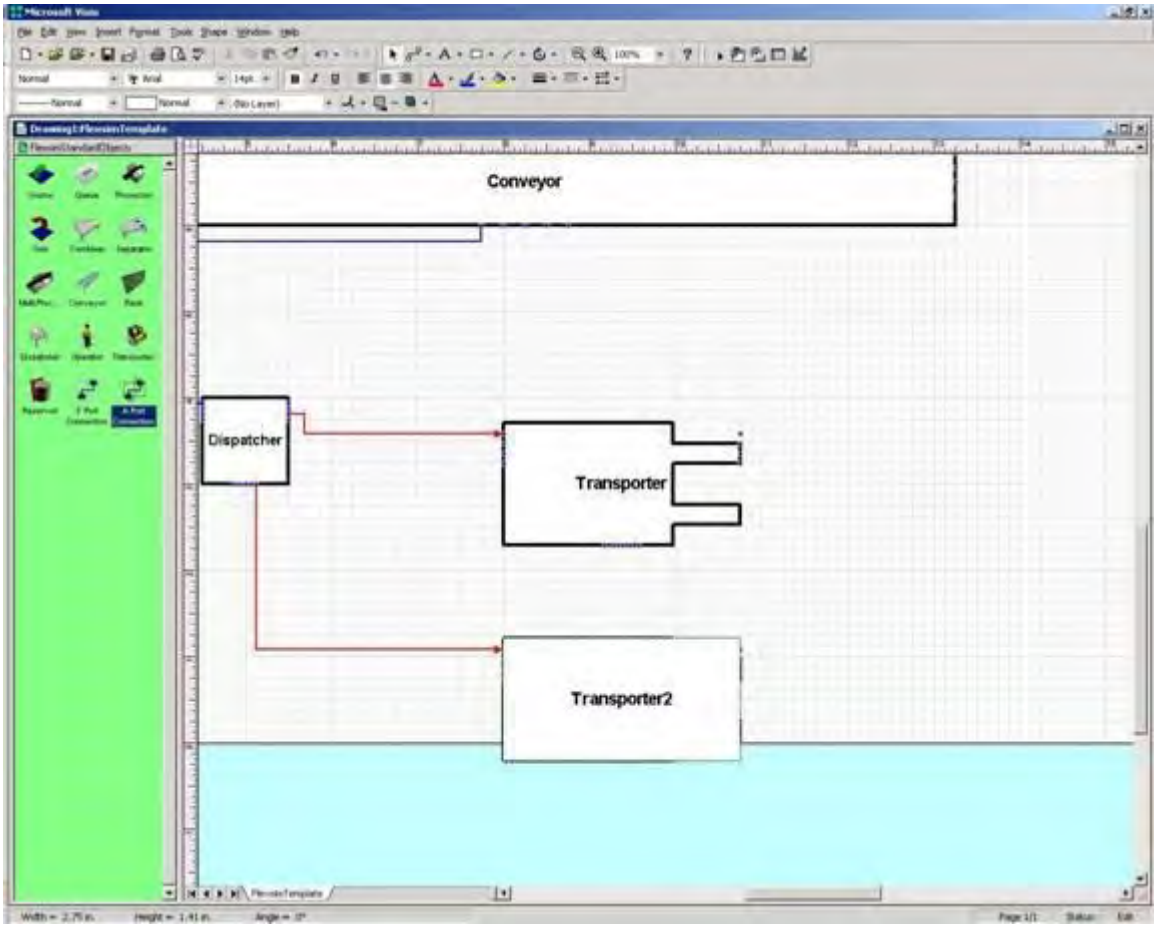
Each object in your model must have a unique name in order for it to be correctly imported into FlexSim. If objects are given the same name in Visio, their connections will not be created correctly in FlexSim. In the example below, the Transporters have the same name, so the Dispatcher was connected to the first one twice.



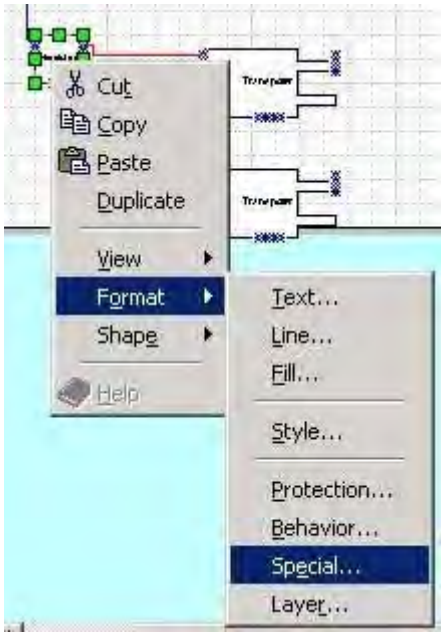
Here is a nearly completed Visio page with a FlexSim layout.

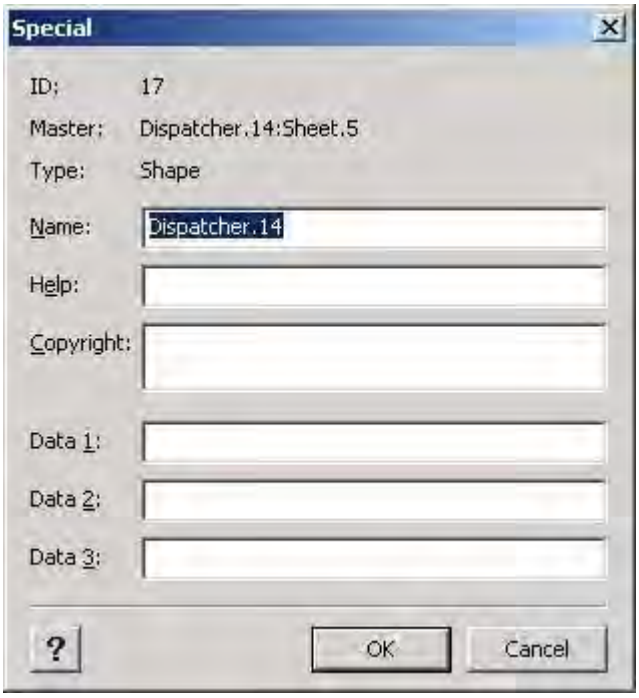


There are a few more things to check to be sure are correct before exporting it to Excel to import into FlexSim. Be sure each object has a unique name.



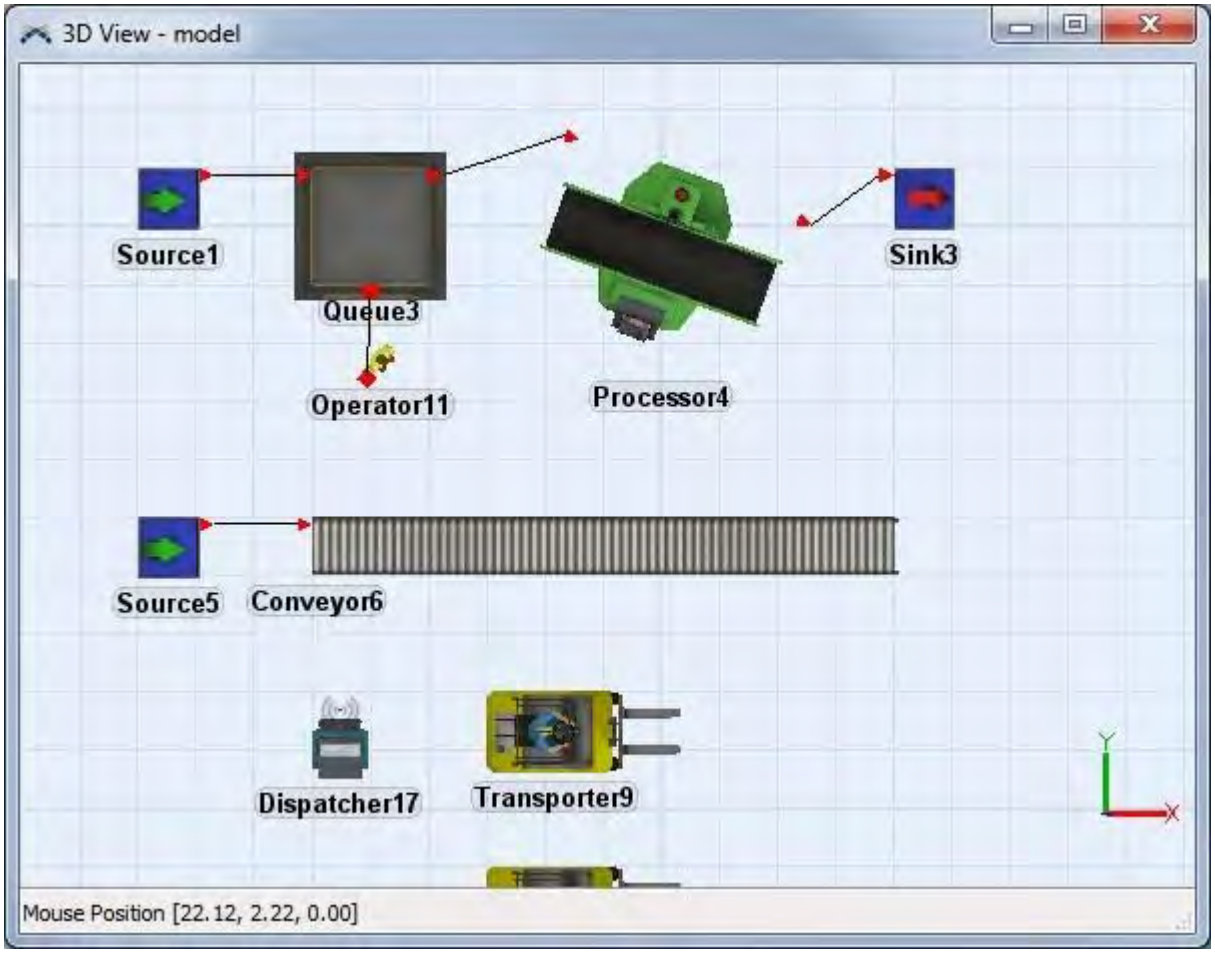
and each object must be correctly formatted in this menu.



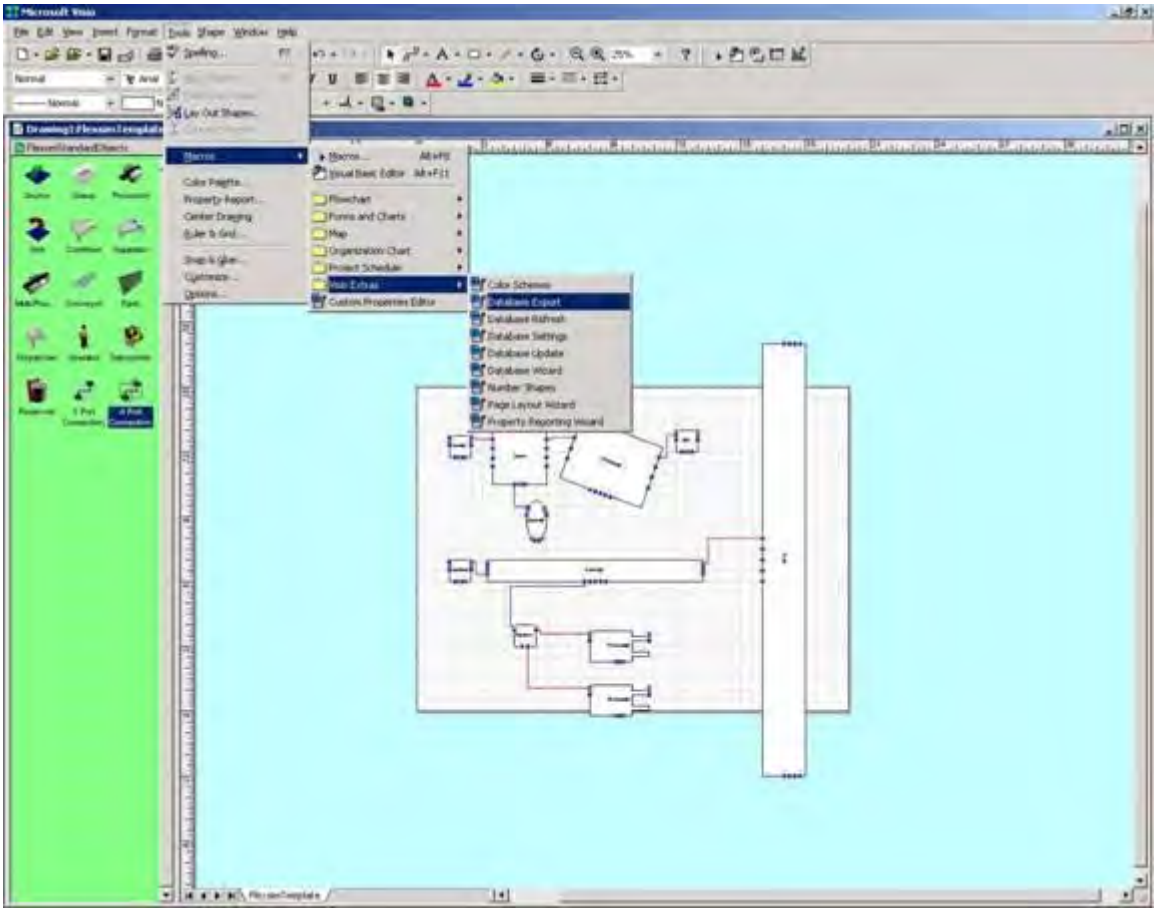


In Visio 2010 this dialog box has been renamed “Shape Name” as seen in the picture below

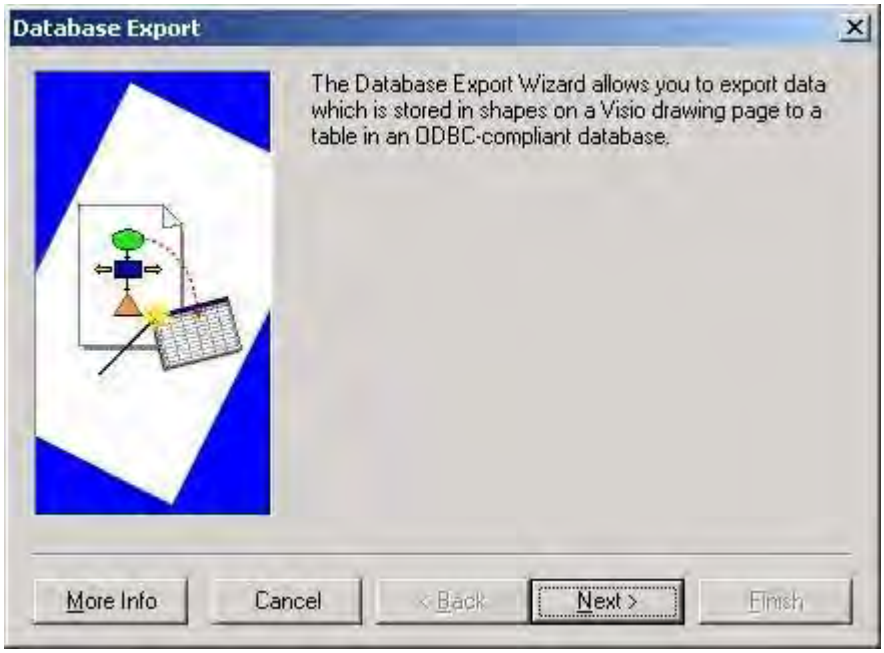
The Name field contains the object’s class and a number separated by a period or just the object’s class. If there is a number, it must match the ID number or the object will not be correctly exported. Typically the objects do this naturally, but sometimes Visio will incorrectly name an object in this 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”.



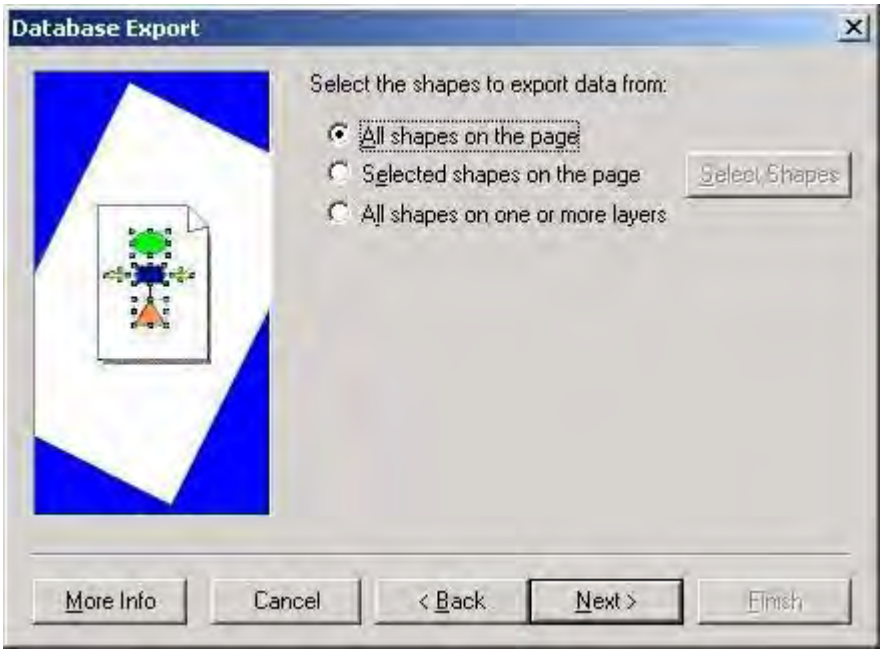
Now that everything has a unique name, and is correctly named in the Special Format, we are ready to export. Go to the Tools > Macros > Visio Extras > Database Export to export the file.



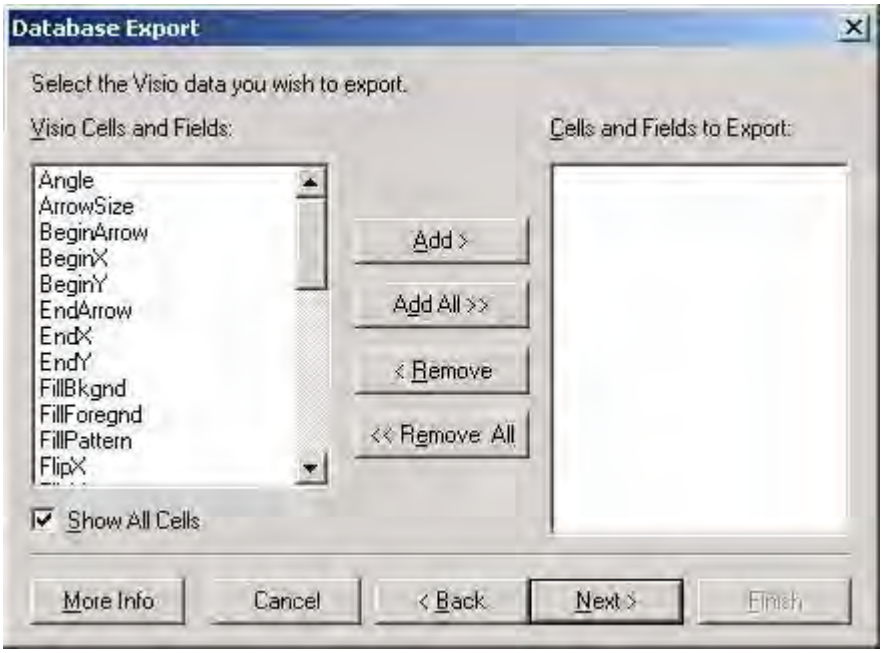
This screen will appear. Click next.



Select “All shapes on the page” and click Next.

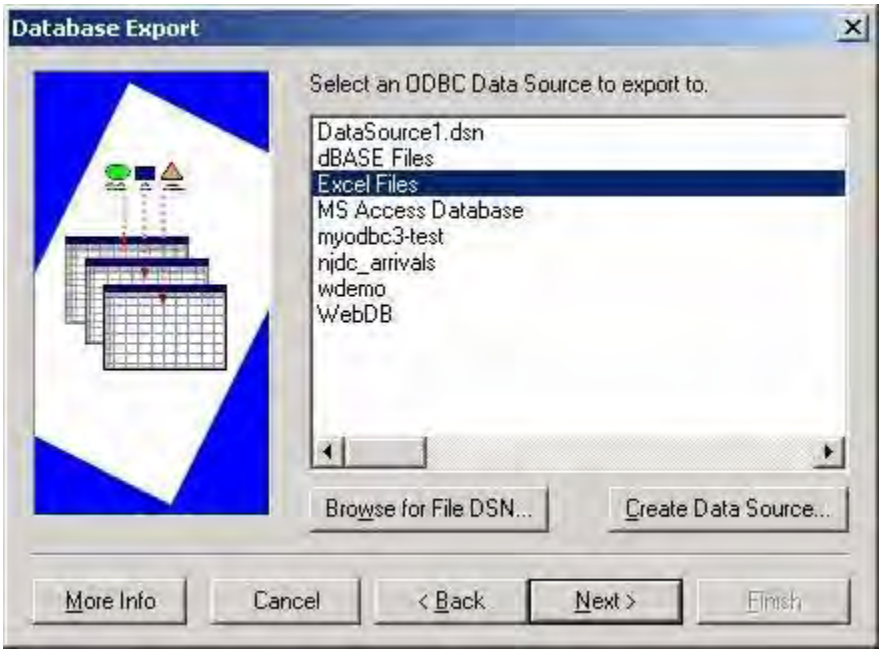


Click the “Add All >>” button to get all the correct information exported.

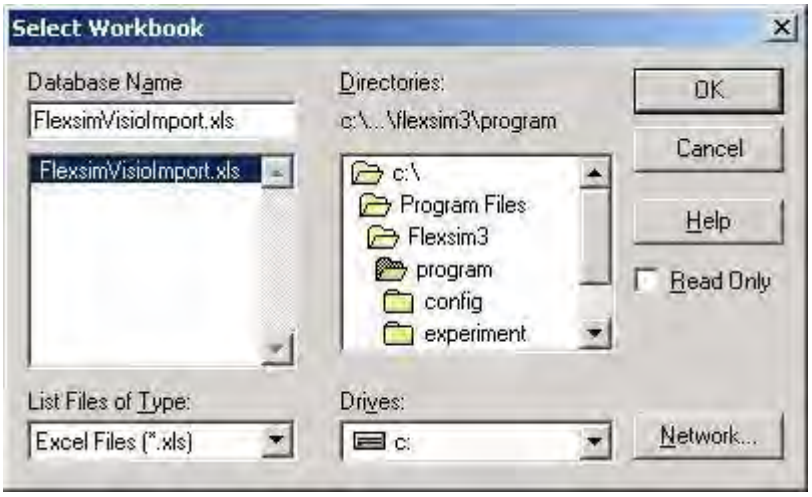


Then click Next.

Select Excel Files and click Next.

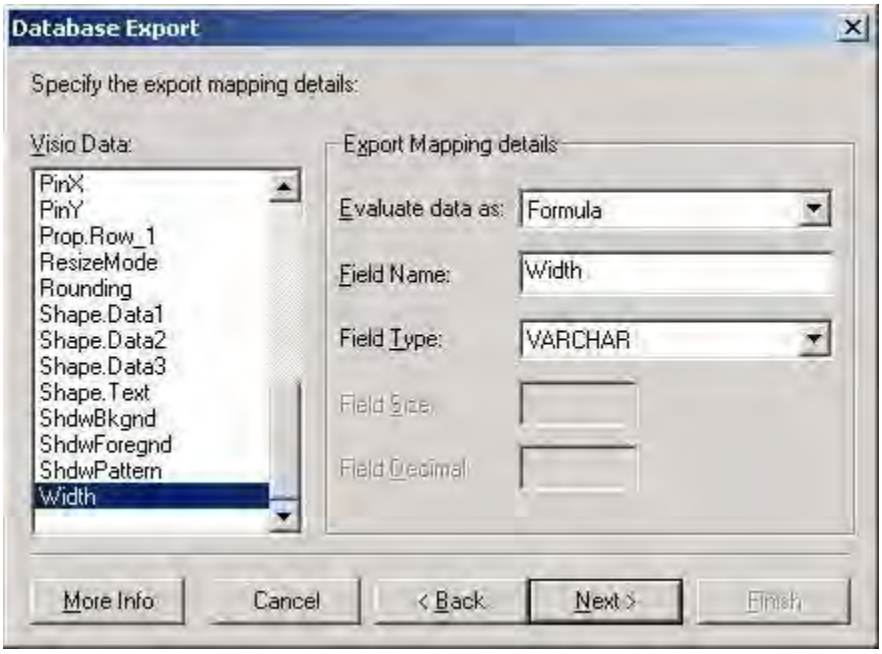


Select the excel file to export to. This should usually be C:/Program Files/Flexsim3/program/FlexsimVisioImport.xls.

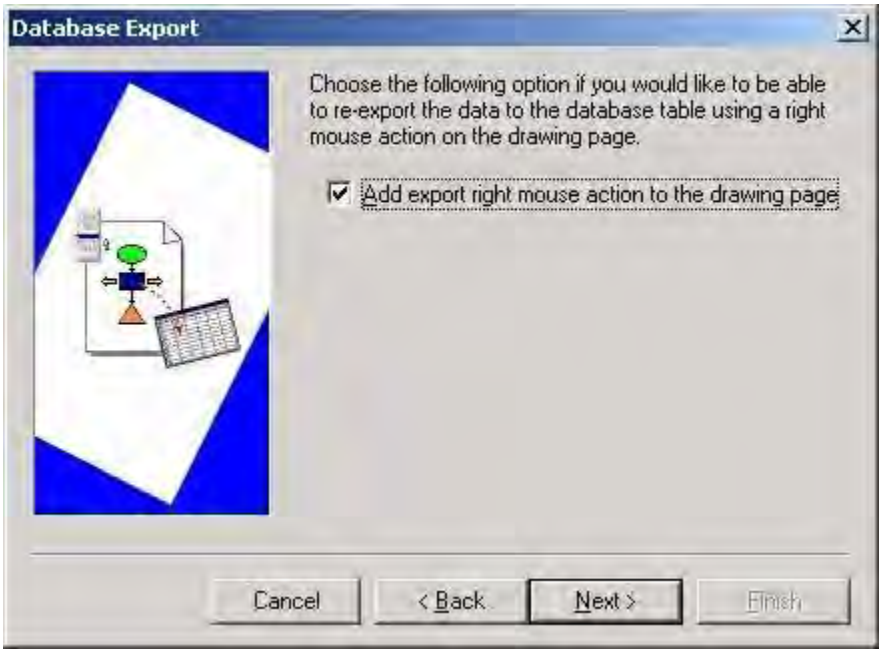


Type a name into Table Name. This name will be typed again into FlexSim so that it knows which sheet to import from. Remember the name you typed and click Next.

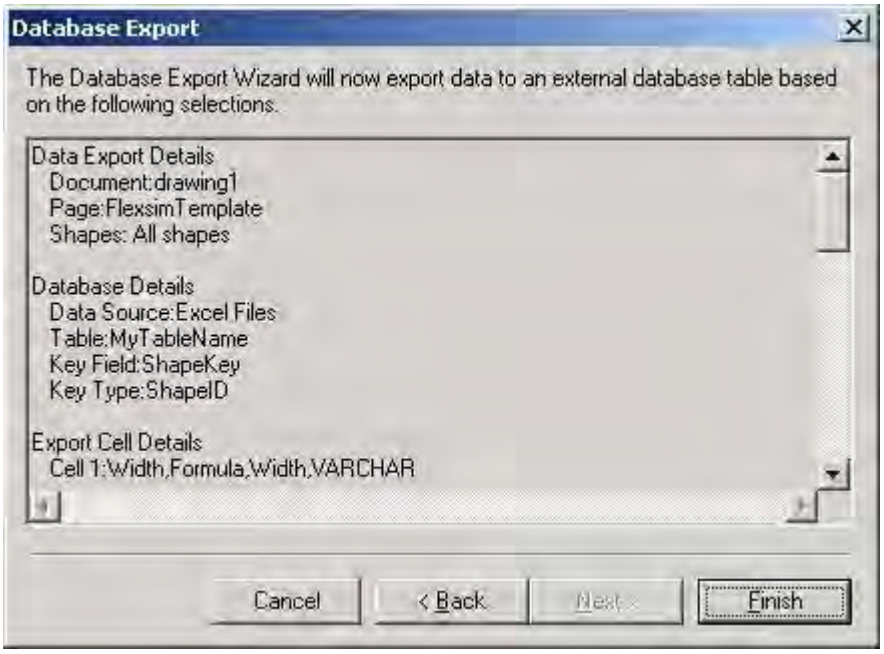
Do not change any values for the data on this screen. Click Next.


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

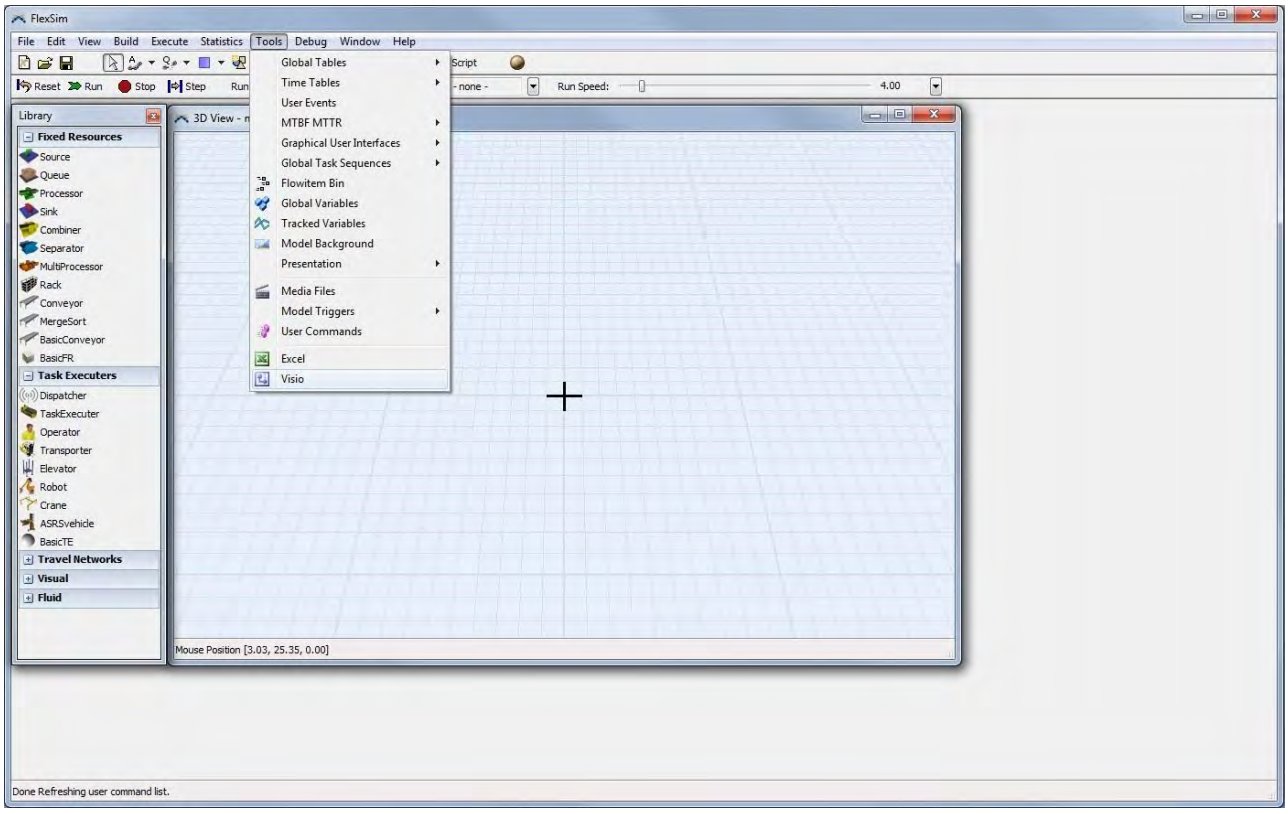
Next.


Database Export
Choose the following option if you would like to be able to re-export the data to the database table using a right mouse action on the drawing page.
Add export right mouse action to the drawing page
Cancel
< Back
Next >
Finish

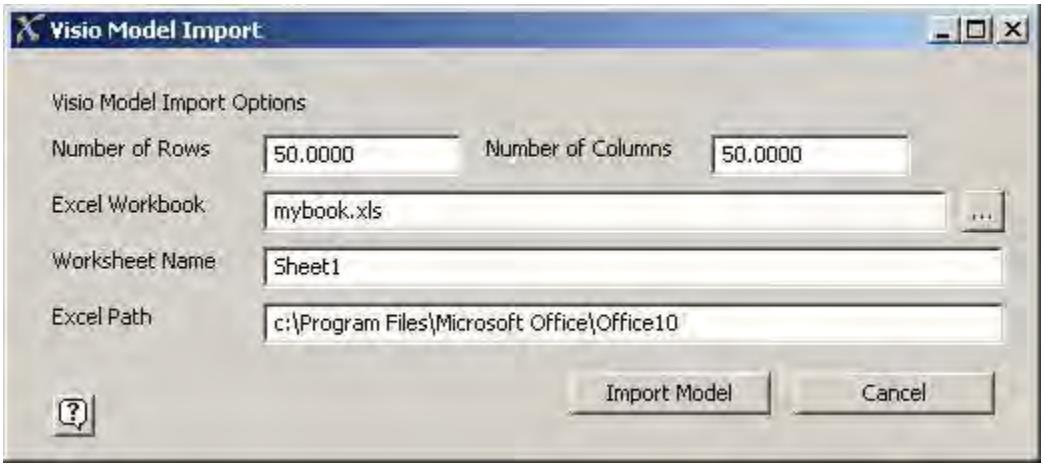
Finish.



Now, in FlexSim go to Tools > Visio…

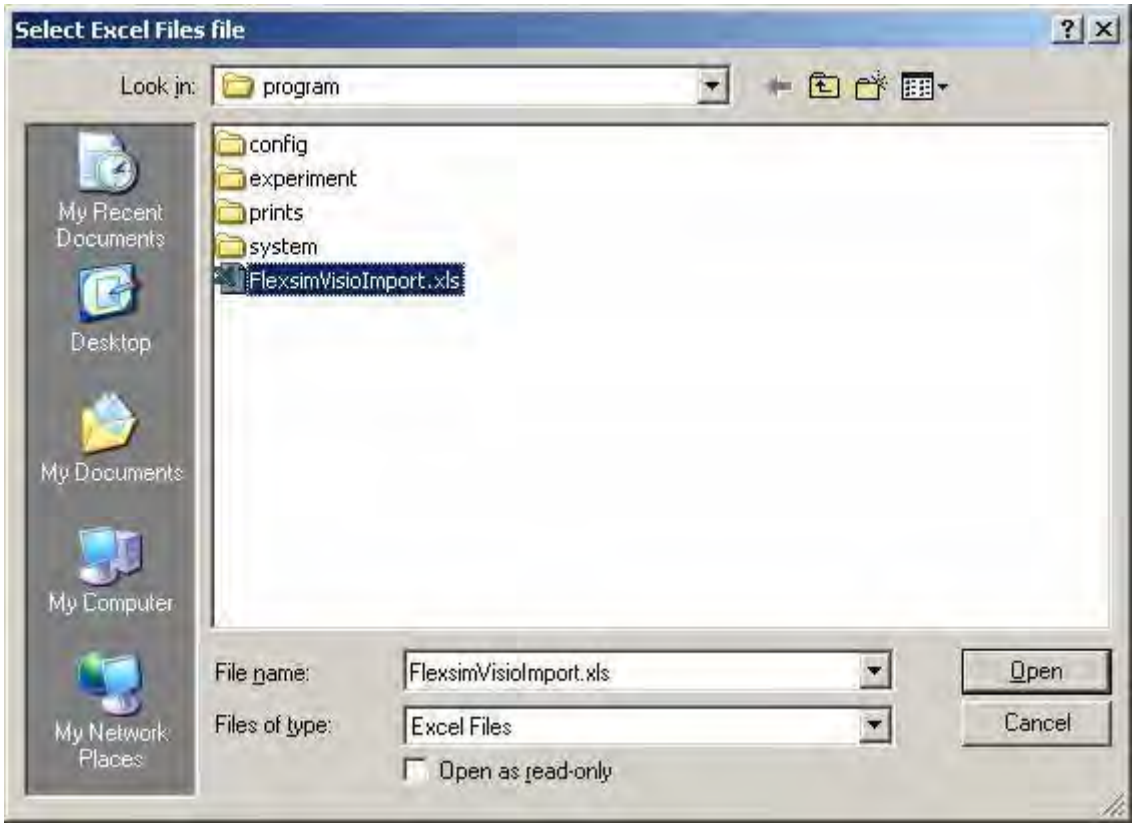


This dialog box will appear.

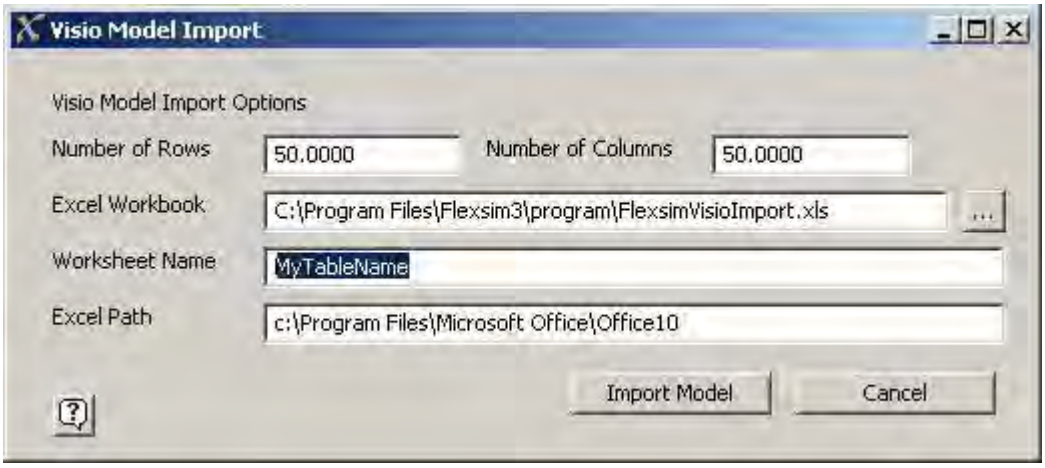


Enter a number into Number of Rows and Number of Columns that is larger than the number of rows and columns in the excel sheet that was just created by Visio Database Export. 50 is usually large enough. If your model does not import correctly, you may need to increase these values after checking your excel sheet to see how large it is.

Click the … button to browse for the correct excel file to open.

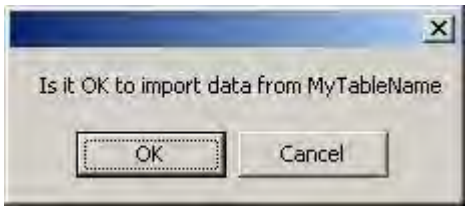


Type the Table Name that was entered earlier into the space for Worksheet Name.



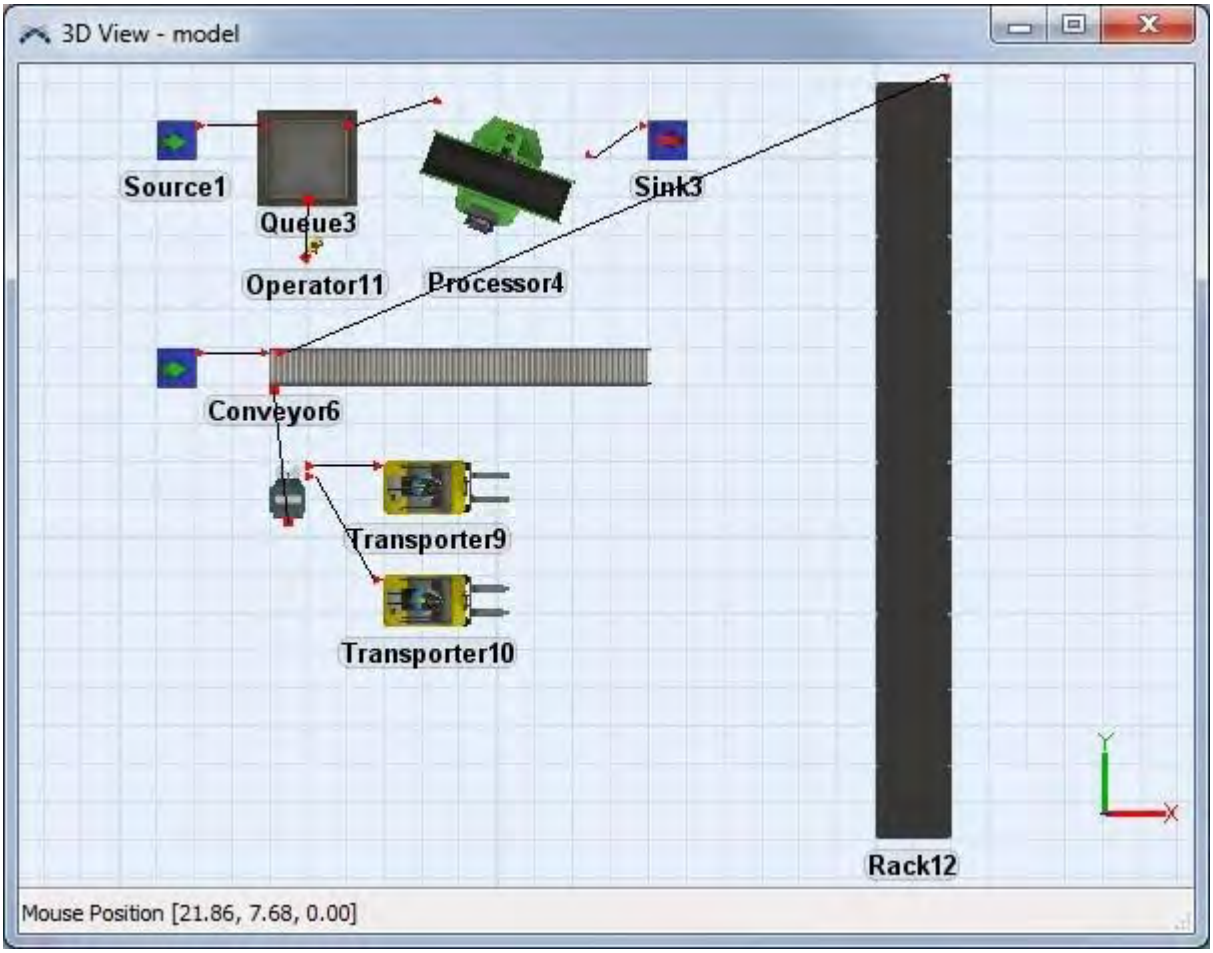
Be sure that the correct Excel Path is specified.

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.



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

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

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)

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 of curmember in the members list

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

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 station: the station where the current object is loading from or unloading to

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:

current: the current object
msgsendingobject: the object that sent the message
msgparam(1): the message's first value parameter
msgparam(2): the message's second value parameter
msgparam(3): the message's third value parameter

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. The edge numbers for a given node are found by opening the node's properties, clicking on the General tab, then selecting Output Ports in the Ports section.

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

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[,opaci ty,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,textu re])
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,width,height,thickness[,xrot,yrot,zrot,red,green,blue,opacity])
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

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

Overview

This trigger is called by the TrafficControl object when another object makes an entry request.

Access variables:

current: the current object
traveler: the object which requested entry

Overview

This trigger is called on any dispatcher object (including all task executers). It is fired whenever the object receives a task sequence. This trigger can be used to override the default passing logic, allowing more control over which object will receive the task sequence.

Access variables:

current: the current object

ts: the task sequence

Overview:

This trigger is available for either a Dispatcher or a TaskExecuter and behaves a little differently on each. For a Dispatcher, the trigger is fired whenever a downstream TaskExecuter becomes available. For a TaskExecuter, the trigger is fired whenever that TaskExecuter finishes a task sequence. If the TaskExecuter is also a Dispatcher, meaning it has a team that it dispatches tasks to, then the trigger will be fired for both cases.

If the function returns a 0, the Dispatcher/TaskExecuter will do its own dispatching logic. If the function returns a 1, the Dispatcher/TaskExecuter 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: for a Dispatcher, this is the output port of the Dispatcher; for a TaskExecuter that has just finished a task sequence the port value is 0

resource: for a Dispatcher, this is the downstream resource that has become available; for a TaskExecuter, this value is the TaskExecuter itself (or the same as current) nextts: this value is the next task sequence in the task sequence queue lastts: for a Dispatcher, this value is always NULL; for a TaskExecuter, this value is the task sequence that was just completed before it became available

Overview:

This trigger is available for either a BasicTE or BasicFR. It is fired when the object's state is changed with the setstate() command.

If the function returns a 0, the object will proceed to change its state. If the function returns -1, the object will do nothing, leaving it in its previous state. If the function returns a positive number, the state will be changed to this new state number.

Access variables:

current: the current object

state: the state that the object is being set to

profile: the state profile that state is being set on. Usually this will be 0 unless the object has defined multiple state profiles.

Setup/Process Finish Trigger (On Setup Finish/On Process Finish)

Overview:

This code is executed each time a flowitem finishes its setup/process time.

Access variables:

current: the current object item: the object that has finished its setup/process time

Overview:

This function is executed when the model is reset.

Access variables:

current: the current object

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

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

Overview:

This function returns the process time for the processing object.

Access variables:

current: the current object
item: the involved flowitem

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

Overview:

Returns the number of seconds between creation of flowitems.

Access variables:

current: the current object

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 and makes space available in the Reservoir, the in flow rate function is executed again from the OnSend event to determine the next time for 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.

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 picklist uses the same options as the Time picklists, except the return value is interpreted as a speed instead of a time.

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 number, because the function is hard-coded to return a double. There are also additional return value options for this picklist:

- If the field returns 0, then the Processor will call no operators.
- If the field returns -1, then the Processor will call no operators, but it will also call this function again at the time that it is ready to release operators. When the function is called again, the Processor will pass PICK_OPERATOR_PROCESS_RELEASE or PICK_OPERATOR_SETUP_RELEASE as the trigger parameter, depending on whether it is the setup or process step. This allows you to explicitly create task sequence(s) for calling operators, and then explicitly release them at the end, all from within the same code field. For an example of this, look at the code for the "Multiple Teams" option in the picklist.

Access variables:

current: the current object item:
the involved flowitem

trigger: the involved trigger. Possible values are PICK_OPERATOR_PROCESS, PICK_OPERATOR_SETUP, PICK_OPERATOR_PROCESS_RELEASE, PICK_OPERATOR_SETUP_RELEASE.

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

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

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

Overview:

When this object is in pull mode and it is ready to receive its next flowitem, it will first evaluate this Pull Strategy field to determine which input port to open. If the Pull Strategy 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.

PULL_REEVALUATE_WHEN_READY

If the Pull Strategy field returns PULL_REEVALUATE_WHEN_READY then no ports will be opened until the next time the Pull Strategy field is evaluated.

Access variables:

current: the current object

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

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 reevaluate 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

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

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

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

Overview:

When the TaskExecuter performs a Load or Unload task, it executes the Load / Unload picklist. The value returned is the time in model units that the TaskExecuter will delay before moving on to the next task.

Access variables:

item: a reference to item being loaded / unloaded current: the current object

station: the FixedResource object currently being loaded from or unloaded to

Overview:

This function is fired when the Dipatcher 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 TaskExecuter. 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

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

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

Overview:

This function is executed at the end of each replication run. The replication ends when the model has run out of events or the simulation end time has expired.

Access variables:

replication: current replication number

scenario: current scenario number

Overview:

This function is executed after the last replication of each scenario.

Access variables:

replication: current replication number
scenario: current scenario number

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

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.

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

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

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

Overview:

This function is executed every time the VisualTool is redrawn to display 3D text in the model view window.

Access variables:

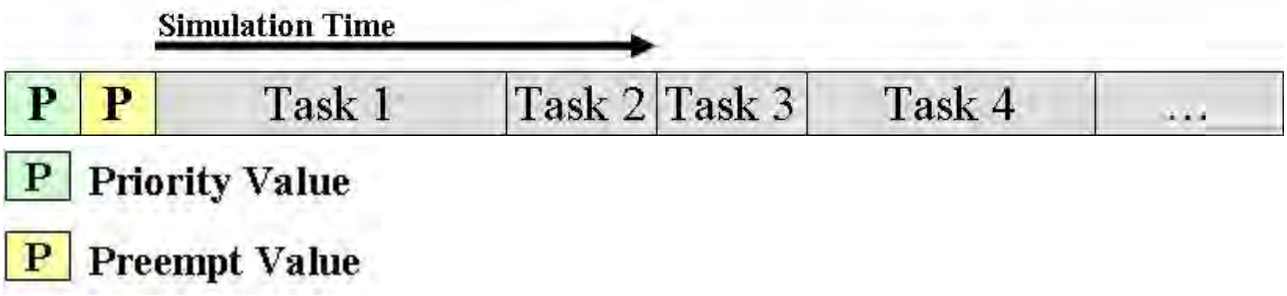
current: the current object
textnode: the node that will store the text to display (a variable node in the visual tool)

Topics

- What is a Task Sequence?
- How Task Sequences Work
- Labels on Task Sequences
- Travel Task
- Load and Unload Tasks
- Break Task
- Operator Task Sequences
- More Task Types

What is a Task Sequence?

A task sequence is a series of tasks to be executed in sequential order by a TaskExecuter, as shown in the figure below. The term TaskExecuter implies any object that inherits from the TaskExecuter class. This includes Operators, Transporters, Cranes, ASRSvehicles, Robots, Elevators, and other mobile resource objects. An object is a TaskExecuter if the TaskExecuter tab page options are in its Properties 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 Properties. 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 Processor or ProcessTimes 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

When a TaskExecuter 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.

Labels on Task Sequences

There are times when it's useful to store information on a task sequence. This can be done by using labels. Just as TaskExecuters and FixedResource objects have the ability to store labels, so does a task sequence. You can reference a label on a task sequence through code by typing:

```
ts.labelName = 5; int value =
ts.labelName;
```

This only works on the task sequence node and nodes of each individual tasks of a task sequence.

Travel Task

The "Travel" task type tells the TaskExecuter to travel to some object in the model. This may be done in several different ways, depending on the model's setup. If the TaskExecuter 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 TaskExecuter 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 TaskExecuter 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 TaskExecuters, the offset travel may differ depending on the type of TaskExecuter. 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 executer 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 TaskExecuter 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.

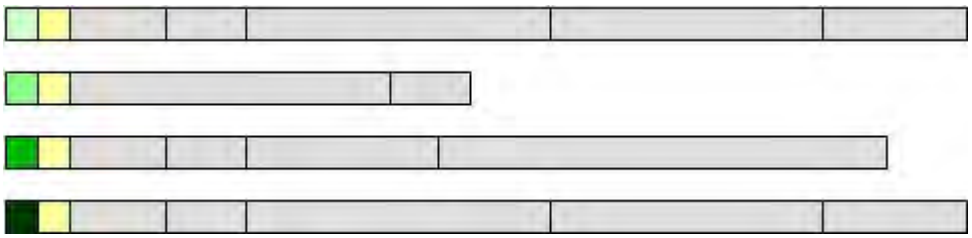
As in the previous example, the first task tells the TaskExecuter to travel to the station. The second task is a new task type not mentioned in the previous example. This is a "Utilize" task type. It tells the TaskExecuter to go into a given state, like "Utilized" or "Processing", and then wait until 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 TaskExecuter 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 TaskExecuters connected to its output ports. This is what distinguishes the Dispatcher object from the TaskExecuter and its sub-classes.

Active Task Sequence

Queue of Waiting Task Sequences



If no task sequences are preempting, then a TaskExecuter 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.

You can 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().

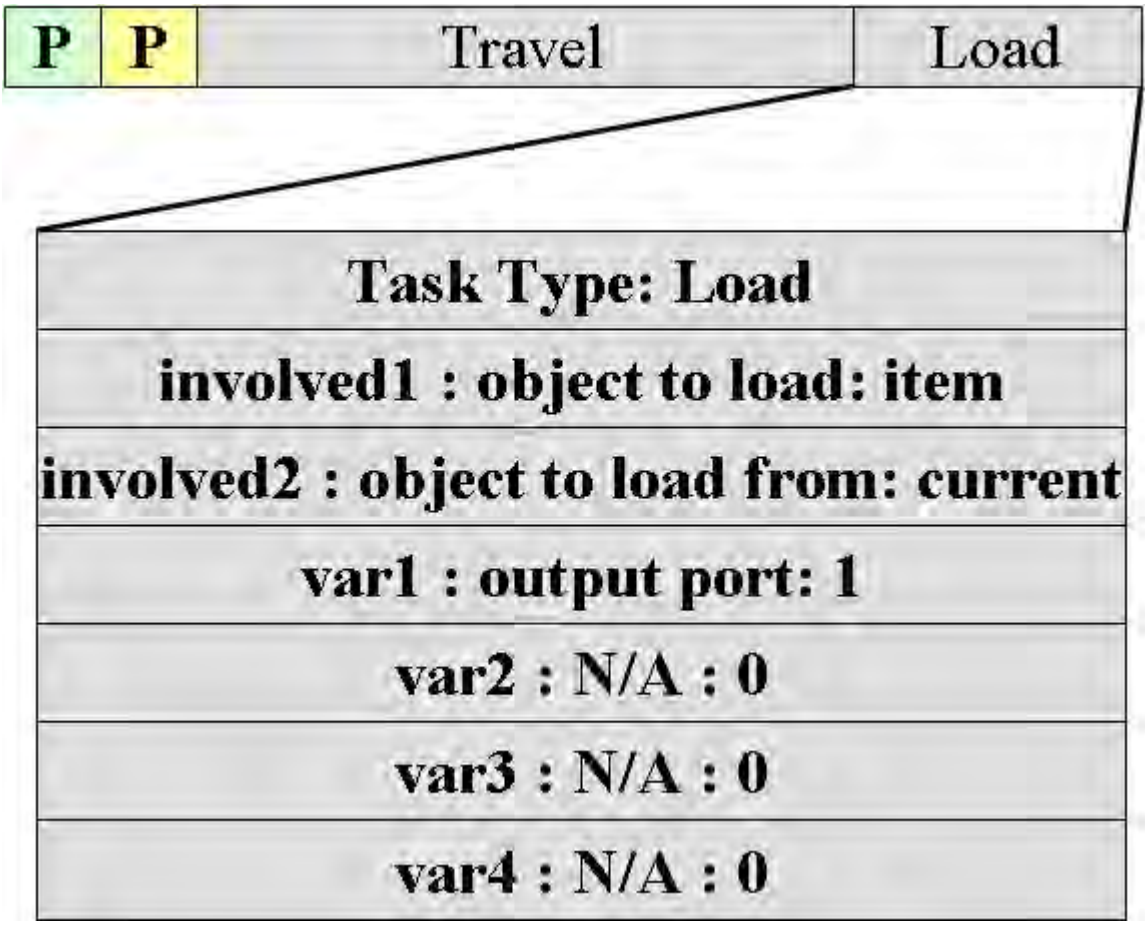
The following example tells a forklift to travel to an object, referenced as "station", then load a flow item, referenced as "item."

```
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 TaskExecuter 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.



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 var1var4. 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.

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 this 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 this value, then the TaskExecuter will stop the currently active task sequence, destroy it, and destroy all task sequences in its task sequence queue.

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 Commands documentation found through FlexSim's Help menu.

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 TaskExecuter, 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.

Preempting Travel Tasks on Networks with Traffic Controls

Preemption can have some undesirable side effects if a TaskExecuter is traveling along a network with TrafficControls when the preemption happens. If preemption occurs when a TaskExecuter is traveling along a network edge, then the TaskExecuter will be "taken off" that edge and connected in an "inactive"

travel state (the red line) to the next network node he would have arrived at had he finished traveling the edge. If the network node is a member of a TrafficControl whose area starts at that node (in other words, the TaskExecuter wasn't in the area yet when he was preempted), then the TaskExecuter will be "forced" into the traffic control area, meaning this may cause the number of objects in the area to exceed the maximum for that traffic control area. To avoid this, special logic has been added to the TaskExecuter's preemption mechanism, so that if he is preempted from a travel task directly to another travel task, then instead of being taken off the edge and connected to the next node, the preemption will call redirectnetworktraveler(), which will essentially keep him traveling as he was before on the edge, but his final destination will be changed so that when he arrives at the end of the edge, he will continue on a new path to the new destination. Note that this will only happen if the preemption mechanism can detect that the first durative task that the TaskExecuter will perform after the preemption is another travel task. By durative we mean any task that will take some amount of time to perform: TASKTYPE_TRAVEL, TASKTYPE_LOAD, TASKTYPE_UTILIZE, TASKTYPE_DELAY, etc. are durative, whereas TASKTYPE_SETNODENUM, TASKTYPE_TAG, TASKTYPE_MOVEOBJECT, TASKTYPE_SENDMESSAGE, TASKTYPE_NODEFUNCTION are not. If the preemption detects that the next durative task is not a travel task, then it will take the TaskExecuter off the edge as described above, in which case the TaskExecuter may be forced into a traffic control area.

Preempting With Dispatchers

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 TaskExecuter, then it will do just that, and not send the preempting task sequence immediately to a TaskExecuter. If you want the Dispatcher to dispatch preempting task sequences immediately, then you will need to specify such logic in its Pass To function.

A TaskExecuter may be connected back to a Dispatcher by dragging an A-connect from the TaskExecuter to the Dispatcher. If this is done, then when the TaskExecuter receives a preempting task it will pass its current task back to the Dispatcher. The Dispatcher will then redistribute that task according to its dispatching logic. Tasks returned to the Dispatcher in this manner are returned in their current state so that the next TaskExecuter will begin where the previous TaskExecuter left off. This may cause some odd behavior that you should take into consideration when assigning preempting tasks. For example, if a TaskExecuter is traveling with an item when it is preempted, the TaskExecuter that receives this task will perform the travel and unload tasks without ever picking up the item from the first TaskExecuter. Instead, the item will "magically" appear at the right location when the unload completes. In order to prevent these odd behaviors you may want to query the TaskExecuter's state to determine if it is in a "preemptable" state (as you define it) before assigning the new task.

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.

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()

insertdeallocatetask()

insertsynctask() insertproxytask()

dispatchcoordinatedtasksequence()

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 TaskExecuter 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 TaskExecuter 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 TaskExecuter or Dispatcher. In the case that it is a dispatcher, meaning you want to allocate any one of several TaskExecuters, 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 third and fourth parameters are the priority and preempting values of the separate task sequence that will be created. The fifth 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.

insertsynctask

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,

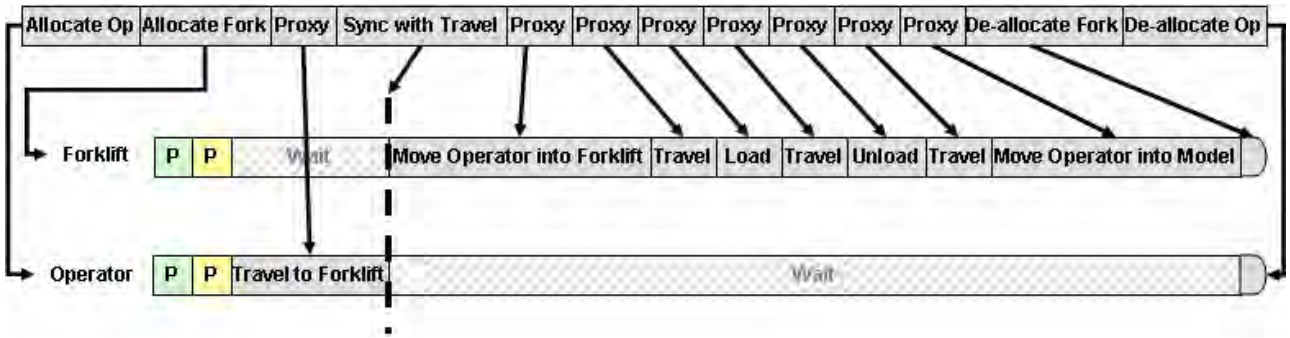
unless a sync task is specified, whereas proxy tasks given to the same TaskExecuter will automatically be done in sequential order, without the need for a sync task.

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.

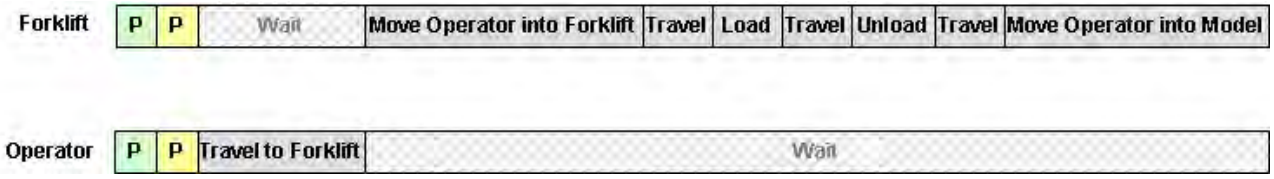
The above code creates a coordinated task sequence that organizes the two task sequences, as shown in the diagram below.

Coordinated Task Sequence



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.



Code

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.

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 Sequences Quick Reference

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	end speed		
TASKTYPE_UNLOAD	item to unload	station	input port	end speed		
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	userdefined	userdefined	userdefined	userdefined
TASKTYPE_MILESTONE	NULL	NULL	range	N/A	N/A	N/A
TASKTYPE_NODEFUNCTION	node	parnode(1)	pv(2)*	pv(3)*	pv(4)*	pv(5)*
TASKTYPE_STARTANIMATION	object	NULL	animationnr	durationtype	durationvalue	
TASKTYPE_STOPANIMATION	object	NULL	animationnr			
TASKTYPE_FREEOPERATORS	object	involved				
TASKTYPE_WAITFORTASK	NULL	NULL	state			

Non-user tasks						
TASKTYPE_TE_STOP	NULL	NULL	state			
TASKTYPE_TE_RETURN	task sequence	task				
TASKTYPE_TE_ALLOCATED	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 Sequence Types

You can also refer to the task type reference guide for quick reference. Task types are as follows.

TASKTYPE_TRAVEL	
Here the TaskExecuter travels to the object specified. This is done by making a travel request to the navigator that it is connected to. The navigator then takes over control of the TaskExecuter, and pushes it like a pawn on a chessboard, according to the navigator's own logic, until the TaskExecuter reaches the destination and the navigator notifies the TaskExecuter that the travel task is finished. How the TaskExecuter is pushed is dependent on the type of navigator. If the TaskExecuter is connected to a network, then its associated navigator is a network navigator and will push the TaskExecuter along network paths. Note on TaskExecuters and navigators: Some objects by default are not connected to a navigator at all. If the TaskExecuter 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, Crane.	
involved1	The object to travel to.
involved2	Not used. Use NULL for this parameter.
var1	This specifies the desired end speed for the travel operation. If 0, then the desired end speed will be the maximum speed of the TaskExecuter. A positive value will use that value directly as the end speed. If the end speed is negative, then the functionality is dependent on the navigator's logic. The standard navigators, i.e. the default navigator and the network navigator will interpret negative end speeds as an end speed of 0. Other non-standard navigators may however use a negative end speed value to customize the logic for how to get the TaskExecuter to the destination, allowing the modeler to use different negative end speeds on travel tasks to effect customized travel logic.
var2	Specific to the network navigator. If this value is 1, then the object will travel to the destination node even he is already connected to it.
var3 - var4	N/A

Example	inserttask(ts, TASKTYPE_TRAVEL, outobject(current,1))

TASKTYPE_LOAD

This task causes the TaskExecuter to load an item from a station. If the TaskExecuter's "Travel Offsets for Load/Unload Tasks" checkbox is checked in its Properties 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 TaskExecuter will figure out the load time. At the end of the load time, the TaskExecuter will move the item into itself. If loading into a FixedResource, 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.

involved1	the object to load (usually a flow item).
involved2	the object to load from.
var1	this is the output port through which the object will exit the station. Usually a 0 is fine.
var2	The requested end speed for the task.
var3 - var4	N/A
Example	inserttask(ts, TASKTYPE_LOAD, item, current)

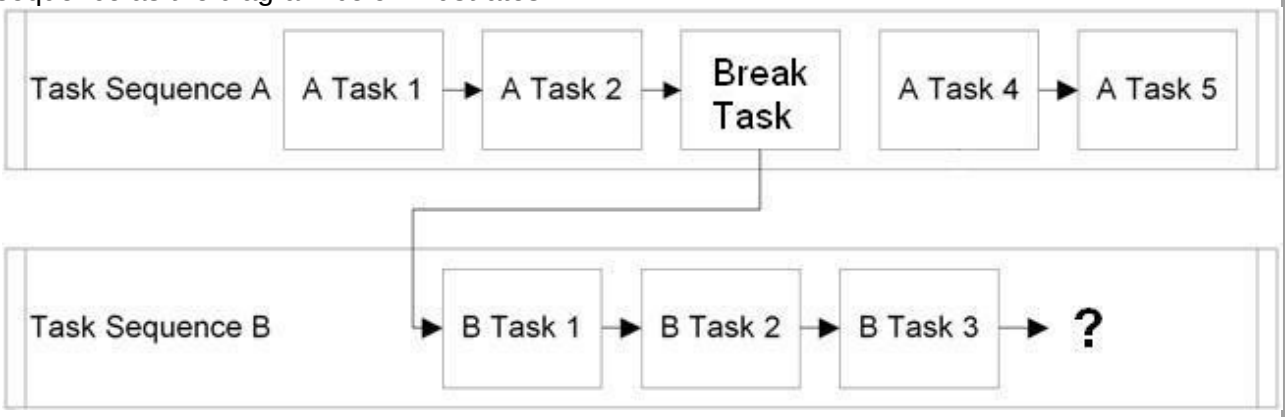
TASKTYPE_UNLOAD

This task causes the TaskExecuter to unload an item to a station. If the TaskExecuter's "Travel Offsets for Load/Unload Tasks" checkbox is checked in its Properties page, then it will travel to a drop-off location by querying the station and using offset travel. Then the TaskExecuter will figure out the unload time. At the end of the unload time, the TaskExecuter will move the item into the station. If unloading from a FixedResource, 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.	
involved1	the object to unload (usually a flow item).
involved2	the object to unload to.
var1	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 or a Separator. The Combiner needs to know which input port the item is coming through in order to update its input table. The Separator just needs a value that is greater than 0.
var2	The requested end speed for the task.
var3 - var4	N/A

Example	inserttask(ts, TASKTYPE_UNLOAD, item, outobject(current, 1))

TASKTYPE_BREAK

This task causes the TaskExecuter 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 TaskExecuter will call its "Break To Requirement" function. This function should return a reference to the task sequence that you want the TaskExecuter to break to. In your break logic you may search through the TaskExecuter'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 TaskExecuter to break at all, then return NULL.

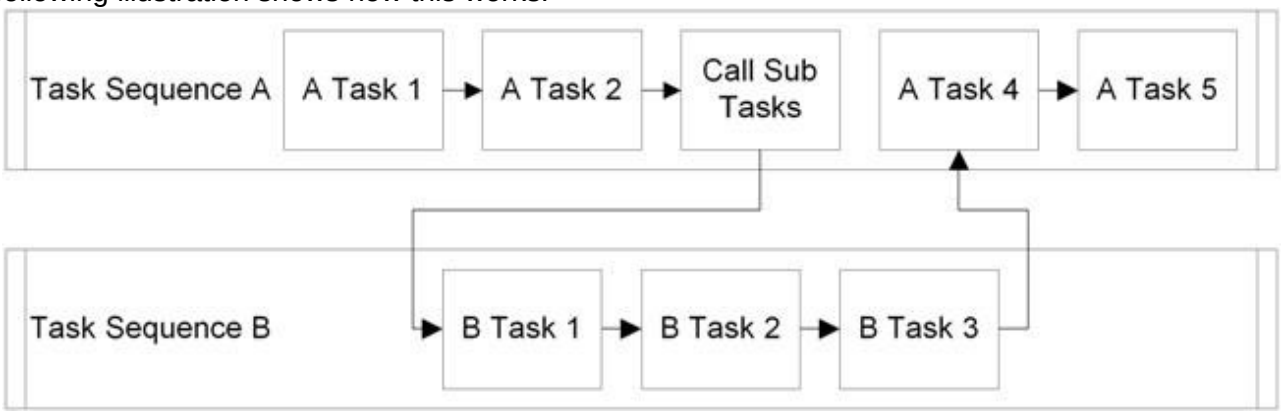
involved1	If involved1 is specified, then it should be a reference to some object in the model. Instead of calling the "Break To Requirement" the TaskExecuter will send a message to this object. The only difference here is the place in which you place your logic for finding a task sequence to break to. 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."
involved2	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. Note on using both involved parameters: 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.
var1	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 Properties 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 parval(3) if it is to call its Break To Requirement function, and as msgparam(1) if it is to send a message.
var2	This parameter specifies whether or not the TaskExecuter should check to receive task sequences from an upstream Dispatcher. By default (0), the TaskExecuter 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 started 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 TaskExecuter 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.
var3 - var4	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.

Example	inserttask(ts, TASKTYPE_BREAK, NULL, NULL) //a basic break. the "Break To" Requirement on the TaskExecuter tab will fire
	inserttask(ts, TASKTYPE_BREAK, centerobect(current, 1), NULL)
	//Sends a message to the referenced object, where your logic will be written in the OnMessage trigger of the object
	inserttask(ts, TASKTYPE_BREAK, NULL, specificTaskSequence)
	//breaks to a specified task sequence

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 TaskExecuter 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 TaskExecuter 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 won't have a reference to the item that you want to load until the TaskExecuter arrives at the other side of the model. In this case, you want the TaskExecuter 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 TaskExecuter arrives at the load location. Call sub tasks can also be hierarchical. This means that Task Sequence B can also have a call sub tasks type in it. If you decide to create a task sequence when the TaskExecuter gets to the call sub tasks task, then the involved object you call sub tasks on will receive a message. In that object's OnMessage trigger, you will need to create a new task sequence using createemptytasksequence(), and insert tasks using inserttask(), but DO NOT dispatch the task sequence, instead, simply return the reference to your newly created task sequence.

Note on the Break To Requirement function: When the TaskExecuter 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 TaskExecuter that executes the call sub tasks task from Task Sequence A must be the first object to be allocated in the Task Sequence B.

involved1	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 do one of two things. If involved1 is a reference to an object (a node with object data) then the TaskExecuter will send a message from itself to the object specified by the involved1 parameter. If involved1 is a reference to a dll, flexscript, or c++ node (a node with string data) then it will call nodefunction on that node, and it will pass a reference to itself as parnode(1). The only difference here vs. the default case 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 or nodefunction. Again, the return value of the message/nodefunction 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."
involved2	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. Note on using both involved parameters: 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.

var1 - var4	These parameters are passed into the Break To Requirement as parval(3), parval(4), parval(5) and parval(6). When sending a message, var1, var2, and var3 are passed into the message as msgparam(1),msgparam(2) and msgparam(3) respectively. When calling a nodefunction, var1, var2, var3 and var4 are passed in as parval(2), parval(3), parval(4), and parval(5).

Example	inserttask(ts, TASKTYPE_CALLSUBTASKS, NULL, NULL) // The "Break To" Requirement on the TaskExecuter tab will fire
	inserttask(ts, TASKTYPE_CALLSUBTASKS, centerobect(current, 1), NULL) // Sends a message to the referenced object, where your logic will be written in the OnMessage trigger of the object
	inserttask(ts, TASKTYPE_CALLSUBTASKS, NULL, specificTaskSequence) // breaks to a specified task sequence

TASKTYPE_UTILIZE

This task causes the TaskExecuter 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.

involved1	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. 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. 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 this 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.
involved2	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.
var1	This is the state into which the operator will go during the utilize task. If it is 0, then the TaskExecuter will go into STATE_UTILIZE.
var2 - var4	N/A
Example	inserttask(ts, TASKTYPE_UTILIZE, item, NULL, STATE_UTILIZE)

TASKTYPE_STOPREQUESTBEGIN

This task causes the TaskExecuter to call stopobject() on the involved1 object. Refer to the stopobject() command documentation for more information.

involved1	This parameter specifies the object to call stopobject() on. If NULL, then the TaskExecuter will call stopobject on himself.
involved2	Not used. Use NULL for this parameter.
var1	This is the state to request the stopped object to go into.

var2	This variable is necessary only if the TaskExecuter 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 TaskExecuter will change the variable to 2 as a flag to not execute it again.
var3	This is the id for the stopobject command

var4	This is the priority of the stopobject command
Example	inserttask(ts, TASKTYPE_STOPREQUESTBEGIN, current)

TASKTYPE_STOPREQUESTFINISH

This task causes the TaskExecuter to call resumeobject() on the involved1 object. Refer to the resumeobject() command documentation for more information.

involved1	This parameter specifies the object to call resumeobject() on.
involved2	N/A
var1	This variable is necessary only if the TaskExecuter 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.
var2	This is the id for the resumeobject() command.

var3	N/A
var4	This variable is managed by the TaskExecuter, and tells whether this task has already been executed once.
Example	inserttask(ts, TASKTYPE_STOPREQUESTFINISH, current)

TASKTYPE_SENDMESSAGE

This task causes the TaskExecuter to send a message to the involved1 object.

involved1	Involved1 is the object that the message is sent to. If NULL, then a message is sent to the TaskExecuter himself.
involved2	Involved2 specifies msgsendingobject in the message trigger. If NULL, then msgsendingobject is the TaskExecuter himself. Usually this will be NULL, because it is the only way that you can access the TaskExecuter within the message trigger.

	However, you may want the message to be sent "from" a different object, so you have the option here.
var1	This parameter is passed in as msgparam(1) in the message trigger.
var2	This parameter is passed in as msgparam(2) in the message trigger.
var3	This parameter is passed in as msgparam(3) in the message trigger.

var4	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. 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 TaskExecuter 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 TaskExecuter 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 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.
Example	inserttask(ts, TASKTYPE_SENDMESSAGE, current, NULL, p1,p2,p3, delay)

TASKTYPE_DELAY

This task causes the TaskExecuter to go into a given state, and then simply wait for a specified amount of time.

involved1	Not used. Use NULL for this parameter.
involved2	Not used. Use NULL for this parameter.

var1	This is the amount of time that the TaskExecuter will wait in the specified state.

var2	This is the state into which the operator will go during the delay task. If it is 0, then the TaskExecuter will remain in the previous state it was in.
var3	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.
var4	N/A
Example	inserttask(ts, TASKTYPE_DELAY, NULL, NULL, time, STATE_NUMBER)

TASKTYPE_MOVEOBJECT

This task tells the TaskExecuter to move a specified object into a specified container. This would be used if you want the TaskExecuter 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 TaskExecuter.

involved1	The object to move.
involved2	The object to move involved1 into.
var1	The output port of the object that involved1 will exit. 0 is usually fine.
var2 - var4	N/A
Example	inserttask(ts, TASKTYPE_MOVEOBJECT, item, outobject(current, 1))

TASKTYPE_DESTROYOBJECT

This task tells the TaskExecuter 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.

involved1	The object to destroy.
involved2	Not used. Use NULL for this parameter.
var1 - var4	N/A.
Example	inserttask(ts, TASKTYPE_DESTROYOBJECT, item, NULL)

TASKTYPE_SETNODENUM

This task causes the TaskExecuter to set the value on a specified node. This would be used if you want to set a variable or label on the object.

involved1	The node to set the value on. This can be something like label(current, "mylabel"). or var_s(current, "maxcontent")
involved2	Not used. Use NULL for this parameter.
var1	The value to set the node to.
var2	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.
var3 - var4	N/A.

Example	inserttask(ts, TASKTYPE_SETNODENUM, theNode, NULL, 42)

TASKTYPE_TRAVELTOLOC

This task causes the TaskExecuter to travel to a specified location using offset travel.

involved1	Not used. Use NULL for this parameter.
involved2	Not used. Use NULL for this parameter.
var1	This is the x location to travel to.
var2	This is the y location to travel to.
var3	This is the z location to travel to.
var4	This is the desired end speed.
Example	inserttask(ts, TASKTYPE_TRAVELTOLOC, NULL, NULL, 3,3,3)

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.

involved1	Not used. Use NULL for this parameter.
involved2	Not used. Use NULL for this parameter.

var1	This is the x offset to travel.
var2	This is the y offset to travel.
var3	This is the z offset to travel.
var4	This is the desired end speed.
Example	inserttask(ts, TASKTYPE_TRAVELRELATIVE, NULL, NULL, 1,0,0)

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 organize 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.

involved1	Just like in a load task, this is the reference to the item that would be loaded.
involved2	Just like in a load task, this is the reference to the station from which the item would be loaded.

var1 - var3	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 TaskExecuter will travel none of the corresponding offset. If 1, then the TaskExecuter 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 TaskExecuter would travel 90% of the corresponding offset.
var4	This is the desired end speed.
Example	inserttask(ts, TASKTYPE_PICKOFFSET, item, current, 1,0,0)

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.

involved1	Just like in an unload task, this is the reference to the item that would be loaded.
involved2	Just like in an unload task, this is the reference to the station to which the item would be unloaded.
var1 - var4	Same as pick offset task.
Example	inserttask(ts, TASKTYPE_PLACEOFFSET, item, outobject(current, 1), 1,0,0)

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.

involved1	For your use.
involved2	For your use.

var1 - var4	For your use
Example	inserttask(ts, TASKTYPE_TAG, current, centerobject(current, 1), 1)

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 TaskExecuter can revert back to if it is preempted away from the task sequence. Normally when a TaskExecuter 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 TaskExecuter 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 TaskExecuter 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.

class="note">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 subtask sequence, you can insert milestone tasks as needed.

involved1	Not used. Use NULL for this parameter.
involved2	Not used. Use NULL for this parameter.

var1	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 TaskExecuter is preempted while executing any one of tasks 6, 7 or 8, then it will revert back to the milestone task.
var2 - var4	N/A
Example	inserttask(ts, TASKTYPE_MILESTONE, NULL, NULL, 3)

TASKTYPE_NODEFUNCTION

This task type will call nodefunction() on the specified node.

involved1	The node to call nodefunction() on.
involved2	Passed in as parnode(1). If specified as NULL, then when the TaskExecuter executes the task, it will pass a reference to itself as parnode(1).
var1 - var4	These parameters are passed in as parval(2), parval(3), parval(4), and parval(5) in the nodefunction.
Example	inserttask(ts, TASKTYPE_NODEFUNCTION, node, NULL, 1)

TASKTYPE_STARTANIMATION

This task type will call startanimation().

involved1	Passed as parameter 1 to startanimation(). If NULL, the TaskExecuter will pass a reference to itself.
involved2	Not used. Pass NULL.
var1	Passed as the animation parameter into startanimation(). Defines the rank of the animation to start.
var2	Passed as the durationtype parameter into startanimation().
var3	Passed as the durationvalue parameter into startanimation().
var4	Not used. Pass 0.
Example	inserttask(ts, TASKTYPE_STARTANIMATION, NULL, NULL, 1)

TASKTYPE_STOPANIMATION

This task type will call stopanimation().

involved1	Passed as parameter 1 to stopanimation(). If NULL, the the TaskExecuter will pass a reference to itself.
involved2	Not used. Pass NULL.
var1	Passed as the animation parameter into stopanimation(). Defines the rank of the animation to stop.
var2-var4	Not used. Pass 0.

Example	inserttask(ts, TASKTYPE_STOPANIMATION, NULL, NULL, 1)

TASKTYPE_FREEOPERATORS

This task type will call freeoperators(). This might be used as an alternative or as a supplement to the coordinated task sequence mechanism. It allows you, as part of one TaskExecuter's task sequence, to free another TaskExecuter from a utilize task.

involved1	Passed as parameter 1 to freeoperators().
involved2	Passed as parameter 2 to freeoperators(). If NULL, the TaskExecuter will pass a reference to itself.
var1-var4	Not used. Pass 0.

Example	inserttask(ts, TASKTYPE_FREEOPERATORS, centerobject(current, NULL, 1), NULL);

TASKTYPE_WAITFORTASK

This task type is used if you don't yet have any new tasks to give the object, but you want to disallow the object from finishing the task sequence until you do have tasks to give it. The object will simply wait until the next task is added to the task sequence, and then will finish this task.

involved1	Should be NULL.
involved2	Should be NULL.

var1	Here you can optionally pass a state that you want the object to go into. If 0, STATE_IDLE will be used.
var2-var4	Not used. Pass 0.
Example	inserttask(ts, TASKTYPE_WAITFORTASK, NULL, NULL);

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.	
involved1	Not used. Use NULL for this parameter.
involved2	Not used. Use NULL for this parameter.
var1	The state that the TaskExecuter should go into when he is down.
var2 - var4	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	
involved1	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.

involved2	This parameter points to the actual call sub tasks task as a node.
var1- var4	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.	
involved1	The object coordinating the task sequence.
involved2	This is a reference to the coordinated task sequence being executed.
var1 - var4	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 TaskExecuter. 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.	
involved1	This is a reference to a dispatcher to give the task sequence to. It may be a specific TaskExecuter, or, if the object to allocate can one of several possible objects, then it can reference a dispatcher that dispatches to those task executers.
involved2	This is not specified when the task is created, but will be set once the object has been allocated and will reference that object.

var1	The priority value for the allocation
var2	The preempt value for the allocation
var3	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
var4	This tells whether the task sequence is blocking. Default (0) is blocking, 1 is nonblocking

TASKTYPE_CT_SYNC

Here the task coordinator blocks the continuation of the task sequence until some previously specified task (referenced by rank), is finished.

involved1	Not used. Use NULL for this parameter.
involved2	Not used. Use NULL for this parameter.
var1	The rank of the task to sync to.
var2 - var4	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.

involved1	Not used. Use NULL for this parameter.
involved2	Not used. Use NULL for this parameter.
var1	The rank of the allocation task that allocated the object.
var2	This variable specifies whether the deallocation task is blocking. Default (0) is blocking. 1 is non-blocking.
var3 - var4	N/A

Querying Information on Task Sequences

Once you have dispatched task sequences, you can 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/taskexecuter 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 executer, 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 TaskExecuter 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 TaskExecuter 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

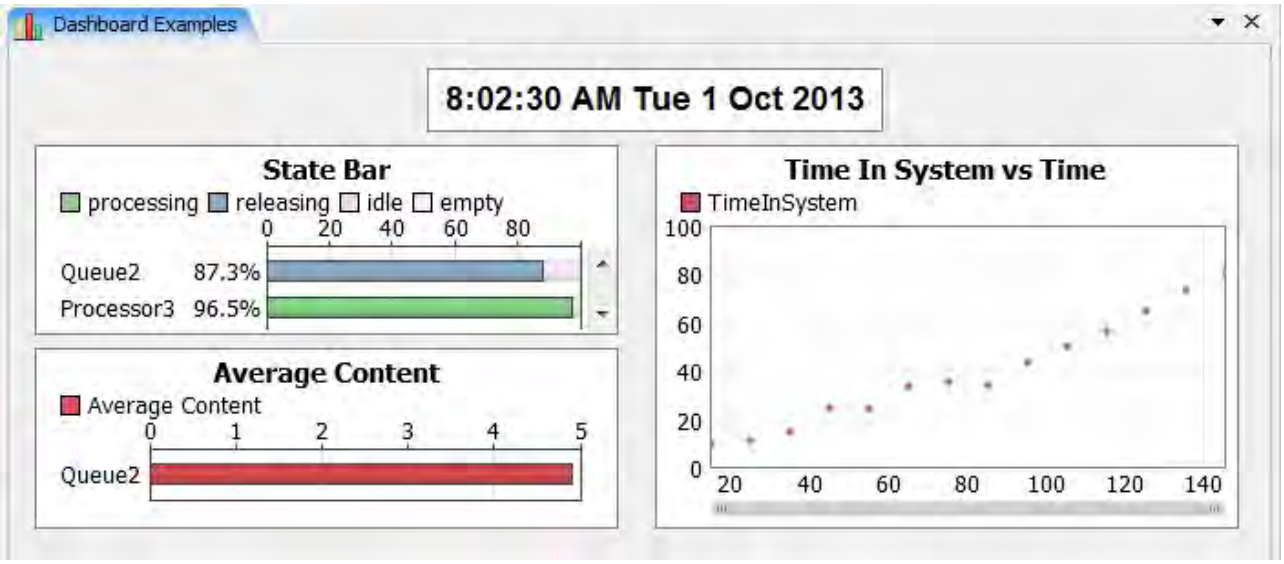
int getpreempt(treenode tasksequence)

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.

Dashboard Concepts



Topics

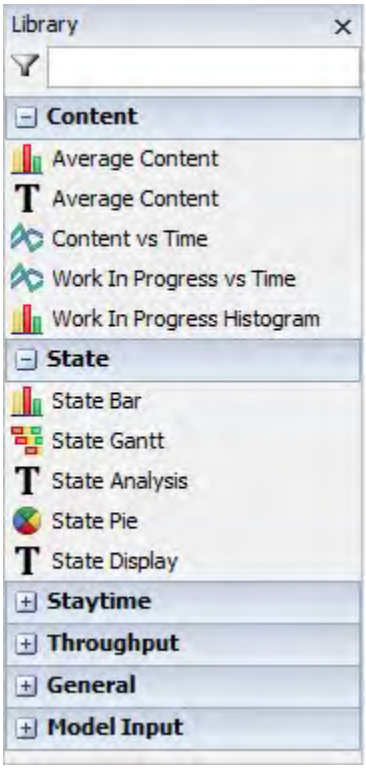
- Edit Mode
- Creating and Arranging Widgets
- Exporting Dashboards
- Graph Types o Text Panel o Bar Chart o Pie Chart o Line Graph o Histogram o Gantt Chart o Financial Analysis o Custom Chart o Date and Time Display o Model Documentation • Model Input

Dashboards are accessed from the Toolbox. (View menu > Toolbox > Add > Statistics > Dashboard). The Dashboard window allows you to view graphs and statistics for the model as it runs. It is especially useful for comparing objects side by side.

Note: Not all statistics make sense for all objects. If a selected object does not have the statistic specified, the graph will not display data for that object.

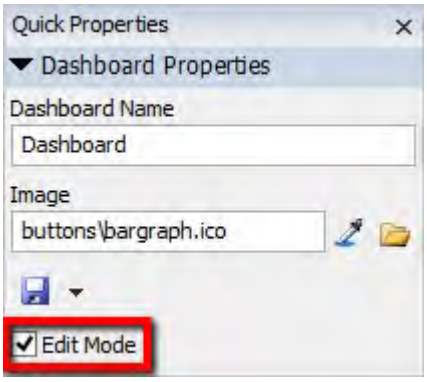
Dashboards are accessed from the Toolbox. (View menu > Toolbox > Add > Statistics > Dashboard).

The Library Icon Grid will change to display all of the dashboard widgets that can be created:



Dashboard Graph Types: The list of Dashboard Graphs displayed in the Library Icon Grid are just a starting point, a list of presets. Graphs may be customized to display additional information by editing the Graph's properties.

Edit Mode



When you first create a Dashboard, it will be in edit mode. This allows you to move and resize the Dashboard's widgets. Unchecking the Edit Mode will lock down all widgets and allow you to interact with the Model Input objects. The edit mode may also be changed through the right-click menu.

Creating and Arranging Widgets

To add a widget from the Library Icon Grid click and drag the widget to the dashboard window or left-click once on the desired widget to enter creation mode, then left-click again on the Dashboard to create a new widget. Right-click or press the Escape key to exit creation mode.

You can also add widgets through the Quick Library. Double click anywhere in the Dashboard to bring up the Quick Library. Select the desired widget and it will be added at the point where you double clicked.

To arrange widgets, left-click on the widget you wish to move. Sizers will appear around the outside of the widget. Click and drag the widget to move it, or click and drag on one of the sizers to resize the widget.

Hold the Control key down and click on dashboard widgets to select multiple widgets.

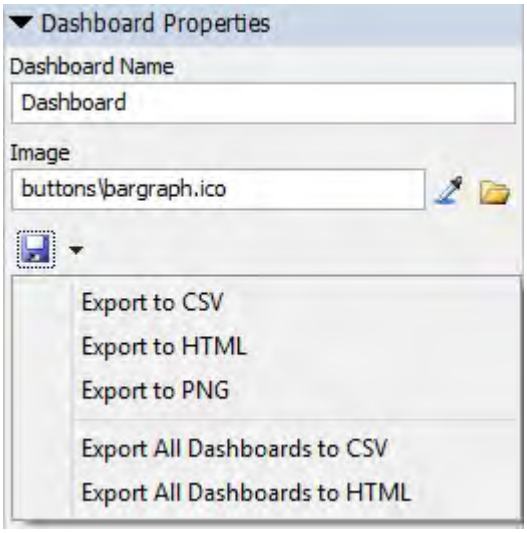
Arrow Keys - Arrow keys may be used to moved Dashboard widgets. Select the desired widget(s) then use the up, down, left or right arrow keys to move the widget(s) 1 pixel. Hold the shift key while pressing the arrow keys to move the widget(s) 5 pixels at a time.

Copy and Paste - Select a single or multiple model input widgets and press Ctrl + C to copy them. Press Ctrl + V to paste into the same Dashboard, another Dashboard in the same model, or Dashboards in other instances of FlexSim. This only works for Model Input widgets, not for Statistics Widgets.

Locking to an Edge

Widgets can be "locked" to the right edge of the dashboard window by right clicking on the widget and selecting "Lock to Right," and can be locked to the bottom edge of the dashboard by selecting "Lock to Full Height." Sides that are locked will show yellow sizers. If a widget is locked, that widget will resize as the window is resized in that direction. Moving the widget will break any locks on it, as well as right clicking and selecting "Remove Locks." Right clicking on the dashboard and selecting "Remove Locks" will remove all locks any widgets have on the dashboard.

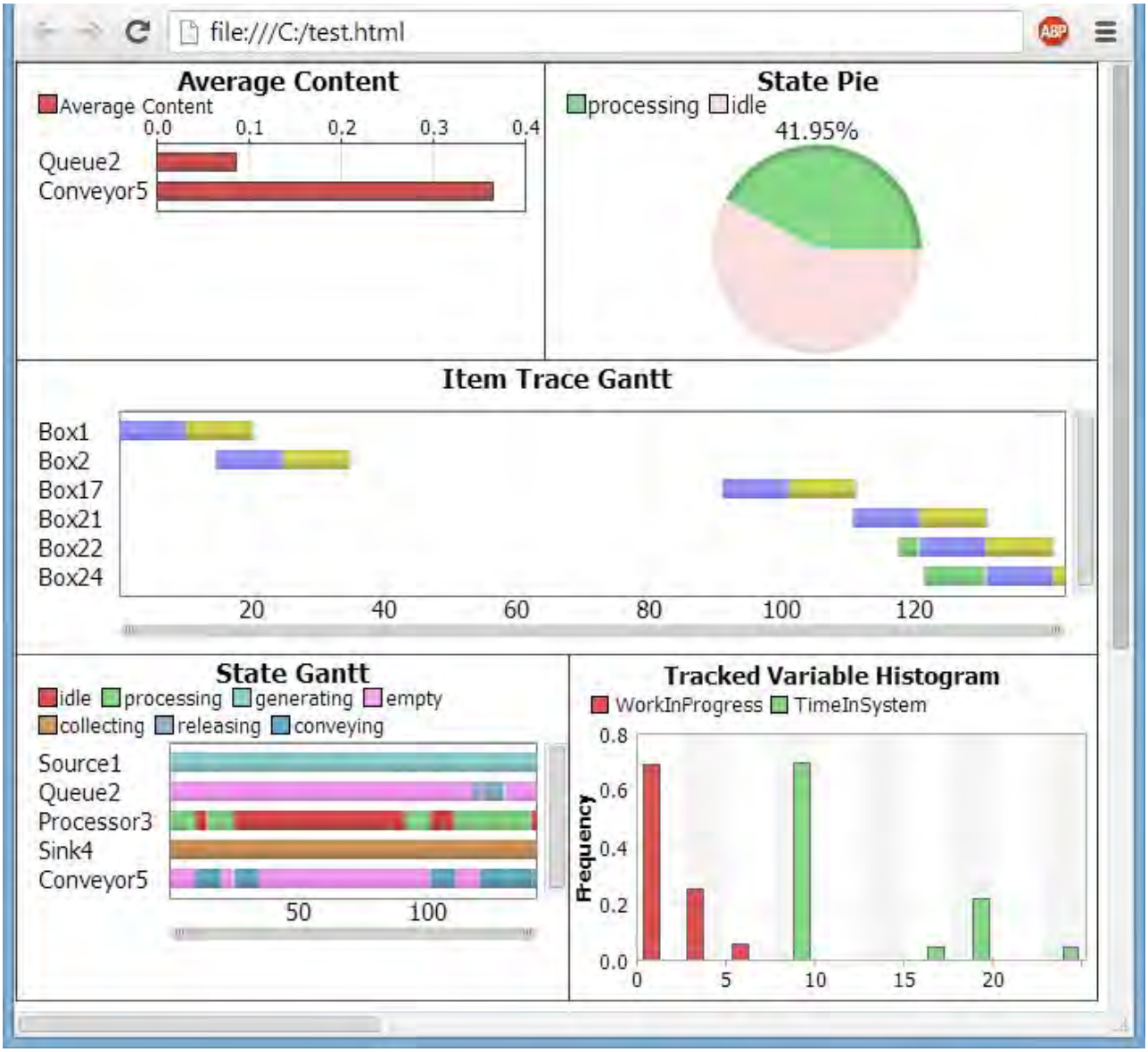
Exporting Dashboards



 -

Exports dashboard to CSV, HTML or PNG formats. Saving CSV files will only save the Dashboard Data associated with statistic objects. Saving to HTML will save the Dashboard with all of it's datat maintaing the layout to an HTML file. In this file you can move, resize and explore the widgets (see image below). Choosing Export All Dashboards to HTML will create one HTML file with all dashboard widgets contained in it. These HTML files are completely self contained and may be opened on any computer with a web browser (though some web browsers may not be supported). After resizing and arranging widgets in the HTML view, the file may be saved again to preserve the layout.

Individual widgets may be exported to these formats as well by right-clicking on the widget.



Exporting Model Inputs

Model Input objects, like buttons and fields, will be exported as read-only objects when exporting to HTML. GUI Class model inputs will not be exported at all except when saving to PNG.

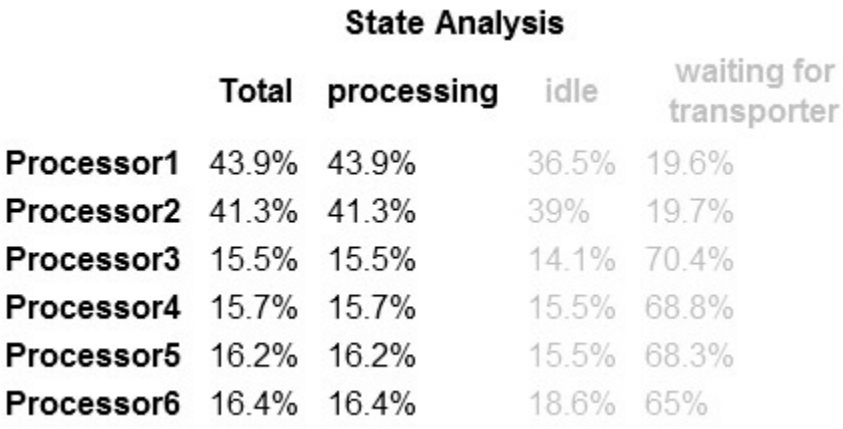
Exporting Images as HTML

The Static Text and Button model input objects can be set to display images. These images are not contained in the exported file, but are referenced based upon the path in the Image box of Quick Properties. If the path is relative to the saved model, exporting the HTML to the model directory, or moving those images into the same directory as the HTML file will allow them to be display. If the Image box contains an absolute path (starting with C:\ or D:\ etc) the image must be in that location in order for the HTML to display it.

Graph Types

Dashboard Graph Types: The list of Dashboard Graphs displayed in the Library Icon Grid are just a starting point, a list of presets. Graphs may be customized to display additional information by editing the Graph's properties.

Text Panel

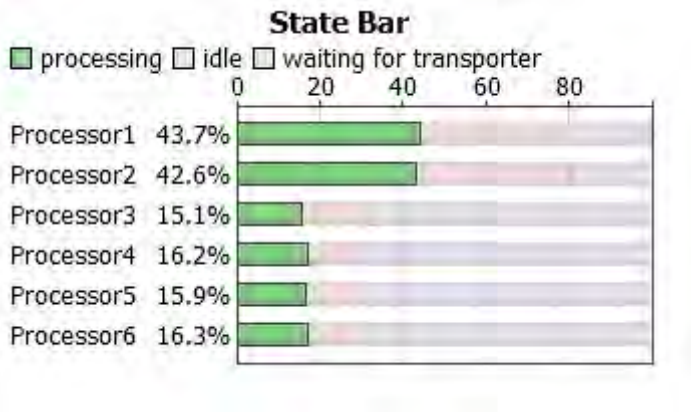


State Analysis

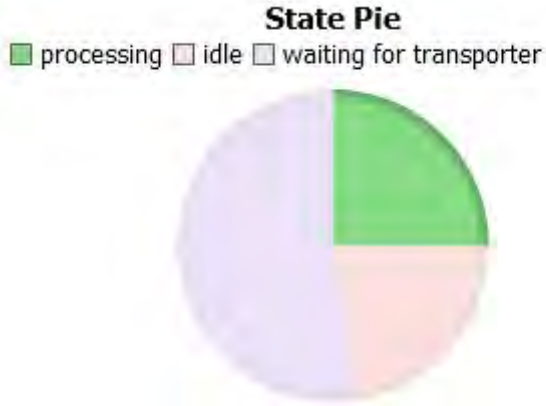
	Total	processing	idle	waiting for transporter
Processor1	43.9%	43.9%	36.5%	19.6%
Processor2	41.3%	41.3%	39%	19.7%
Processor3	15.5%	15.5%	14.1%	70.4%
Processor4	15.7%	15.7%	15.5%	68.8%
Processor5	16.2%	16.2%	15.5%	68.3%
Processor6	16.4%	16.4%	18.6%	65%

Displays text data in a table format. This panel can be used to display state values, statistics and custom data.

Bar Chart

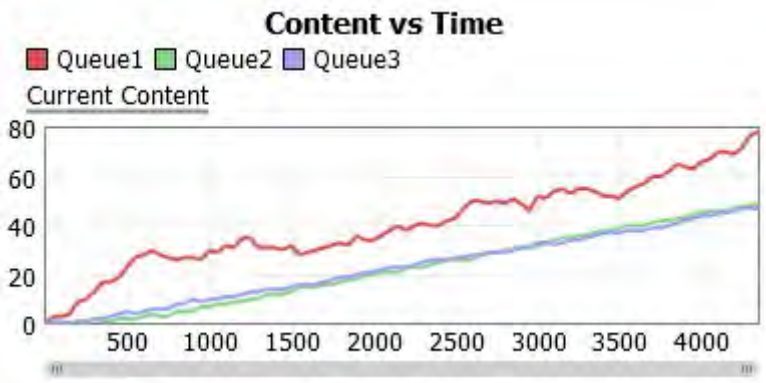


Pie Chart



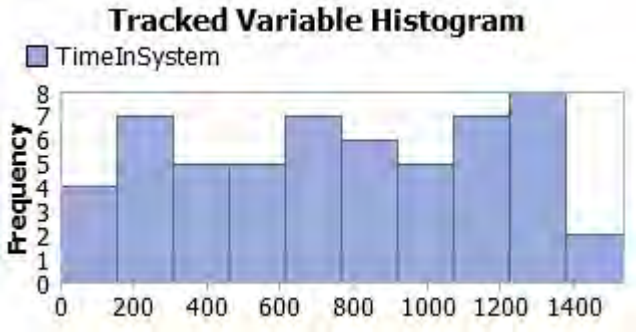
Pie Charts are only available when recording object State data.

Line Graph



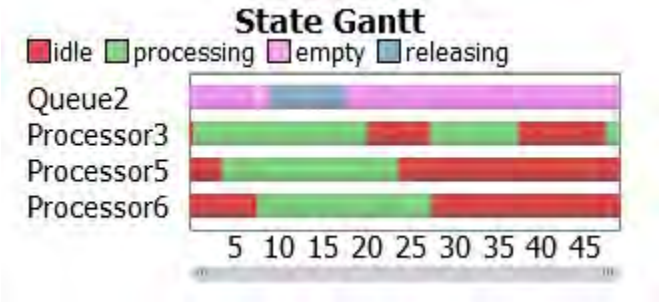
The Line Graph displays data over time. Specify a start time and a time interval between updates.

Histogram



The Histogram is only available for Tracked Variables.

Gantt Chart



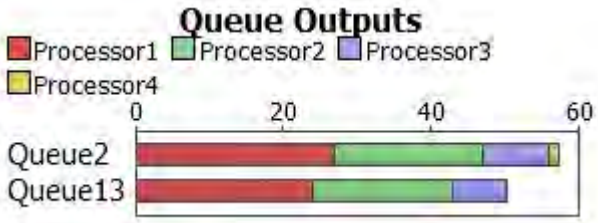
There are two types of Gantt Charts:

- State Gantt Chart (displayed above) - The State Gantt chart shows a time graph of what times and object was in a specific state. State changes in 0 time are not recorded.
- Item Trace Gantt Chart - shows what time an item is created and which objects the item travelled through and at what times. Item Trace charts record data even when an item is in an object for 0 time, however, that data is not graphically displayed on the chart. Export the chart's data to CSV to see the full history.

Financial Analysis

The Financial Analysis graph allows you to specify financial values for objects and flowitems in your model. Individual objects and groups may be added to this graph any number of times. Positive and negative values may be defined for each value. Negative values will be displayed in red surrounded by parentheses.

Custom Chart



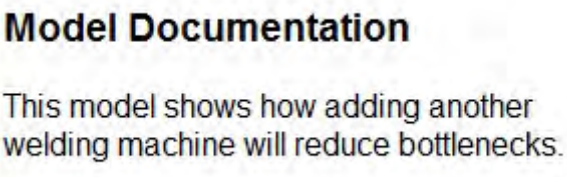
The Custom Chart allows you to graph any kind of numeric data in either a table of values, bar chart, or line graph. Rather than adding only objects to this chart, you can add objects, nodes, tables, global variables, bundles etc. For more information, see the Custom Chart page.

Date and Time Display

The Date and Time Display graph can display the model's current run date and time in multiple formats:

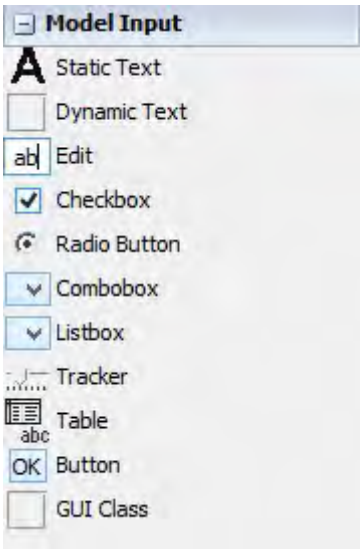
- No Format - Displays the current model time beginning from Day 1 at the model start time (as defined in the Model Settings window).
- Use Default Format - Displays the current model time using the format defined in the Model Settings window.
- Custom Format - This option allows you to define a custom format for the date (displayed above).

Model Documentation



The Model Documentation graph is a custom HTML widget that gives you a blank canvas to add in any custom HTML or flexscript code. Adding flexscript code, you can display any information you want using the pd(), pf(), pt() and pr() commands. These commands will create HTML code to display on the graph.

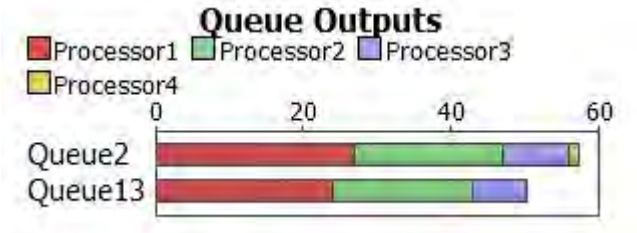
Model Input



Previously, custom user interfaces to control or view variables and parameters from the model was only available through Graphical User Interfaces. Dashboards now have the capability of handling a lot of the same interface requirements as GUIs.

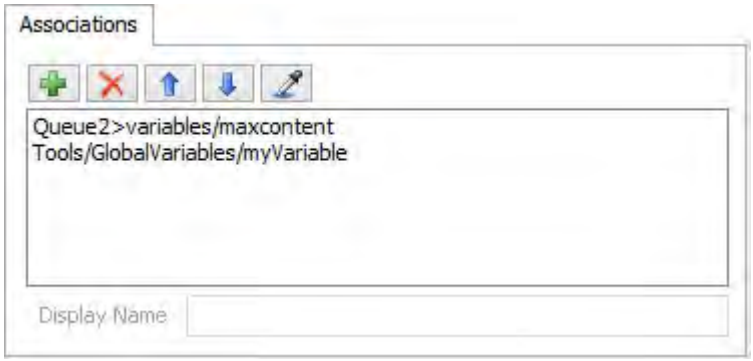
For more information on each ModelInput widget, see the Model Input page.

Custom Chart



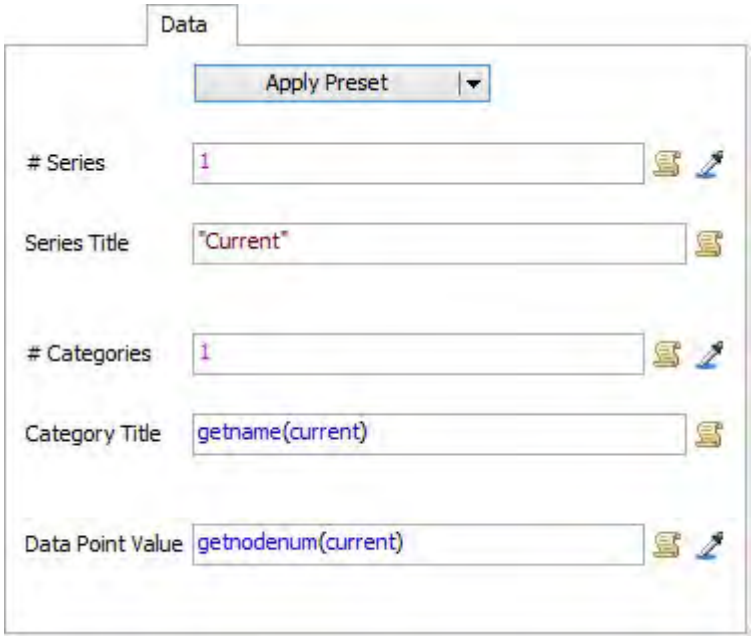
The Custom Chart allows you to display data in the dashboard as a table of values, bar chart or line graph. Custom Charts are not limited to objects. Nodes or global variables may be added to the Custom Chart. Code is then executed to specify what data will appear in the chart. This allows the flexibility of displaying just about anything in the Dashboard.

Associations



Custom charts are tied to associations, rather than restricted to only objects as with other Dashboard Widgets. An association can be a node in the tree, a global variable, an object, global table, label table, etc. Associations can be added to the custom chart any number of times, however, the first association is what will be used to specify the number of series to display (see # Series below).

Displaying Data



There are 5 picklists that allow you to specify how many series and how many categories are displayed in the chart. Think of the series as columns in a table, and the categories as rows in a table. All picklists are fired each time the graph is drawn, or for a time graph, each update time. This allows the number of series and categories to be dynamically changed, ie a global table that has columns or rows added to it. A list of presets is available for displaying current values, tables, etc.

Global Variables: Global Variables that are added to the custom chart to be accessed as an association must be either an integer or double type. When using the # Series, # Categories or Data Point Value picklists with global variables, use getnodenum(current) to get the current value of the global variable. This allows you to tie to multiple global variables and other nodes and display all of their data in the same chart. Any other reference to current will give you the global variable's node, ie /Tools/GlobalVariables/myVariable.

Other global variables may be used directly, however, the special functionality of using getnodenum(current) will only return valid data for integer and double types.

Series

The number of series is only calculated once for all associations. This picklist is the first to be fired and passes in the first association object/node. The value returned should be an integer. Access Variables

- current: the first association object/node in the list.

Series Title

This picklist is fired for each series. These titles will display in the charts legend. The value returned should be a string.

Access Variables

- current: the first association object/node in the list.
- seriesnum: the index of the series.

Categories

The number of categories is calculated for each association in the list. Each category will be a row in the chart, beginning with the first association, in order, down to the last. The value returned should be an integer.

Access Variables

- current: the association object/node.

Category Title

This picklist is fired for each category, for each association. If two or more categories who are sequential (next to each other) have the same title, those categories will become a group. By default, a group's total values are calculated as an average of all of its members. To calculate the total as a sum, the title must have a #sum at the end. ie, *"Total#sum"*. When using groups, the default titles displayed for the group members will be their path in the model. To change this to a custom display, set a Display Name for the association in the Associations tab. The value returned should be a string. Access Variables

- current: the association object/node.
- categorynum: the index of the category for the association.

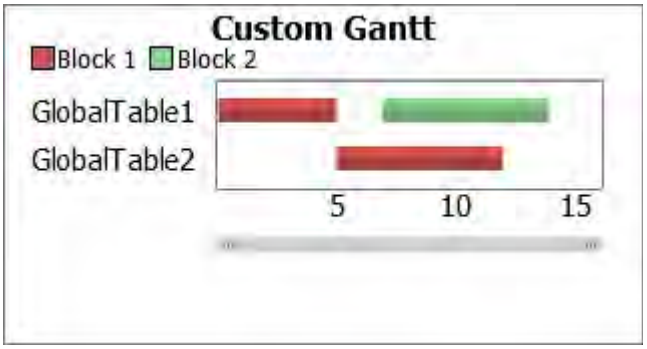
Data Point Value

This picklist is fired for each association, for each category, for each series. The value returned should be a number.

Access Variables

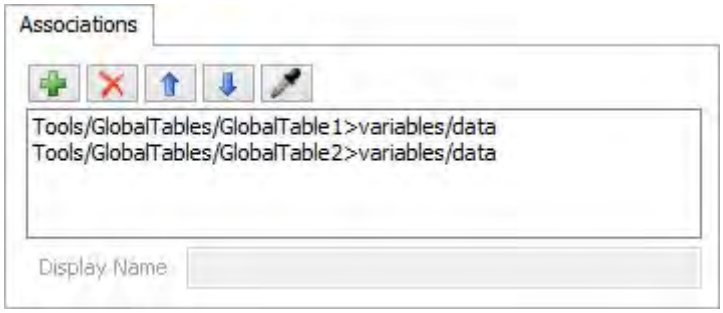
- current: the association object/node.
- seriesnum: the index of the series.
- categorynum: the index of the category for the association.

Custom Gantt Chart



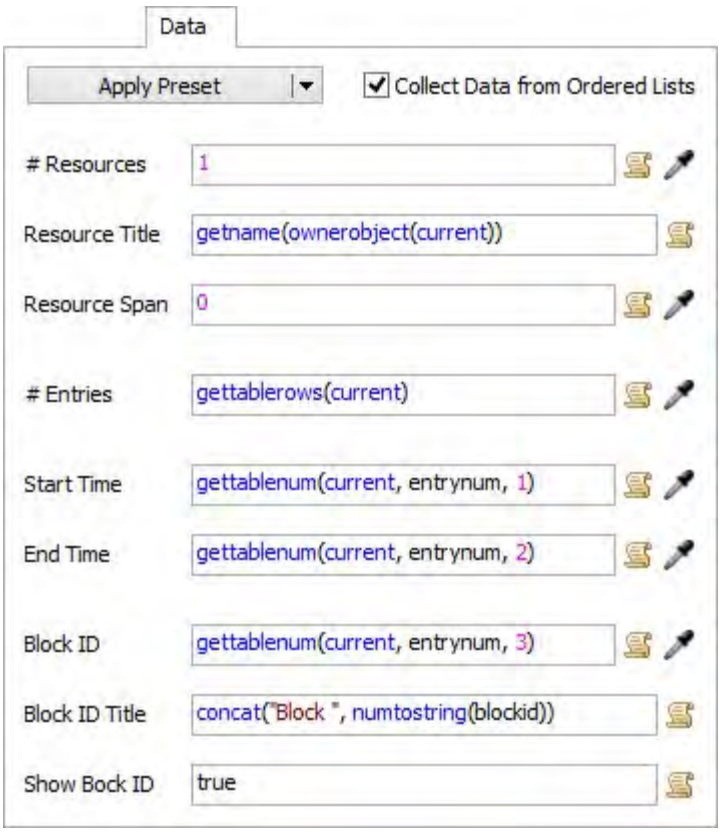
Like the Custom Chart, the Custom Gantt Chart allows you to display several forms of data in the dashboard as a gantt chart. Nodes or global variables may be added to the Custom Chart. Code is then executed to specify what data will appear in the chart..

Associations



Custom Gantt Charts are tied to associations, rather than restricted to only objects as with other Dashboard Widgets. An association can be a node in the tree, an object, global table, label table, etc. Associations can be added to the custom chart any number of times.

Displaying Data



There are 9 picklists that allow you to specify how many resources and how many entries are displayed in the chart. Resources will be the rows on the left and entries will be blocks on that row. All picklists are fired each time the graph is drawn. This allows the number of series and categories to be dynamically changed, ie a global table that has columns or rows added to it. A list of presets is available for displaying table and bundle values.

Resources

The number of resources for each association. This will be evaluated once per association. The value returned should be an integer. Access Variables

- current: the association object/node.

Resource Title

This picklist is fired for each resource. These titles will display as the row titles. The value returned should be a string.

Access Variables

- current: the association object/node.
- resourcenum: the index of the resource.

Resource Span

This picklist is fired for each resource. The value returned should be a 1 or 0. Normally you should return 0. Returning a 1 will make that resoure display its entries as blocks that are drawn taller and lighter and on the

same row as the next resource behind its entries. This allows you to display two types of information in the same row.

Access Variables

- current: the association object/node.
- resourcenum: the index of the resource.

Entries

The number of entries is calculated for each resource. The value returned should be an integer.

Access Variables

- current: the association object/node.
- resourcenum: the index of the resource.

Start Time

The start time is calculated for every entry of every resource. The value returned should be a number.

Access Variables

- current: the association object/node.
- resourcenum: the index of the resource.
- entrynum: the index of the entry.

End Time

The end time is calculated for every entry of every resource. The value returned should be a number.

Access Variables

- current: the association object/node.
- resourcenum: the index of the resource.
- entrynum: the index of the entry.

Block ID

The block ID is calculated for every entry of every resource. Each unique block ID will be turned into a title in the legend and will be represented with a different color. The value returned should be an integer.

Access Variables

- current: the association object/node.
- resourcenum: the index of the resource.
- entrynum: the index of the entry.

Block ID Title

The block ID title is evaluated for every block ID. These titles will display in the charts legend. The value returned should be a string.

Access Variables

- current: the association object/node.
- blockid: the index of the block ID.

Show Block ID

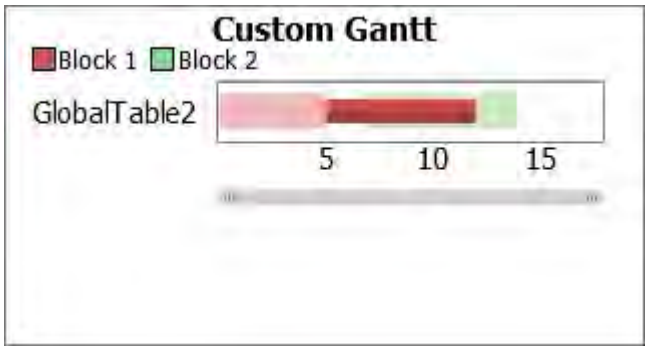
This is evaluated for every block ID to determine if the block ID should be shown The value returned should be a boolean. Access Variables

- current: the association object/node.
- blockid: the index of the block ID.

Example

	Start Time	End Time	Block ID
Row 1	0.00	5.00	1.00
Row 2	7.00	14.00	2.00

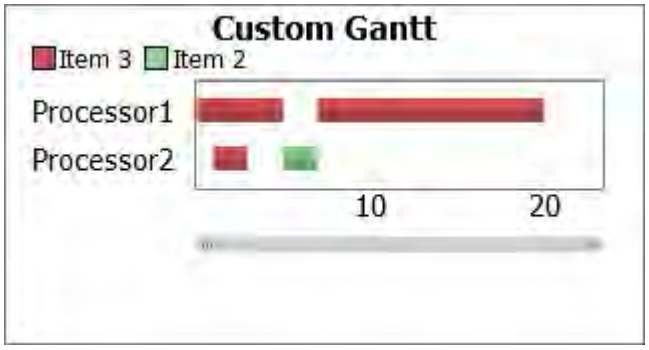
The example Custom Gantt Chart at the beginning of this section was made by pointing at this global table and one other and using the settings shown in the Data Tab image above.



This chart was made using the same settings as above except the Resource Span was set to resourcenum == 1.

Collect Data from Ordered Lists

Unchecking this box allows you to display data from a table that has unordered data.

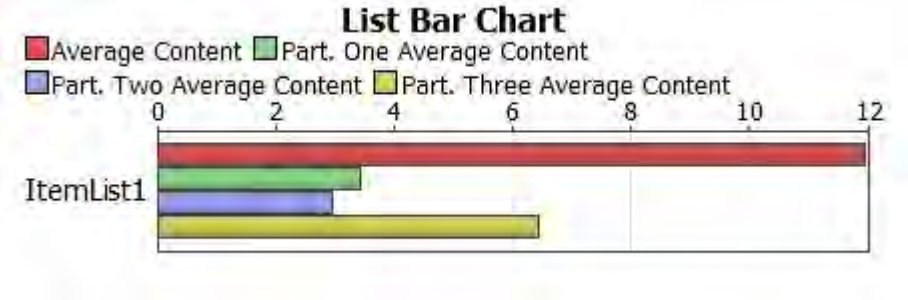


The chart above was made by pointing at the table below using the default preset settings for table values.

	Resource Title	Block Title	Start Time	End Time
Row 1	Processor1	Item 3	0.00	5.00
Row 2	Processor2	Item 3	1.00	3.00
Row 3	Processor1	Item 3	7.00	20.00
Row 4	Processor2	Item 2	5.00	7.00

The main difference here is that the resource and block titles will be determined by the data the chart is associated with. In the above example the resource titles come from the first column and the block titles come from the second. Each unique name will be its own resource or block and values with the same resource and block names will be grouped together.

List Chart



The List Chart allows you to display data related to lists. The List Chart is capable of displaying information from any FlexSim object but it has some extra options that allow you to track backorders, entries, and partitions that you would not be able to chart otherwise. The List Chart is very similar to other charts except for a few differences, which will be discussed in more detail in the following sections.

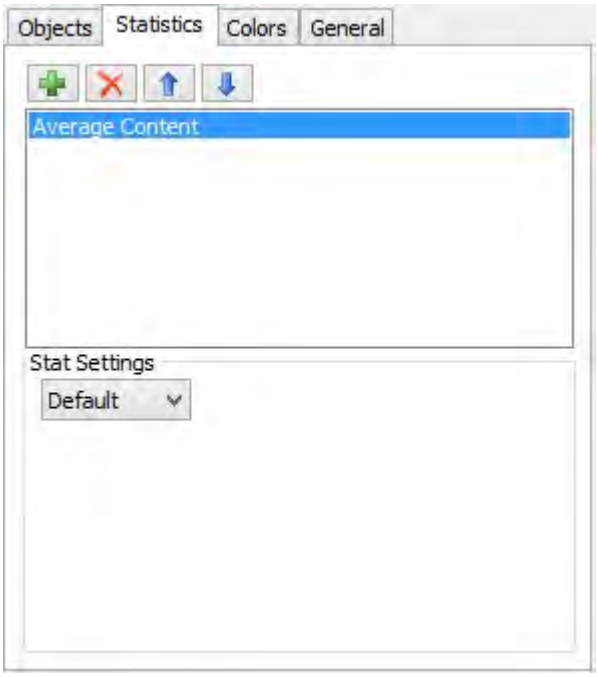
Objects

The List Chart allows you to select Lists, Tracked Variables, or standard FlexSim objects as linked objects for the chart data. This means that you can compare a list's statistics to any other object if you would like. Adding, removing, sorting, grouping and ungrouping objects works the same way as with any of the standard charts. See Dashboard Example for more information on this process.

Combining a List with other FlexSim objects may cause some interesting combinations but any standard FlexSim object will show a zero for invalid statistics for example the backorder statistics of a processor will always be zero.

Statistics

The Statistics tab of the List Chart is unique from other dashboard chart properties because it has an extra Stat Settings panel which is where you define advanced statistic settings for the list. These settings control what data is pulled from the list such as backorder or partition data.

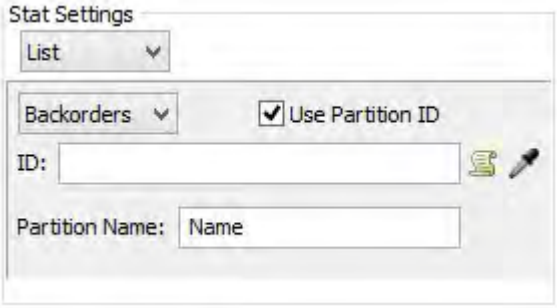


Default

When you first add a statistic to the chart it will be set to default Stat Settings. This means that it will retrieve the basic information of that statistic from the objects. For example, Average Content with the default setting will retrieve the average number of entries in the entire list regardless of partitions and ignores backorders.

List

The List option must be selected for the Entries/Backorders drop down box and the Use Partition ID checkbox to appear.



Entries/Backorders

This drop down menu sets whether the chart will display Backorder or Entry data from the list. For example if Backorder is selected for Average Wait Time then the chart will only display the Average Wait Time for Backorders.

Use Partition ID

This checkbox indicates whether or not you would like to specify a specific partition to retrieve the data from. If this is checked then the ID and Partition Name edit boxes will be shown.

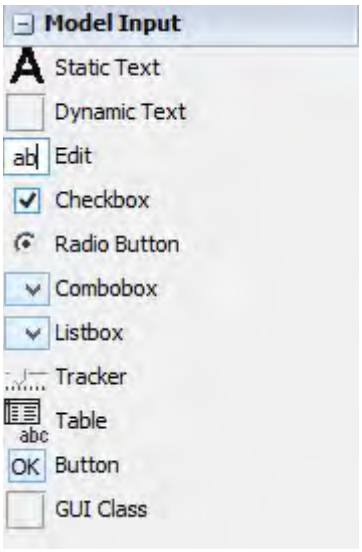
ID

This code edit box is where you enter the ID of the partition which you would like to retrieve the data from. This can execute any flexscript but just needs to return the ID value. For example if the partition ID of a list is set as the treenode of the object which pushed an item onto the list then the code node("Queue2",model()) could be used to chart the data from the partition which holds the items from Queue2.

Partition Name

This field is used to help name duplicate statistics. The name must be unique among other statistics of the same type of data this means that if you have three different Average Content statistics each one needs a unique name otherwise errors will be shown. This name will be added onto the front of the statistic name so if you name an Average Content statistic "One" then the name that will be displayed on the chart will be One Average Content.

Model Input



Model Input controls allow you to have interaction between the user and the model without the need for making separate Graphical User Interfaces. Each control has customizable properties that can be set through the Quick Properties window. Each control can have its own ID string that can be used to reference the control using getdashboardcontrol("idName").

Static Text

The Static Text control can be used to either display text, or to display an image.

Dynamic Text

The Dynamic Text control can be linked to a variable, label or node in the model and will display its current value. This can include number data or string data. Dynamic Text is not editable and is for display only.

Edit

The Edit control is similar to the Dynamic Text control but it can also be edited. Changing the value of the Edit will also change the value of the node in the model that it is linked to. ie, linking to the Max Content of a Queue would allow the user to change the Max Content without editing the Queue's parameters directly.

Checkbox

The Checkbox control can be linked to a variable or node in the model. The node will be 'evaluated' as a boolean (true or false, 1 or 0), so linking to a non-number node will have no affect. You can also add code to the Checkbox's OnPress trigger.

Spinner

The Spinner control can be linked to a variable or node in the model with numeric data. Clicking on the up or down values in the spinner will increment or decrement that numeric data. By holding down the button and moving your mouse, you can drag the value up or down.

Radio Button

The Radio Button control can be linked to a node in the model and it can be linked to other radio buttons. To attach to a group of radio buttons, click on the Link and then click on another radio button in the same Dashboard. This will "Attach" the radio button to the group. Any number of radio buttons may be added to a group. Each radio button in the group has a designated value. If the parent radio button is linked to a node in the model, that node will be set to the value of the selected radio button. You can also add code to the Radio Button's OnPress trigger.

Combobox

The Combobox control can be linked to a variable, label or node with number data. Any number of options can be added to the drop down list. You can also add code to the Combobox's OnSelect trigger.

Listbox

The Listbox control can be linked to a label or node with number or string data in the model. If the node is string data then the name of the listbox item will be set to the node. You can also add code to the Listbox's OnSelect trigger.

Tracker

The Tracker control can be linked to a label or node with number data in the model. Tracker's have a set of additional parameters that can be set including the Minimum, Maximum and Exponential values as well as the Style and if it's a vertical or horizontal tracker. You can also add code to the Tracker's OnDrag trigger.

Table

The Table control can be linked to any table in the model. The table will display the linked table's current values. Editing the table will update the linked values directly.

Button

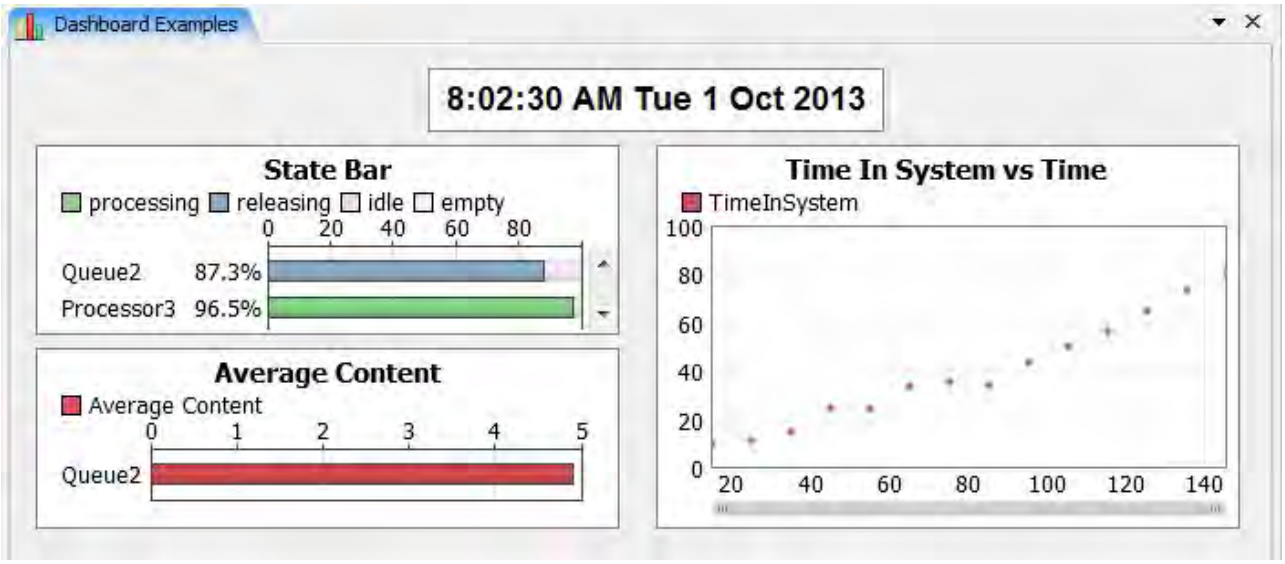
The Button control can be linked to a node in the model. The button can display a text title, or it can display an image. If linked to a node, the button will display the current value of the node. You can also add code to the Button's OnPress trigger.

GUI Class

For more advanced functionality, the GUI Class object allows you to build a GUI using the GUI Builder and then point the GUI class to the GUI. Changes made in the GUI Builder will be updated in the Dashboard when the widget is refreshed. You can refresh the widget either by closing and reopening the Dashboard, or by right clicking on the widget and selecting *Refresh*.

Using paths with the GUI Class - If your GUI created in the GUI Builder and then referenced with a GUI Class widget has nodes being referenced using the @ symbol (ie @>objectfocus+>variables/maxcontent), the returned paths will not be correct in the Dashboard. The @ symbol moves up the tree until it finds the top most object so it will move up through the Dashboard and the other panels in the view.

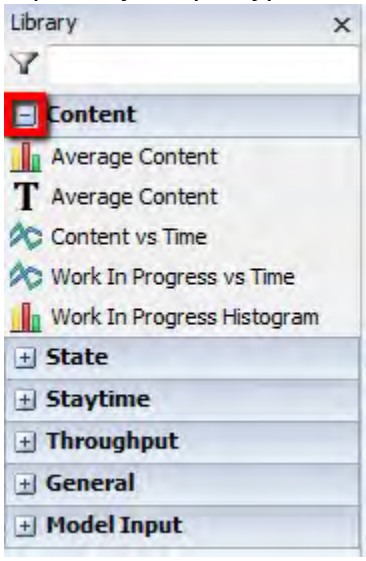
Dashboard Example



Adding Graphs

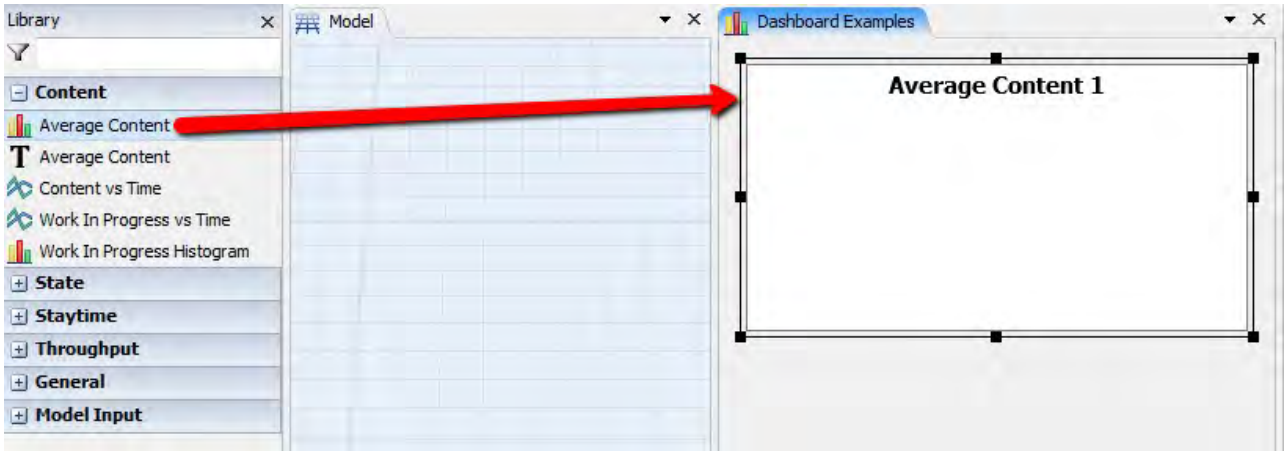
1. Add a Dashboard to the model by selecting Add > Statistics > Dashboard from the Toolbox. 2. If you already have a Dashboard displayed, make sure the Dashboard tab is selected

3. If it is not already expanded by default, expand the needed Graph Type from the Library. You can also collapse any Graph Types that are not needed.



4. Choose a Statistic

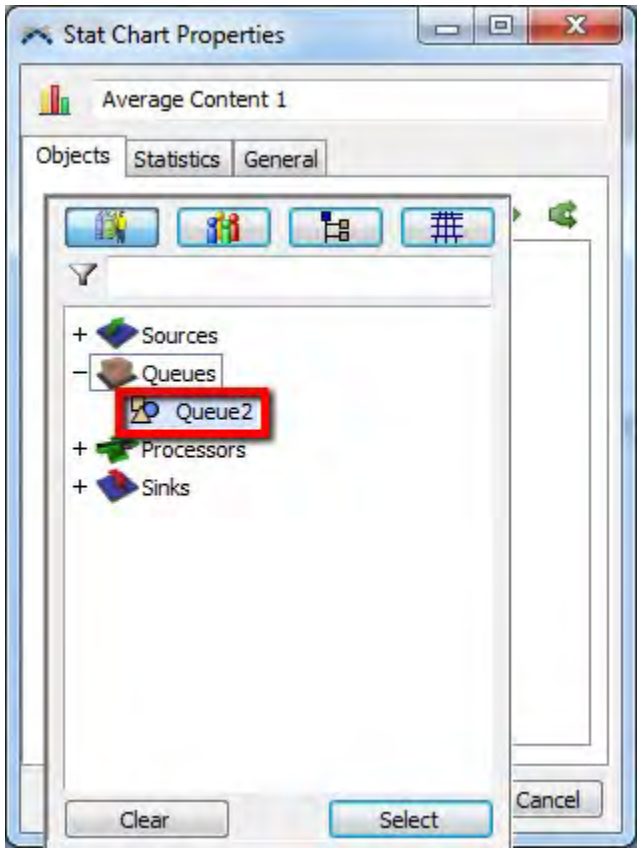
   Either click and drag one of the options from the menu to the Dashboard or select one of the options and click anywhere in the Dashboard to display the graph.



Model Edit Modes: If you clicked once on a graph in the Library Icon Grid, you will enter a Model Edit Mode. This allows you to create multiple graphs by clicking in the Dashboard view. To get out of Model Edit Mode, right click the view or press the Escape key.

5. Add Objects

The graph properties menu should automatically appear. Click on the ✚and select which objects to include with this graph. You may select multiple objects through this window if desired. Alternatively, click on the 🖉to enter "Sample" mode, then click on an object in the 3D view to add it to your graph.
Click on the Select button, then the OK button.
You can double click on the graphs to reopen their properties or right-click the view and select Properties.



6. Reset and Run the Model

The new graph will be blank until the model runs.

Dashboard Reference

Working With Graphs

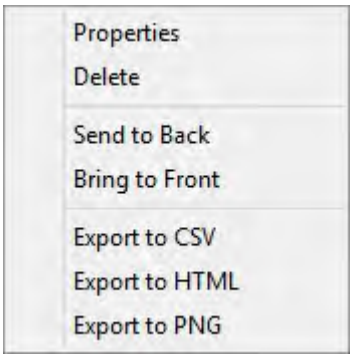
Editing - Double-click the graph to open its properties window. The type of properties window that opens will depend on the type of graph that is being selected. For more information, see the Graph Properties.

Moving - Click the graph, then drag it by its black edges.

Resizing - Click the graph, then drag any of the black edge sizers.

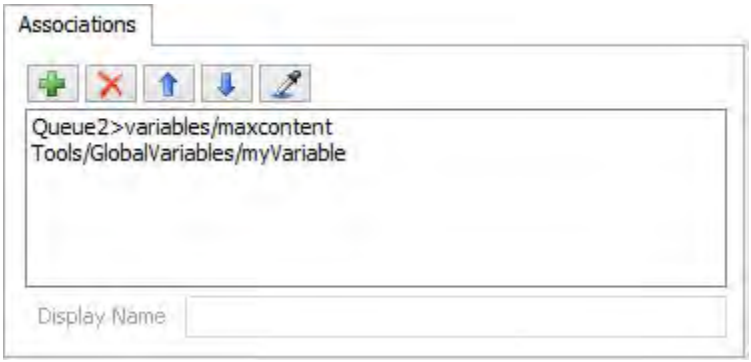
Auto-resizing - Dragging a graph sizer to the edge of the dashboard window will cause that side to be "locked" to the edge. Locking to the right or bottom edge of the window will cause the graph to resize when the dashboard window is resized.

Context Menu - Right-click the graph to open its context menu.



- Properties - Opens the graph settings window.
- Delete - Deletes the graph from the Dashboard.
- Send to Back - Sends the selected graph to the back.
- Bring to Front - Brings the selected graph to the front.
- Export to CSV - Allows you to save the graph's current data set as a .csv file.
- Export to HTML - Allows you to save an self contained HTML file of the graph's current data.
- Export to PNG - Allows you to save a .png image file of the graph's current display.

Associations Page



Associations are objects, nodes, or variables in your model.

 - Adds associations to the current list of associations. See the Adding Associations section for more info.

 - Removes the selected associations from the list.

 - Moves the selected associations up in the list

 - Moves the selected associations down in the list

 - Sample a node or object in the model.

Display Name - When using groups, if no display name is specified, the titles displayed for the group members will be their path in the model.

Adding Associations

There are multiple ways to add associations to the custom chart.

Tree Browse Dialog

For more information on the tree browse dialog, see the tree browse dialog page.

Global Variables / Tables

The list of Global Variables and Global Tables will be populated based upon what is available in your model. The list of global variables will only include integer and double types.

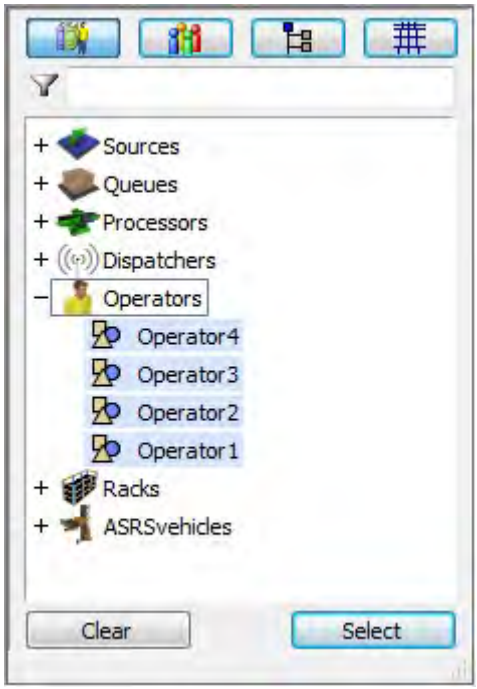
Sampler

For more information on using the Sampler to select objects and nodes, see the Sampler page.

Object Selection Window

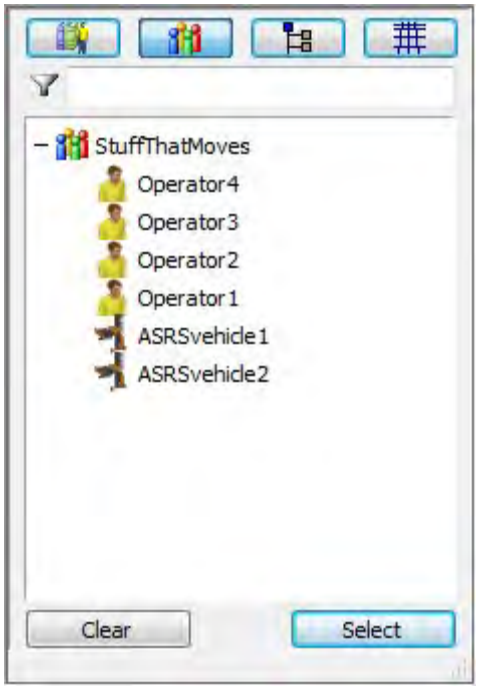
The current mode is highlighted at the top of the browsing window.

Browse by Class

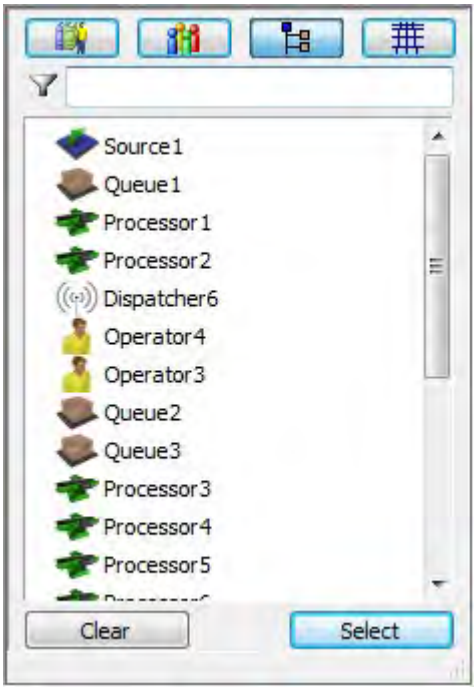


This method of adding objects sorts the objects by class. To select an entire class, click on the type icon. Click on an object to select or deselect it.

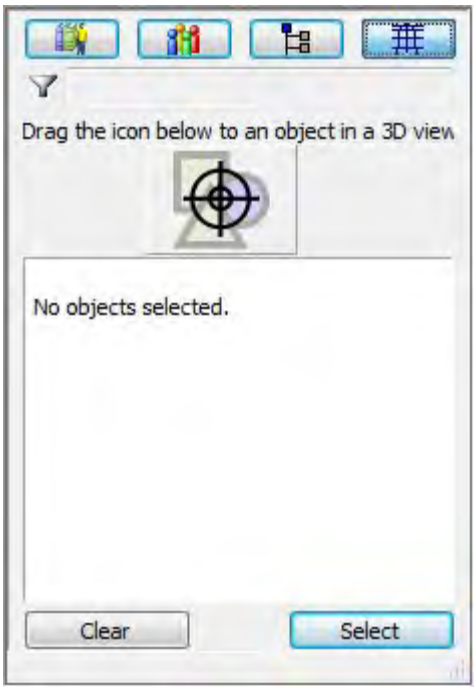
Browse by Group



This method of adding objects sorts the objects by group. To select an entire group, click on the type icon. Click on an object to select or deselect it. Browse by Object



This method of adding objects lists all objects in the model. Click on an object to select or deselect it. Select by Dragging



This method of adding objects uses the 3D model view. Simply drag the target icon from the current window to an object in the model you would like to add.

The colors page allows you to define colors for your graph. The options available are based upon the graph. For State charts, each State has a color. Notice in the above image there are two colors displayed. The color on the left is the normal color for that state. The color on the right is the translucent color used when *Show Yellow Checked States as Translucent* is checked from the Utilization Analysis page. For statistic graphs (Avg. Content, etc), the colors are for each statistic. Line charts define colors for each object or group. Gantt Charts either display colors for States or for objects.

Colors are not available for Text graphs.

Define Colors - Check to define a custom set of colors.

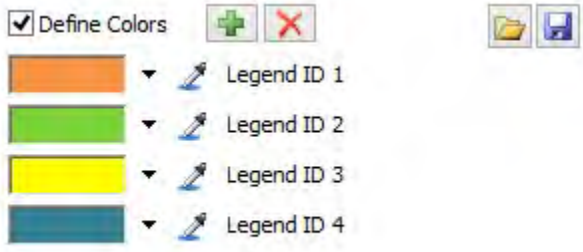
- Loads saved color schemes and allows you to edit color schemes in the tree.

- Saves the current color scheme.

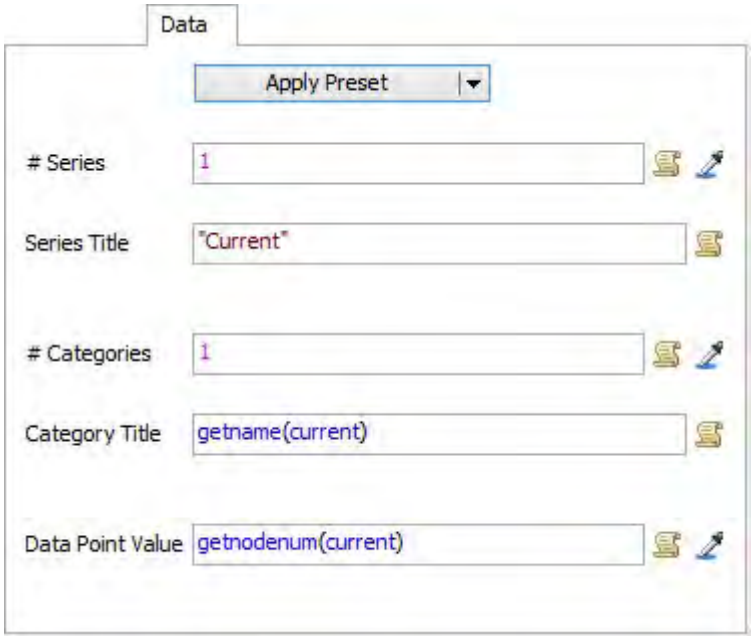
- Displays a color palette.

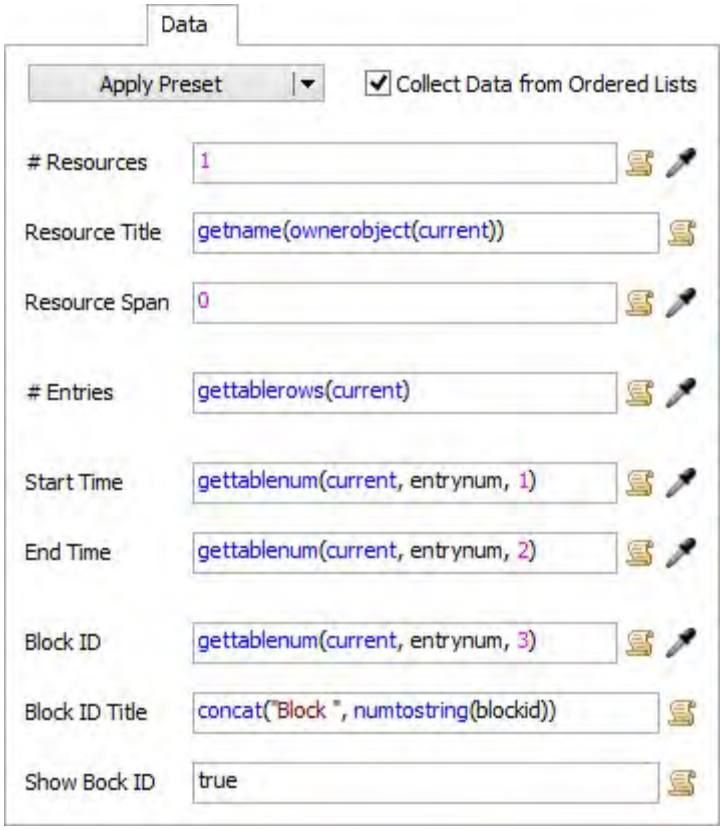
- Allows you to sample a color in FlexSim or outside of FlexSim.

Item Trace Gantt and Custom Chart



The Item Trace Gantt Chart and the Custom Chart dynamically add Legends or Series to their charts so a and are available to add and remove colors.

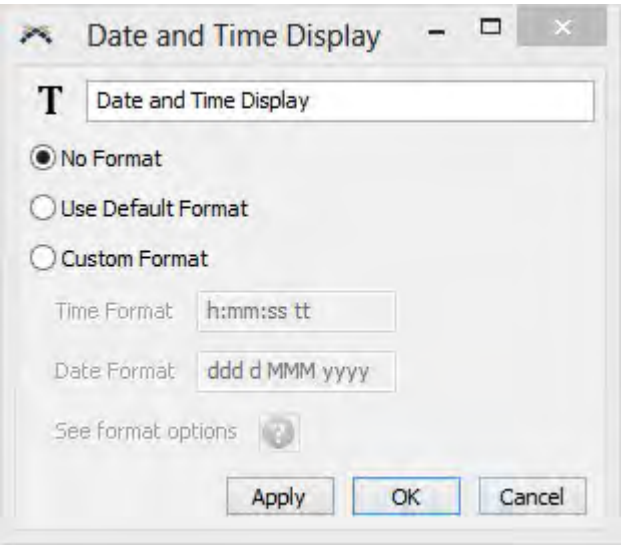




The data tab is used by the Custom Chart and the Custom Gantt Chart.

The Data tab gives you multiple picklists that allow you to customize the data to be displayed on your chart. Apply Preset - This button contains a list of presets to get you started on using the custom chart or custom gantt chart. The presets include code to access values from tables, bundles etc.

See the Custom Chart page and Custom Gantt Chart page for more information on the picklists.

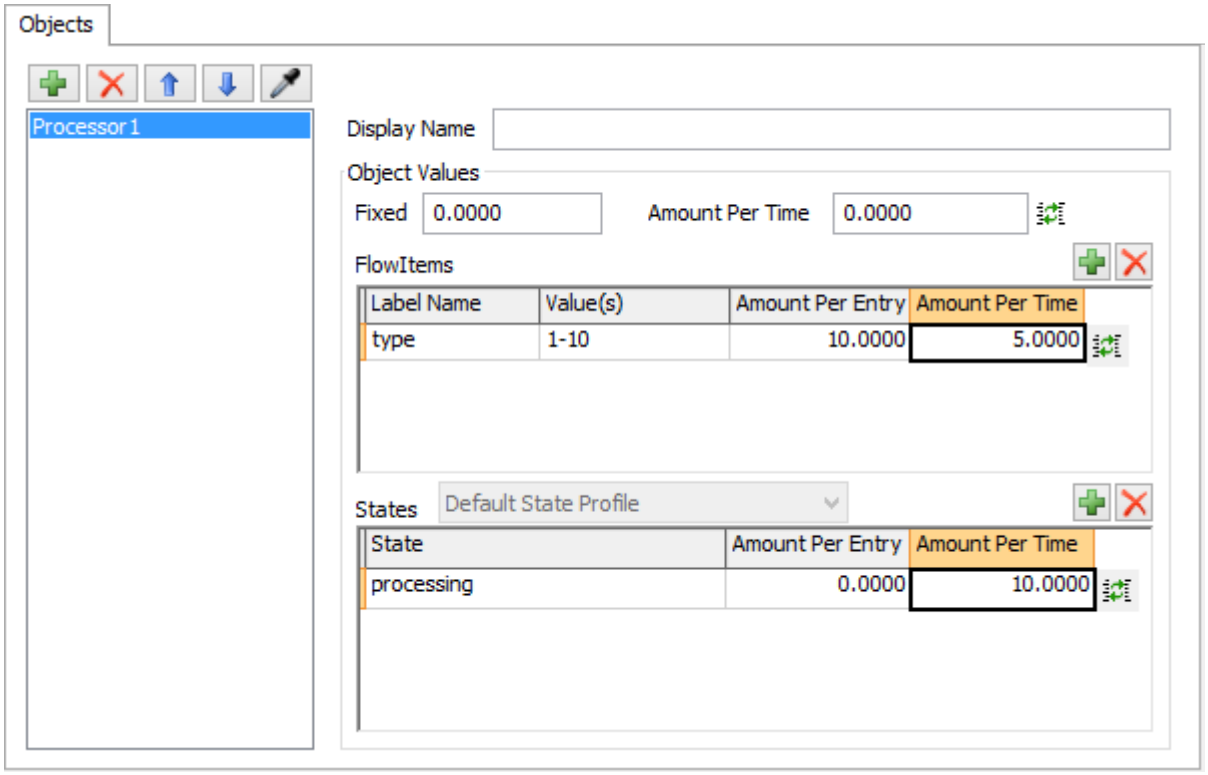


Name Field - Changes the name of the widget to the specified name.

No Format - Displays the current elapsed model time from Day 1 at the time specified in the Model Settings window with the format: Day 1, 08:00:00

Use Default Format - Displays the current model date and time as defined in the Model Settings window.

Custom Format - Displays the current model date and time using a custom format.



- Adds ob jects/groups to the current list of objects. Objects and groups may be added multiple times.

- Removes objects/groups from the current list of objects

- Moves the selected object up in the list

- Moves the selected object down in the list

- Sample an object in the model.

Display Name - Enter text to display on the graph. Leave this field blank to display the object or group name.

Object Values

All object values can be negative or positive numbers. Negative numbers are shown in red and surrounded by parentheses.

Fixed - This value is added once when the model is reset. This value might represent the initial purchase price of a resource.

Amount Per Time - This value is continually added as the model runs. A value of 1.0 would mean at time 50.0 the total time value for the object would be 50.0.

Use the to convert units to model time units.

FlowItems - Use the and to add or remove items from the table. Each item has an Label Name, Value(s), Amount Per Entry and Amount Per Time. To apply the values to all label values, leave the Value(s) field blank. To define multiple label values, separate numbers using commas and dashes. For example: 1,2,5,10-15,20

- Amount Per Entry value is applied when a FlowItem with the specified label value enters the object.

- Amount Per Time value is applied continually while the flowitem with the specified label value is in the object.

Use the to convert units to model time units.

States - Use the and to add or remove items from the table. Each item has a State, Amount Per Entry and Amount Per Time.

- Amount Per Entry value is applied when the object enters the specified state. If an object is in the state for 0 time, the fixed value will NOT be applied.
- Amount Per Time value is applied continually while the object is in the specified state.

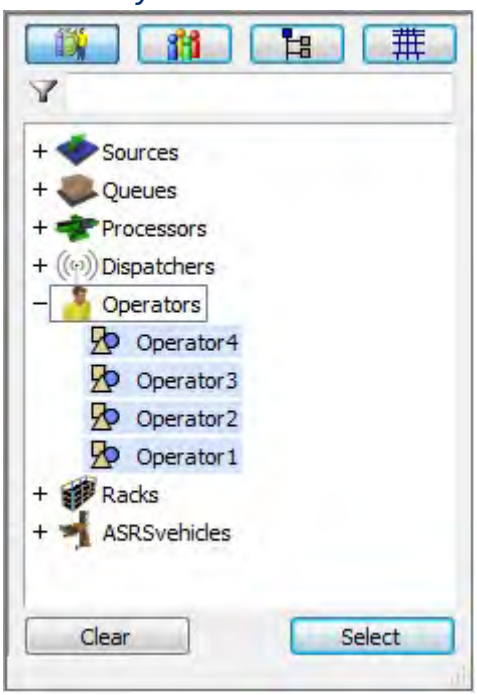
Use the to convert units to model time units.

Adding Objects

There are five ways to add objects to a graph. For more information on using the Sampler to select objects, see the Sampler page.

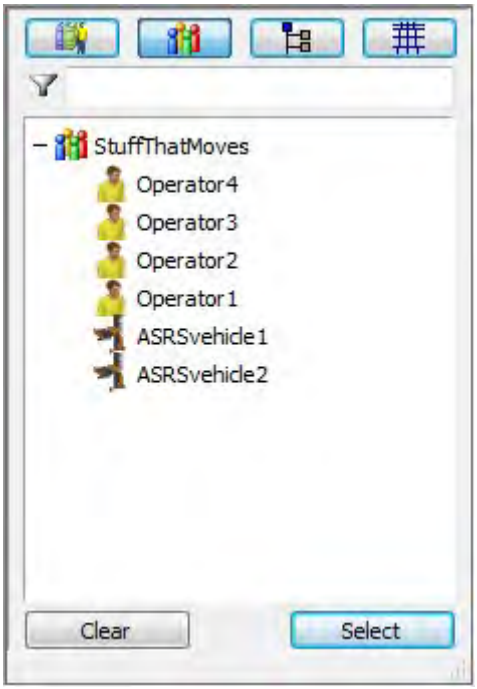
The current mode is highlighted at the top of the browsing window.

Browse by Class



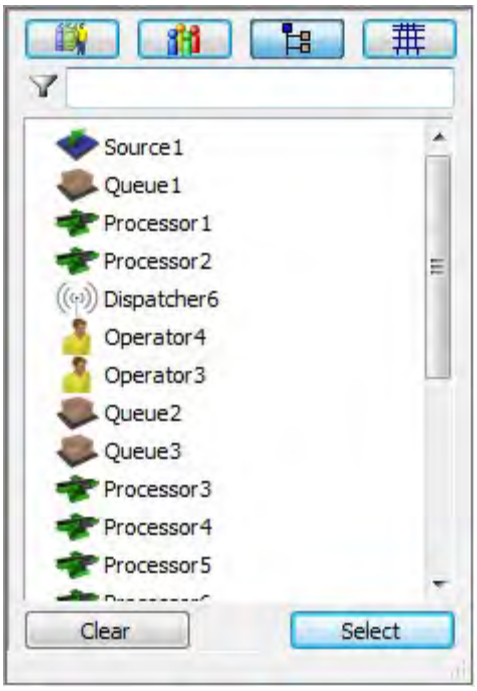
This method of adding objects sorts the objects by class. To select an entire class, click on the type icon. Click on an object to select or deselect it.

Browse by Group



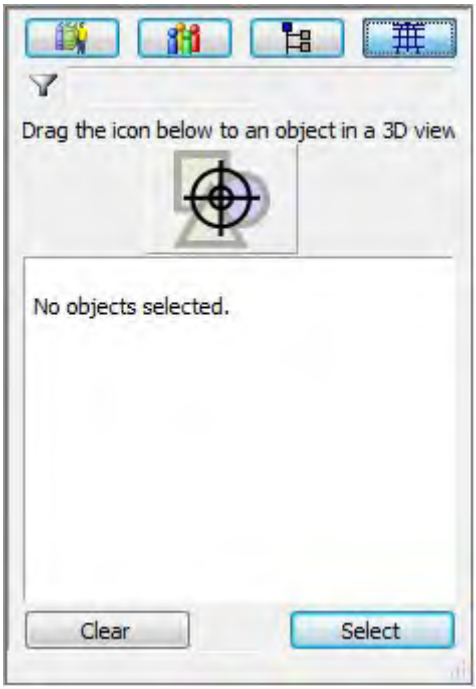
This method of adding objects sorts the objects by group. To select an entire group, click on the type icon. Click on an object to select or deselect it.

Browse by Object



This method of adding objects lists all objects in the model. Click on an object to select or deselect it.

Select by Dragging



This method of adding objects uses the 3D model view. Simply drag the target icon from the current window to an object in the model you would like to add.

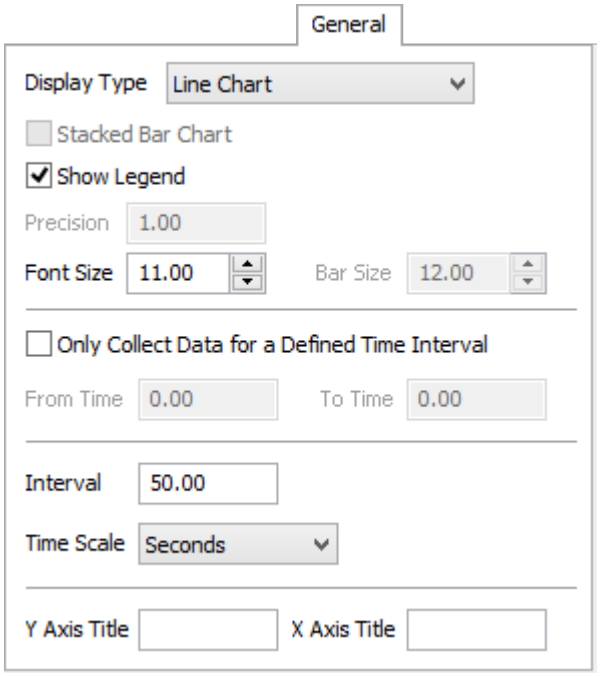
General Pages

The General page of Dashboard graphs vary depending on the statistic object and upon the display type.

Graph Types

- Line Chart
- State Statistic Graphs
- Financial Analysis
- Gantt Chart
- Tracked Variable Graphs

Line Chart



Display Type - This option changes the display style of graph.

Stacked Bar Chart - Bar Chart Only. This option stacks each segment of the data for one object into a single bar.

Show Legend - This option adds a legend to the graph.

Precision - This defines the number precision of graph text. Not available for line graphs.

Font Size - This defines the font size of graph text.

Bar Size - Bar Chart Only. This defines the bar height for Bar Charts.

Only Collect Data for a Defined Time Interval - This option creates a time period during which statistics for this graph will be recorded.

From Time - This defines when statistics will start being recorded.

To Time - This defines when statistics will no longer be recorded.

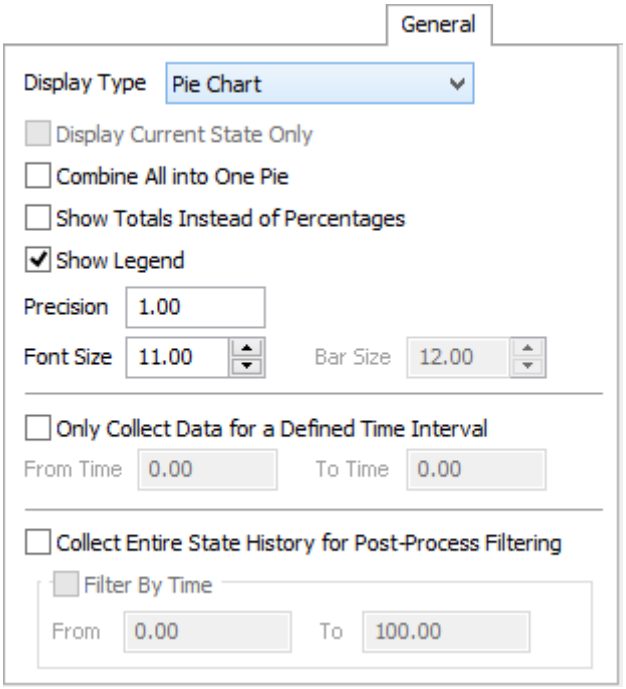
Interval - This defines how often a line graph will update.

Time Scale - This defines the time scale that numbers will be displayed in along the x-axis.

Y Axis Title - Text that will display along the Y Axis.

X Axis Title - Text that will display along the X Axis.

State Statistic Graphs



Display Type - This option changes the display style of graph.

Display Current State Only - This option is only availbe for Table of Values charts. Displays the current state name of objects.

Combine All into One Pie - This option combines all pie charts into a single pie chart. It averages the values from all objects.

Show Totals Instead of Percentages - This option changes the displayed values from percentages to overall totals.

Show Legend - This option adds a legend to the graph.

Precision - This defines the number precision of graph text.

Font Size - This defines the font size of graph text.

Bar Size - Bar Chart Only. This defines the bar height for Bar Charts.

Only Collect Data for a Defined Time Interval - This option creates a time period during which statistics for this graph will be recorded.

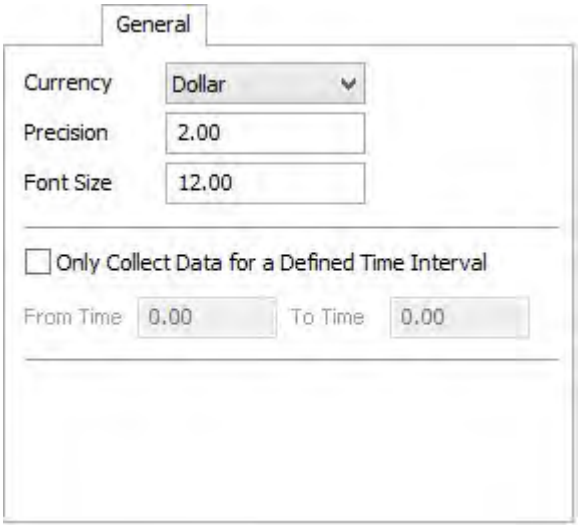
From Time - This defines when statistics will start being recorded. To

Time - This defines when statistics will no longer be recorded.

Post-Process Filtering

If the Collect Entire State History for Post-Process Filtering option is checked, FlexSim will record when each state change occurred for every object in the graph. After running the model, you can check the Filter By Time option and enter a time interval in the From and To fields. If you then click Apply or Update, the graph will display the state data for just that time interval. The Filter By Time option is unchecked when you Reset the model.

Financial Analysis



Currency - This options changes the prefix of each value to be displayed. The *Other* option allows you to define a custom string value, or leave the field blank to display no prefix.

Precision - This defines how many numbers will be displayed right of the decimal point.

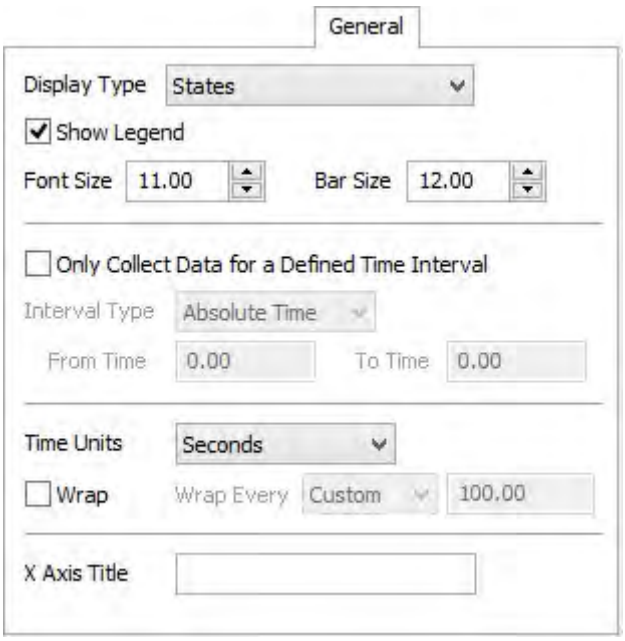
Font Size - This defines the font size of graph text.

Only Collect Data for a Defined Time Interval - This option creates a time period during which statistics for this graph will be recorded.

From Time - This defines when statistics will start being recorded.

To Time - This defines when statistics will no longer be recorded.

Gantt Chart



Display Type - This option changes the display type of graph. This may either be States or Item Trace.

Show Legend - This option adds a legend to the graph.

Font Size - This defines the font size of graph text.

Bar Size - This defines the bar height.

Only Collect Data for a Defined Time Interval - This option creates a time period during which statistics for this graph will be recorded.

Interval Type - Some charts may have an additional option of specifying the Interval Type, like the Gantt Chart. The Interval Type can be set to:

Time Units - This will scale the x-axis to the given time units. Setting the Time Units to Date Based will change the y-axis to display dates and the x-axis to display times of each day. The Wrap will automatically be set to Weekly when using Date Based.

Wrap - Check this box to have the chart wrap it's data each specified time period. This will cause the graph to grow vertically.

Wrap Every - If Wrap is checked, this defines at what time the graph should wrap.

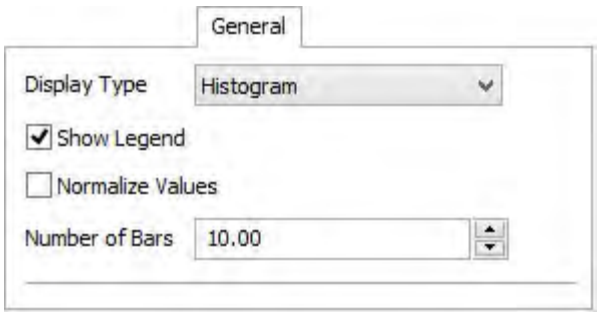
- Absolute Time: Starts collecting data at the From Time and ends collecting data at the To Time.
- Time Window: Collects data for a specified length of time. The Length field defines the length of time from the first recorded statistic to the last recorded statistic and dynamically updates as the model runs. As the data is gathered in a model run, data from the start will be removed in order to keep all of the data in the chart within the specified length of time.

From Time - This defines when statistics will start being recorded.

To Time - This defines when statistics will no longer be recorded.

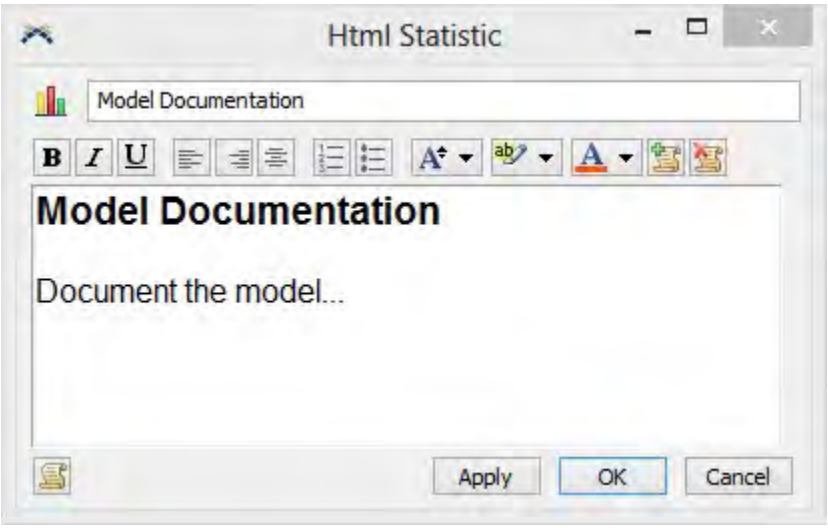
X Axis Title - Text that will display along the X Axis.

Tracked Variable Graphs



Display Type - This option changes the display style of graph.
Show Legend - This option adds a legend to the graph.
Normalize Values - This option shows the values of the histogram as a percent, rather than an actual value.
Number of Bars - This option adjust the number of bars that the histogram has.

HTML Statistic (Model Documentation)



Name Field - This field sets the name of the the documentation display graph.

- This button makes the selected text bold. If the whole selection is already bold, this button removes the bold effect.

- This button italicizes the selected text. I f the whole selection is already italicized, this button removes the italics.

- This button underlines the selected text. If the whole selection is already underlined, this button removes the underline.

- This button makes the text on the current line left justified. If multiple lines are selected, all lines are left justified.

- This button makes the text on the current line right justified. If multiple lines are selected, all lines are right justified.

- This button makes the text on the current line right justified. If multiple lines are selected, all lines are right justified

- This button places the current line in an ordered list. If multiple lines are selected, all lines are placed in the list.

- This button places the current line in an unordered list. If multiple lines are selected, all lines are placed in the list.

- This button changes the font of the current selection to be the specified size.

- This button highlights the selected text in the specified color.

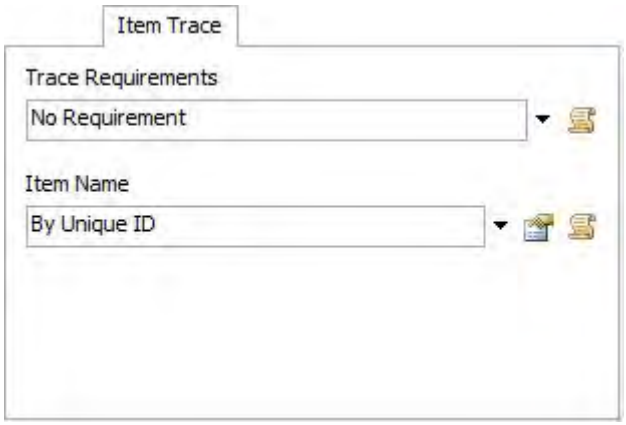
- This button changes the font color of the selected text to the specified color.

- This button inserts a section of flexscript code, which can be used to dynamically update the model documentation based on the model itself.

- This button removes the current section of flexscript.

- This button toggles the view between the visual editor and the html editor.

Item Trace Page



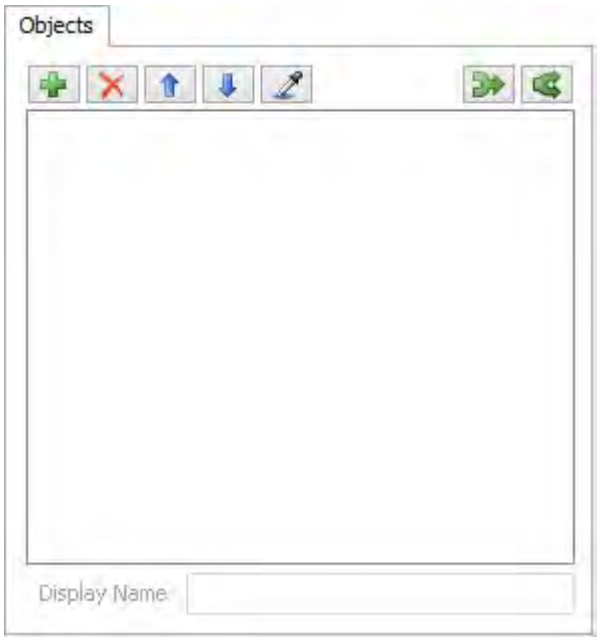
Trace Requirements - This picklist allows you to define which flowitems will be traced.

- current: The object as defined in the Objects page.
- item: The involved flowitem.

Item Name - This picklist allows you to define a custom display name for traced items.

- current: The object as defined in the Objects page.
- item: The involved flowitem.
- itemnode: The node associated with the item and its graph data. Setting the name of this node will set the display name in the graph.

Objects Page



- Adds objects or groups to the current list of objects

- Removes objects from the current list of objects

- Moves the selected object up in the list

- Moves the selected object down in the list

- Sample an object in the model.

- Combines several obj ects into one object

- Splits a combined object into its components.

Display Name - This defines the name of the object or group that will be displayed in the graph. Leaving this field blank will cause the object or group name to be displayed.

State Profile - State Statistics Only. Defines which State Profile should be used. This option is only available if the first object in the object list has additional State Profiles. All objects in the list must have the selected State Profile.

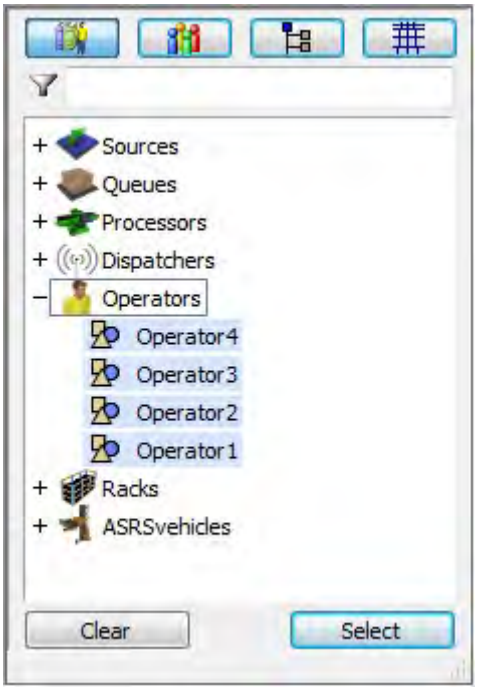
Start Trace On - Gantt Chart (Item Trace) Only. This specifies when a item trace should be started. This value can be set for each individual object.

Adding Objects

There are five ways to add objects to a graph. For more information on using the Sampler to select objects, see the Sampler page.

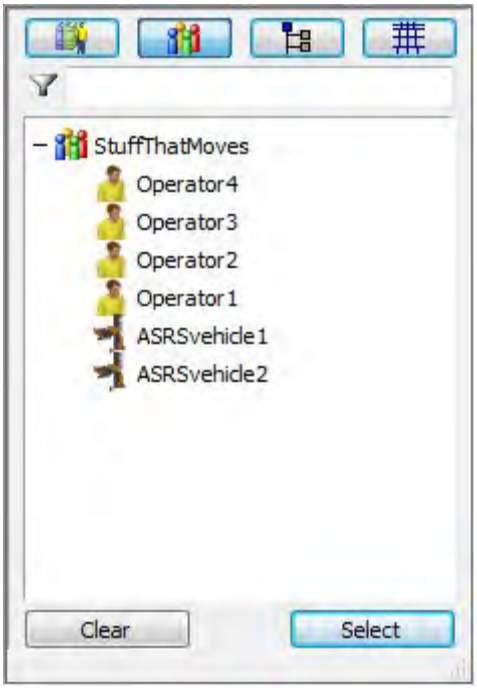
The current mode is highlighted at the top of the browsing window.

Browse by Class



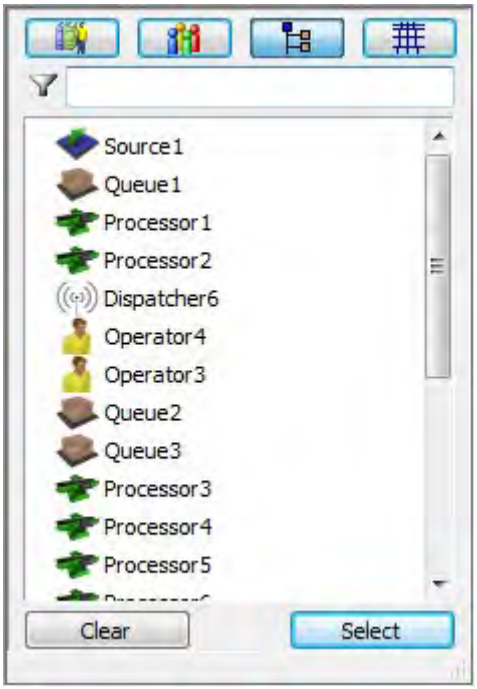
This method of adding objects sorts the objects by class. To select an entire class, click on the type icon. Click on an object to select or deselect it.

Browse by Group



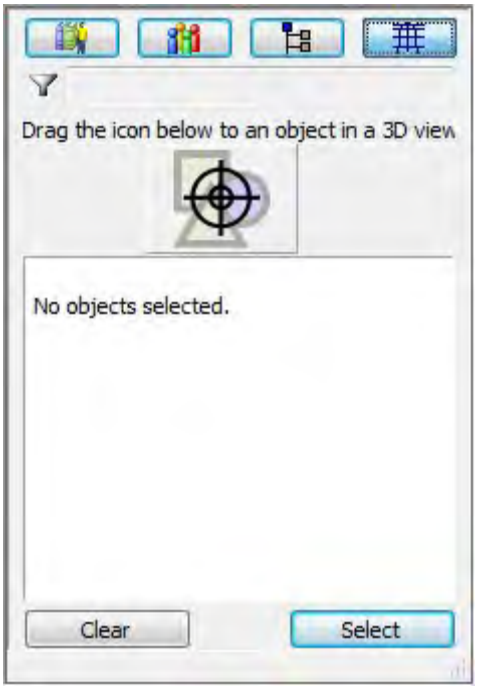
This method of adding objects sorts the objects by group. To select an entire group, click on the type icon. Click on an object to select or deselect it.

Browse by Object

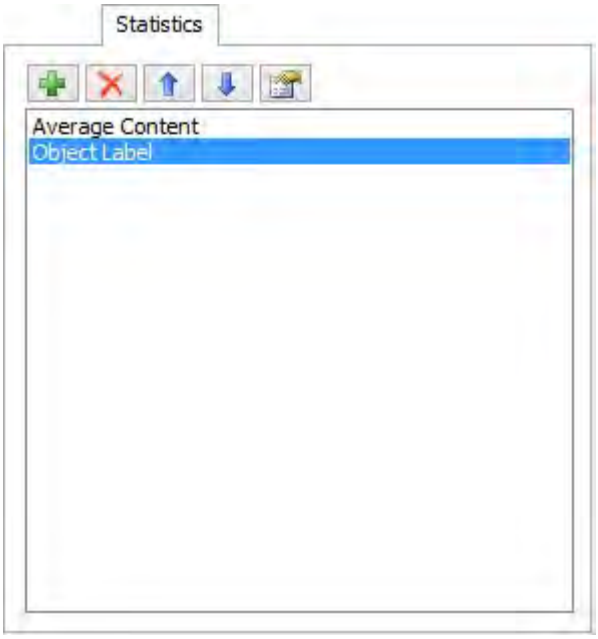


This method of adding objects lists all objects in the model. Click on an object to select or deselect it.

Select by Dragging



This method of adding objects uses the 3D model view. Simply drag the target icon from the current window to an object in the model you would like to add.



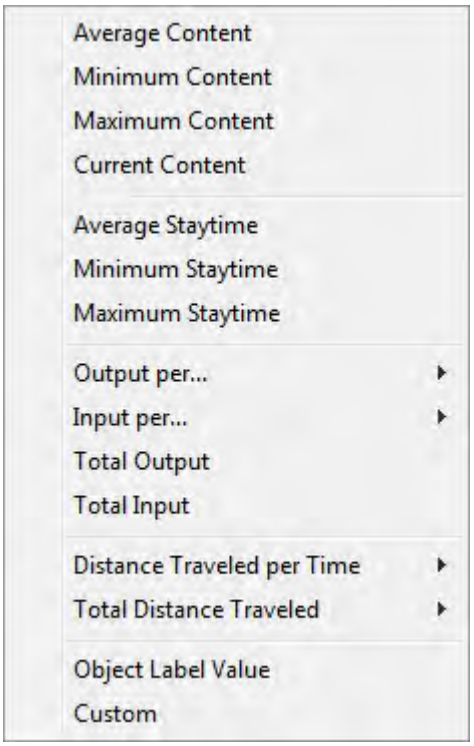
- Adds a statistic to the current list of statistics

- Removes a statistic from the current list of statistics

- Moves the selected statistic up in the list

- Moves the selected statistic down in the list

Available Statistics



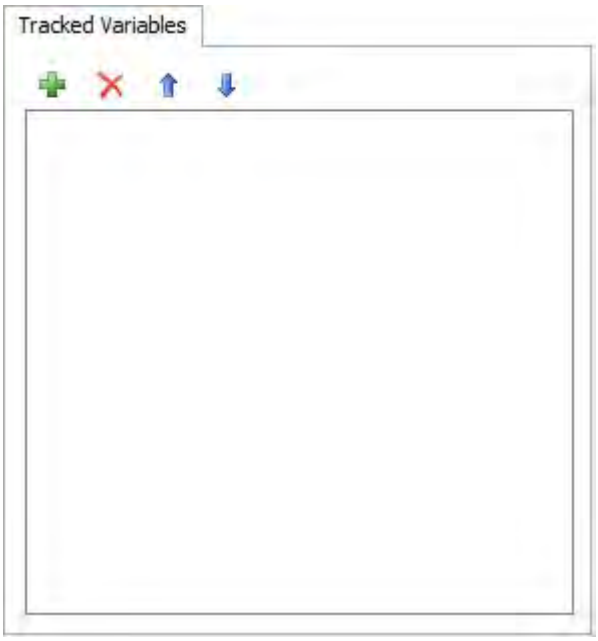
Output per... - This statistic will measure output per time, where the units of time can be selected from the side list.

Input per... - This statistic will measure input per time, where the units of time can be selected from the side list.

Distance Traveled per Time - This statistic will measure the velocity of an object. Both the length units and the time units can be selected from the side lists.

Total Distance Traveled - This statistic will measure the the total distance an object travels. The length units can be selected from the side list.

Custom - This statistic allows you to execute custom code to define what value is displayed.



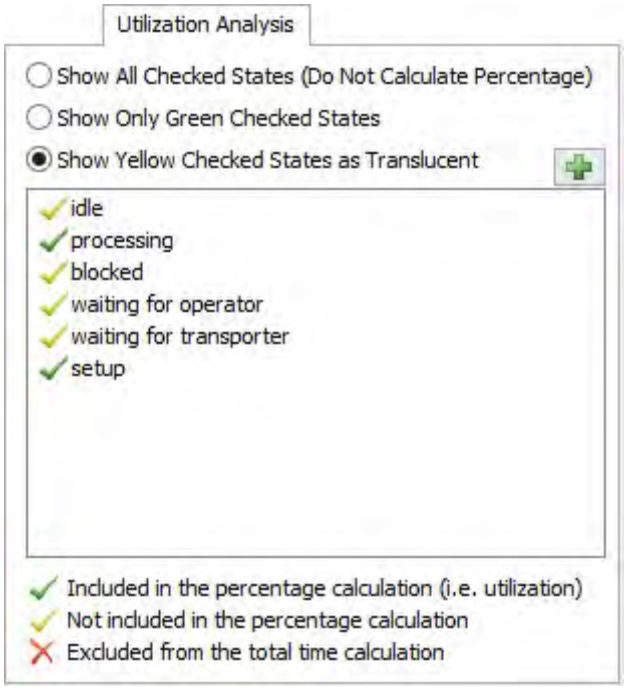
- Adds a tracked variable to the current list of tracked variables

- Removes a tracked variable from the current list of tracked variables

- Moves the selected tracked variable up in the list

- Moves the selected tracked variable down in the list

For more information on tracked variables, see the Tracked Variables page.



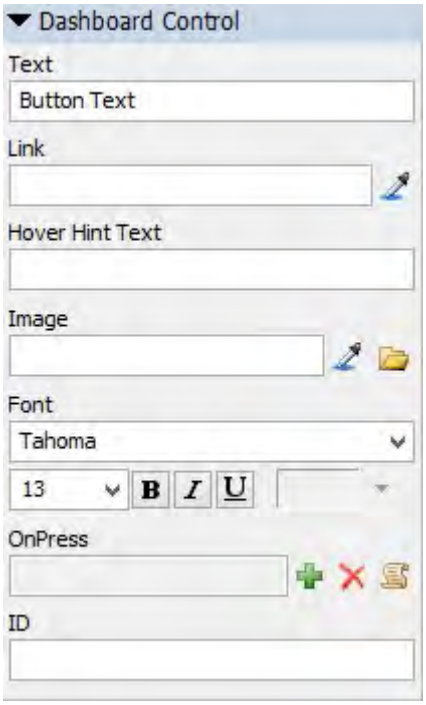
Show All Checked States - This option shows all states as solid colors. It does not calculate the percent time for which the object is utilized.

Show Only Green Checked States - This option shows the percent time that the object is in a green checked state. It does not show any other state information.

Show Yellow Checked States as Translucent - This option shows green checked states in solid green. States with yellow checks are shown in translucent colors.

- Manually add states to the list. Some states will not show up automatically in the State List. This is due to objects having a set of 'used' states that may not include states that are set manually through the setstate() command or stopobject() command or through the Time Table or MTBF/MTTR tools. Use the right click menu to remove manually added states from the list.

State List - Valid states for the selected objects will be displayed in this list. Click on the check marks to toggle between the the three calculation options.



Text - Specify the title or text of the control.

- Available for: Static Text, Checkbox, Radio Button, Button

Link - Specify the path to a node in the model. If the widget is an Edit or Dynamic Text, the Link can also be to a Global Variable.

- Available for: Dynamic Text, Edit, Checkbox, Radio Button, Combobox, Listbox, Tracker, Table, Button, GUI Class

Hover Hint Text - This text will be displayed as a tooltip when the user hovers their mouse over the control.

- Available for: Dynamic Text, Edit, Checkbox, Radio Button, Combobox, Listbox, Tracker, Button

Image - Displays an image instead of text for the control.

- Available for: Static Text, Button

Font - Specifies the font name, size, properties and color of the control's text (color only available for Static Text, Dynamic Text and Edit controls).

- Available for: All (except GUI Class)

Triggers - Some model input objects have triggers that fire, allowing you to execute your own custom code.

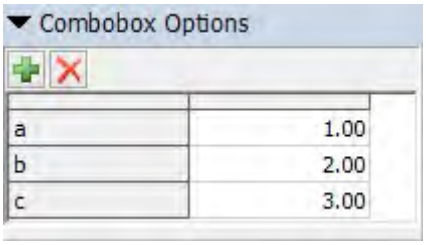
- OnPress - Fires when a button, checkbox or radio button are pressed.

- OnApply - Fires when the enter key is pressed in an edit field and when you click off of an edit field (focus is removed).
- OnDrag - Fires as a tracker is clicked/dragged
- OnSelect - Fires when an item in the combobox/listbox is selected.

ID - A Dashboard Control's ID is a string that allows you to easily reference the control through code or picklist options. Use the getdashboardcontrol() command to get a reference to the model input object (the field, button, etc).

- Available for: All

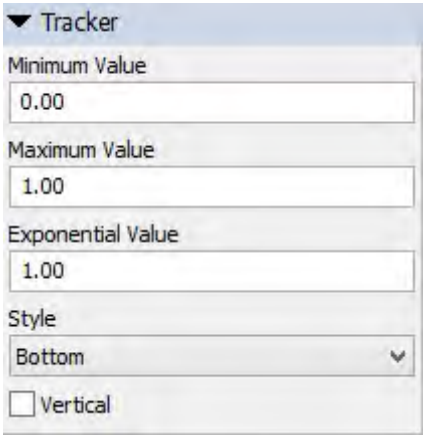
Combobox and Listbox Options



Options - A list of items to display in the combobox/listbox drop down. Values must be numeric.

Linking to string data: If a combobox or listbox is linked to a node with string data like a label, selecting an option will set the value of the linked label or node to be the name of the option, rather than its numeric value.

Tracker



Minimum Value - The minimum value of the tracker.

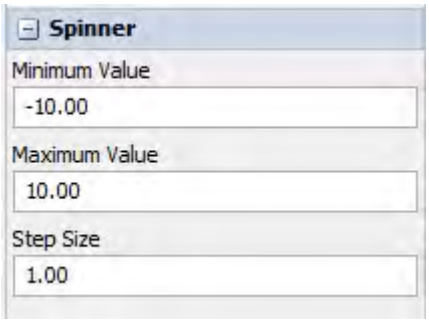
Maximum Value - The maximum value of the tracker.

Exponential Value - Changes the distribution of values between the minimum and maximum to be exponential. Setting this value to 1 will cause the tracker to be linear.

Style - Changes the style of the thumb button.

Vertical - If checked, the tracker will display vertically.

Spinner



Minimum Value - The minimum value of the spinner.

Maximum Value - The maximum value of the spinner.

Step Size - The amount the spinner increments the value by with one click.

Reports and Statistics

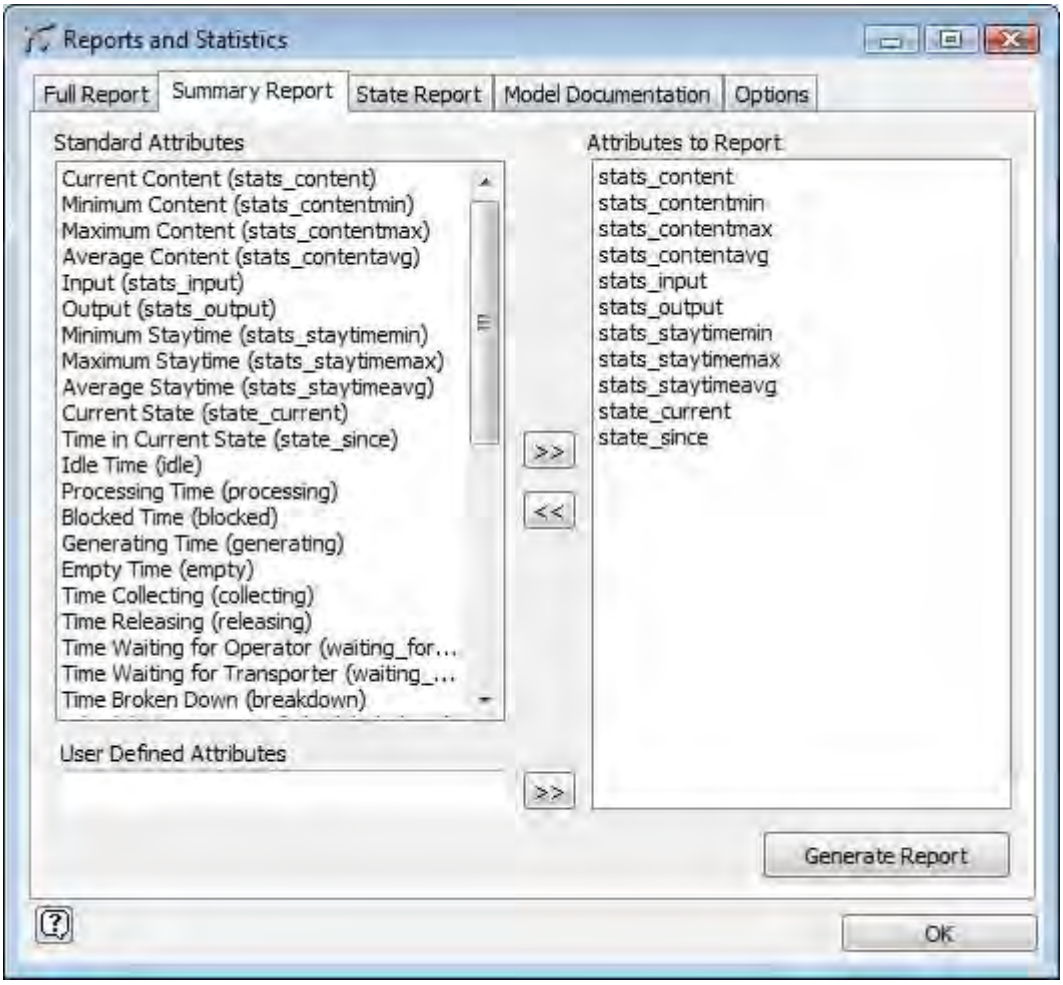
The Reports and Statistics dialog box allows the modeler to create various reports based on statistics gathered during a model run. These reports can include information about flowitem throughput, staytime, state history and other data that the modeller can select or customize. This dialog box also allows the modeler to create a document containing the most important details about objects in the model.

Reference

- Summary Report Tab
- State Report Tab
- Model Documentation Tab
- Options Tab

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.

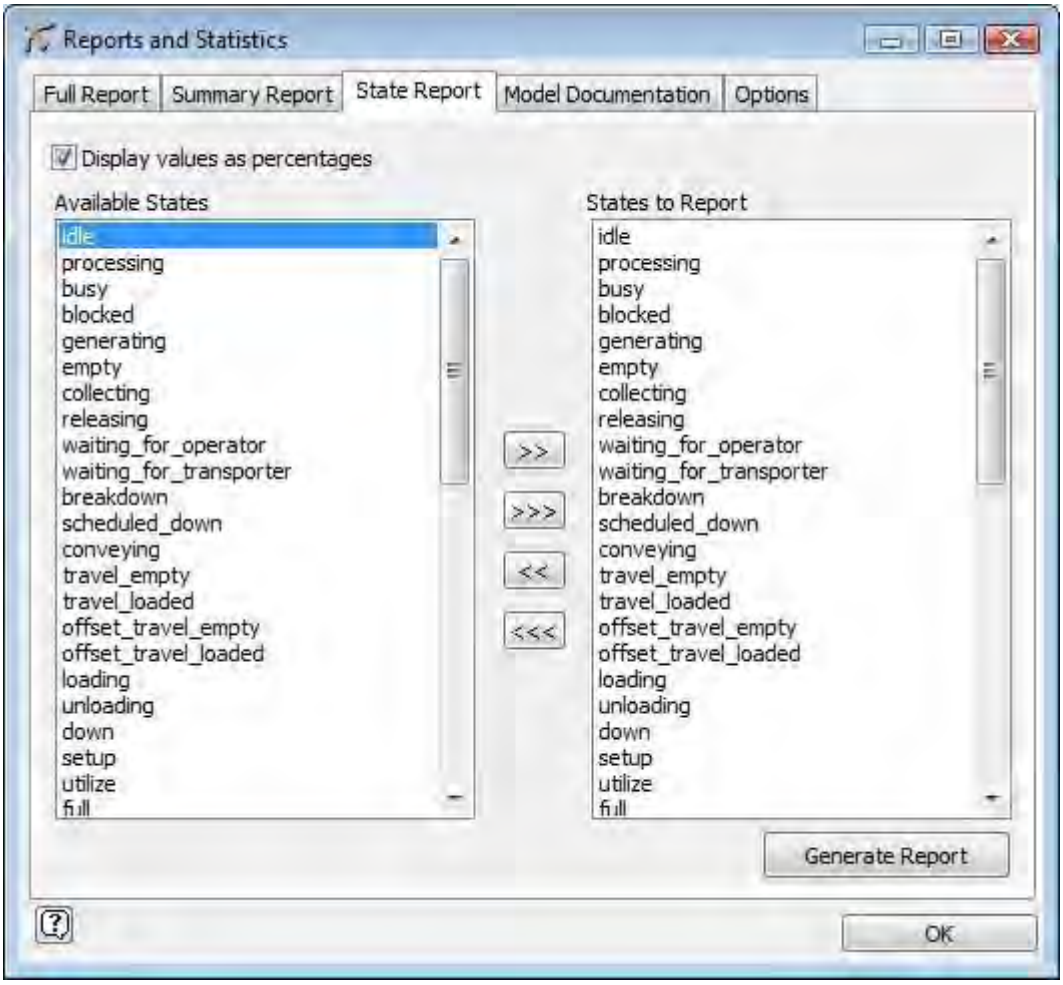
User Defined Attributes - Here you can type in the name of a label or variable that you want reported, then click the >> button at the bottom to add your own attributes to the report list. For example, if one or more Queues have a label called "lastreditem" and you want a report of all such labels and 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.

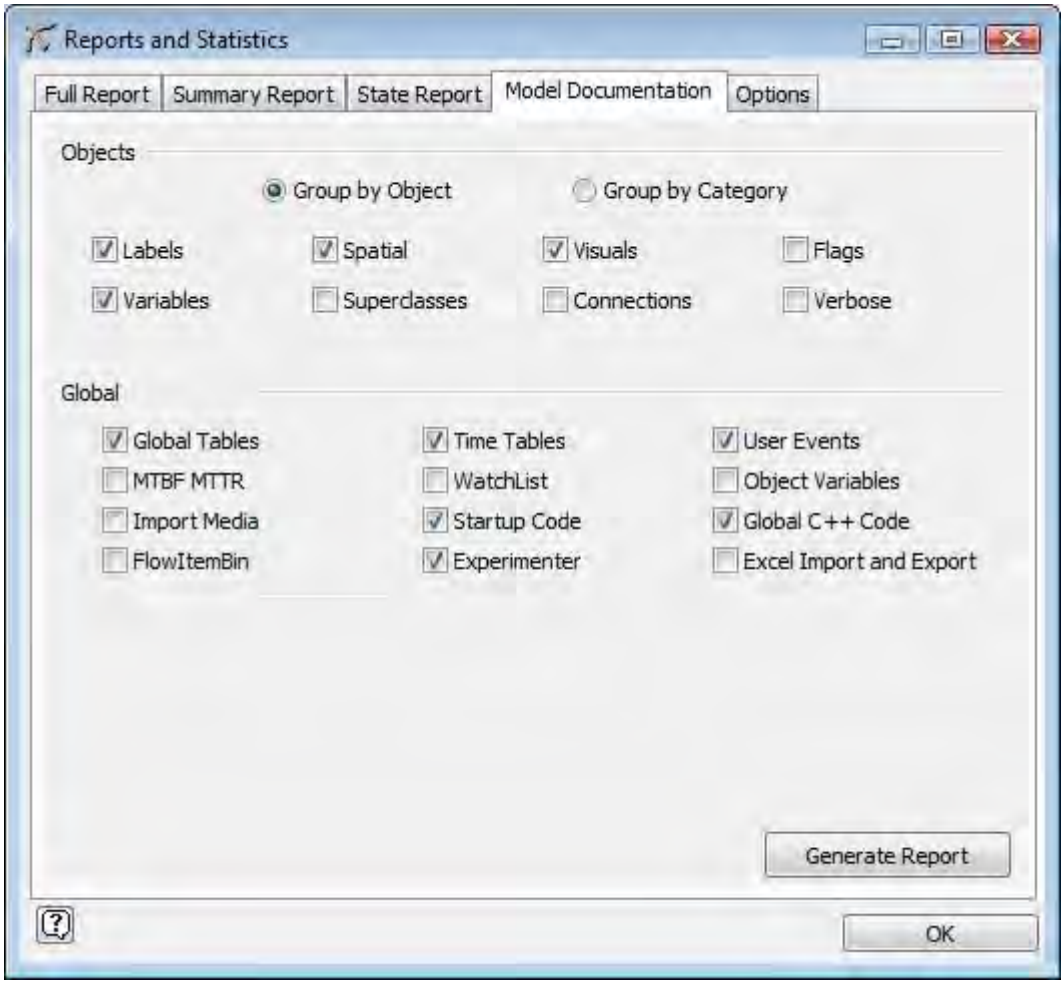
Available States - These are the states available in FlexSim that can be included in the report. To place a state in the States to Report column, highlight the state you are interested in and press the >> button. You can place all of the available states in the States to Report column by pressing the >>> 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.



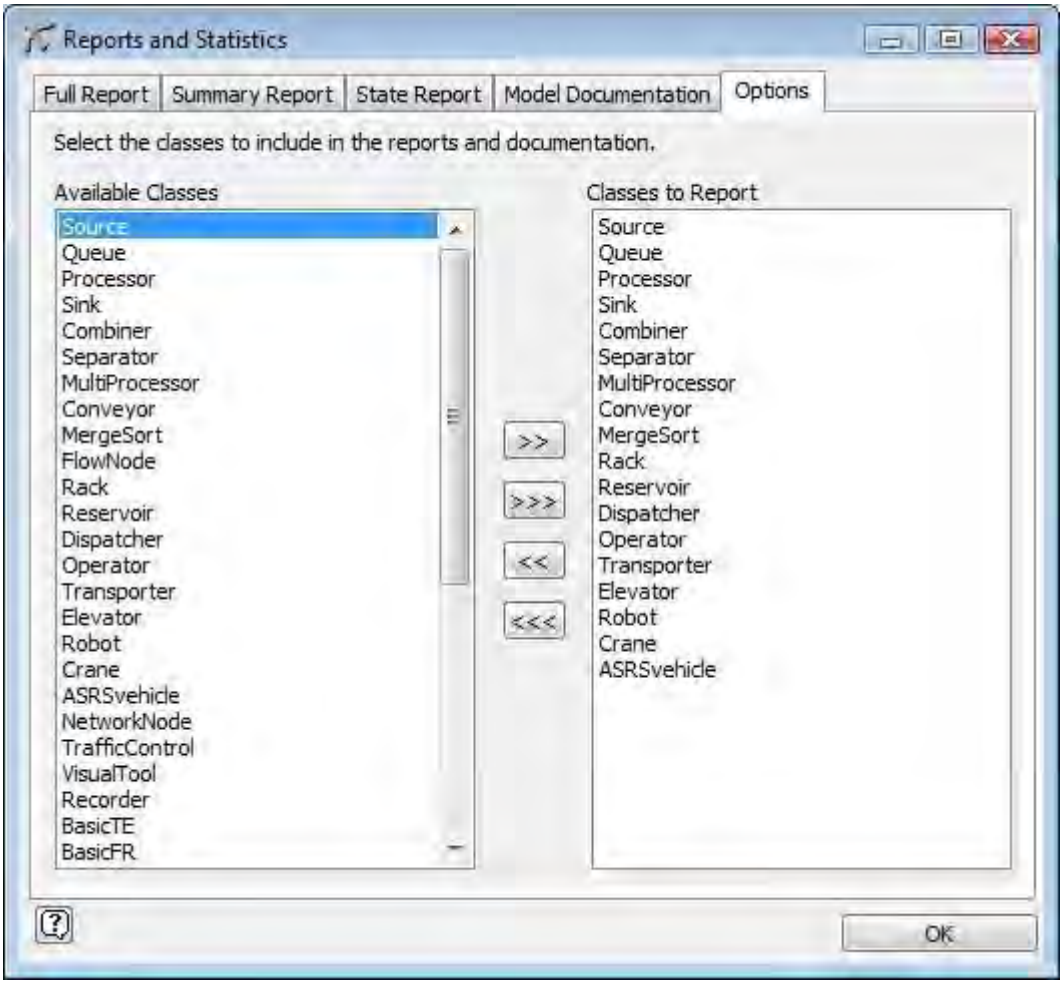
Objects - These options allow the modeler to select which attributes of the model objects should be included in the report. If "Group by Object" is selected, all of the selected attributes for each object will be together in the final report. If "Group by Category" is selected, then all of the values 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.

Writing Logic in FlexSim

FlexSim provides two options for writing custom logic: FlexScript and C++. FlexScript is generally preferred to C++ since the code works immediately in the model without having to be compiled. In previous versions, FlexScript was slower to execute than C++, however, with the introduction of the new FlexScript compiler in FlexSim 2017, FlexScript now compiles to byte code just like C++.

FlexScript is nearly identical to C++ in its syntax and application, but is simplified for ease of use and gives you a few added features. This topic covers the programming options available in FlexScript.

Topics

Where to get help

Whenever you need help with what commands to use and how to use them, you can refer to the "Commands" documentation found through FlexSim's Help menu. Refer to the FlexScript Class Reference for information about FlexScript classes.

FlexScript and C++ 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)
- text strings are usually entered between quotes. “mytext"
- parenthesis follow a function call and commas separate the arguments of the function.
  moveobject(object1,object2);
- the end of a line or function call 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 //
- to comment out blocks of text use /* text */
- don’t use spaces or special characters in name definitions (_ is ok).
- named variables and explicit values can be interchanged in writing expressions.

Variable Types

FlexSim uses just the following variable types.

Single Variables

Type	Description
int	integer number type
double	double precision floating point number type
string	text string
treenode	reference to a FlexSim node
Variant	a number, string, treenode or array
Array	a collection of Variants
var	type inferred variable

For more information on how the treenode (or FlexSim node) type works, refer to the FlexSim tree structure.

Classes

Type	Description
Object	treenode with object data
Color	rgba components of a color
Vec3	xyz components of a vector
Table	A table of elements
List	Dynamic list of things used for synchronizing tasks and logic

Refer to the FlexScript Class Reference for the complete list of FlexScript classes.

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 = model().find("Processor1"); Variant cellValue =
table[2][3]; var value = 3;
```

Declaring and Setting Array Variables

The following are examples of how to use arrays.

```
Array values = [3.5, current.centerObjects[1], [1,2,3], "test"];

Array indexes = Array(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;
```

Constructing Class Objects

The following are examples of how to use constructors.

```
Color myColor = Color(1, 0, 0);   // Makes a red color
object.location = Vec3(0, 0, 0);  // Sets the object's location to the origin
```

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)	

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. The 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)
string comparisons	current.name == "Processor5"
pointer comparisons (treenodes, Objects)	current == model().find("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)
minof()	minof(x, y)
maxof()	maxof(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)

Executing Commands

Executing a command in FlexSim is made of following parts. First type the 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 "Commands" documentation found through FlexSim's Help menu. For a quick reference of the most used commands in FlexSim, refer to the section on basic modeling functions.

Syntax	Examples
commandname(parameter1,parameter2,parameter3...);	getstat(current, "Content", STAT_CURRENT);

Dot Syntax

Classes like the treenode, Object and Variant (to name a few) allow you to call methods, as well as access properties, variables, attributes and labels (if applicable) using dot syntax.

Syntax	Examples
object.method(parameter1,parameter2,parameter3...);	current.setLocation(1, 1, 1); treenode lastItem = current.last; treenode item3 = current.subnodes[3]; int quantity = current.quantity; //Accesses label named 'quantity'

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	Examples

if (*test expression*)
{ *code block*
} else { *code block*
}
	if (item.subnodes.length == 2)
	{ item.color = Color.red;
	} else { item.color = Color.black;
	}

Logical While Loop

The while loop will continue to loop through its code block until the test expression becomes false.

Construct	Examples

while (*test expression*)
{ *code block*
}
	while (current.subnodes.length > 2)
	{ current.last.destroy();
	}

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	Examples
for(*start expression;test expression;count expression*) {	for (int index = 1; index <= current.subnodes.length; index++) {
code block }	current.subnodes[index].color = Color.blue; }

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	Examples

switch (*switchvariable*)
{ case *casenum*:
{ *code block;*
break; } default:
{ *code block;*
break;
}
}
	switch (item.type)
	{ case 1: { item.color =
	Color.yellow; break; } case 5: {
	item.color = Color.red; break; }
	default: { item.color = Color.green;
	break;
	}
	}

Redirection

Each of the flow constructs described can be redirected mid-execution with either a continue, break or return statement. The following explains how each of these statements work.

Construct	Examples
continue;	Only valid in For and While loops. Halts the current iteration of the loop and goes on to the next iteration in the loop. In a For loop the counter is incremented before continuing.
break;	Only valid in For, While and Switch statements. Breaks out of the current For, While or Switch block and continues with the line immediately following this block. Nested statements only break out of the current statement and continue on in the containing statement.
return 0;	Returns out of the current method entirely and continues with the line following the code that called this method. A value is required after the return statement because all Flexscript commands (picklists and triggers included) return a Variant type. Typing return; is not valid.

Basic Modeling Functions and Logic Statements

Here we have provided a quick reference for commands and classes that are used most often in FlexSim. For more detailed information, refer to the Command Index and the FlexScript Class Reference.

Topics

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 Objects

command	Explanation	Example
node.first	This returns a reference to the first ranked object/node inside of node	current.first

node.last	This returns a reference to last ranked object/node inside of node	current.last
node.subnodes[ranknum]	This returns a reference to the object at a given rank inside of node	current.subnodes[3]
object.inObjects[portnum]	This returns a reference to the object connected to the input port number of object	current.inObjects[1]
object.outObjects[portnum]	This returns a reference to the object connected to the output port number of object	current.outObjects[i]
object.centerObjects[portnum]	This returns a reference to the object connected to the center port number of object	current.centerObjects[1]
node.next	This returns a reference to the next ranked object of node	item.next
node.prev	This returns a reference to the previous ranked object of node	item.prev
node.find(path)	This returns the object found at path in the tree beginning from node	model().find("Floor1/Processor")

Object Attributes

command	Explanation
object.name	This returns the name of the object
object.name = newName;	This sets the name of the object to newName

object.color = Color.color	This sets the color of the object (where color is red, green, blue, random() etc)
object.color = Color(red, green, blue)	This sets the color of the object to an rgb color
setobjectshapeindex(object, indexnum)	This sets the 3D shape of the object
setobjecttextureindex(object, indexnum)	This sets the 3D texture of the object

Object Spatial Attributes

command	Explanation
object.location.x object.location.y object.location.z	Gets the x, y, and z locations of the object respectively.
object.setLocation(xnum, ynum, znum)	This sets the x, y, and z location of the object
object.size.x object.size.y object.size.z	Gets the x, y, and z size of the object respectively.
object.setSize(xnum, ynum, znum)	This sets the x, y, and z size of the object
object.rotation.x object.rotation.y object.rotation.z	Gets the x, y, and z rotation of the object respectively.
object.setRotation(xdeg, ydeg, zdeg)	This sets the x, y, and z rotation of the object

Object Statistics

command(parameter list)	Explanation

object.subnodes.length	This returns the current content of the object
object.stats.input	This returns the input statistic of the object
object.stats.output	This returns the output statistic of the object
obj.stats.state().value = statenum	This sets the current state of the object.
obj.stats.state().value	This returns the current state value of the object
obj.stats.state().valueString	This returns the current state of the object as a string
object.rank	This returns the rank of the object
object.rank = ranknum	This sets the rank of the object
getentrytime(item)	This returns the time the flow item entered the object it is currently in
getcreationtime(item)	This returns the time the flow item was created

Object Labels

command(parameter list)	Explanation
object.labelName object.labels[labelRank].value	This returns the value of the object's label
object.labelName = value object.labels[labelRank].value = value	This sets the value of the object's label
object.labels["labelname"] object.labels[labelRank]	This returns a reference to the label as a node. Often used if you have a label that is used as a table

Object Control

command(parameter list)	Explanation

obj.input.close()	This closes the input of the object
obj.input.open()	This re-opens the input of the object
obj.output.close()	This closes the output of the object
obj.input.open()	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
object.stop(downstate)	This tells the object to stop whatever its operation is and go into the given state
object.resume()	This allows the object to resume whatever its operation is
obj.output.stop()	This closes the output of the object, and accumulates stopoutput requests
obj.output.resume()	This opens the output of the object once all stopoutput requests have been resumed
obj.input.stop()	This closes the input of the object, and accumulates stopinput requests
obj.input.resume()	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

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

For more information about the Object class see the FlexScript Class Reference

Other

Tables

In order to get access to the set of table methods and properties, you'll need to create a Table variable.

```
Table table = Table("GlobalTable1"); //Global tables
Table labelTable = current.labels["TableData"]; //Label table
```

The following methods and explanations assume you have a Table variable like the ones listed above.

command	Explanation
table[row num/name][col num/name]	This returns the value in the specified row and column of the table
table[row num/name][col num/name] = value;	This sets the value in the specified row and column of the table
table.setSize(rows, columns)	This sets the size of the table in rows and columns

table.numRows	This returns the number of rows in the table
table.numCols	This returns the number of columns in the table
table.clear()	Sets all number values in the table to 0

For more information about the Table class see the FlexScript Class Reference

TaskExecuter Control

For more information on controlling TaskExecuters, refer to the task sequence section.

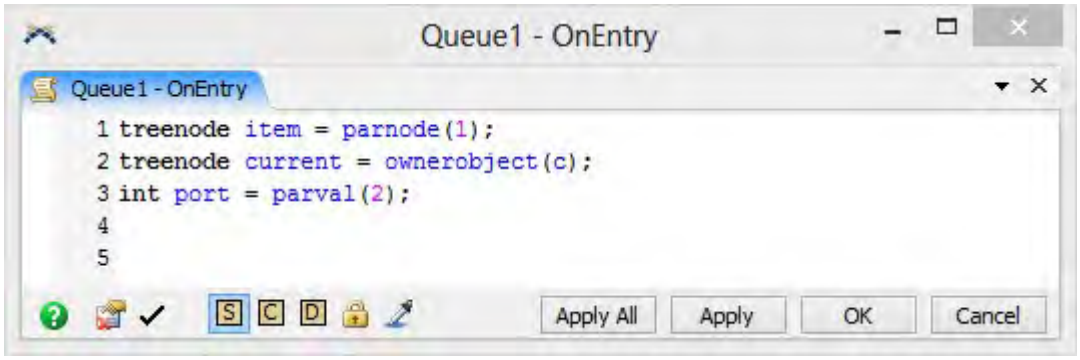
Prompts and Printouts

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
string1 + string2	This returns the string concatenation of two strings

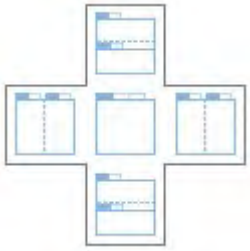
Advanced Functions

For more advanced functions and for a more complete set of command documentation, refer to the Command Summary and the FlexScript Class Reference.

Code Editor



The Code Editor window allows you to edit code for picklists and triggers throughout FlexSim. The window can be used as a floating window (default), or it can be docked into the main FlexSim window in any configuration. Just click the tab or the title bar and drag it over the Dock Windows icon.



Throughout FlexSim you will see icons. Clicking this icon will open the Code Edit window and allow you to edit that picklist or trigger's code. Alternatively, many right click menu's (like in the Tree Window) have the option Explore > As Code that will also open a Code Edit window.

When you open up the Code Editor, you'll likely see some header statements that will look something like this:

treenode current = ownerobject(c); treenode item =

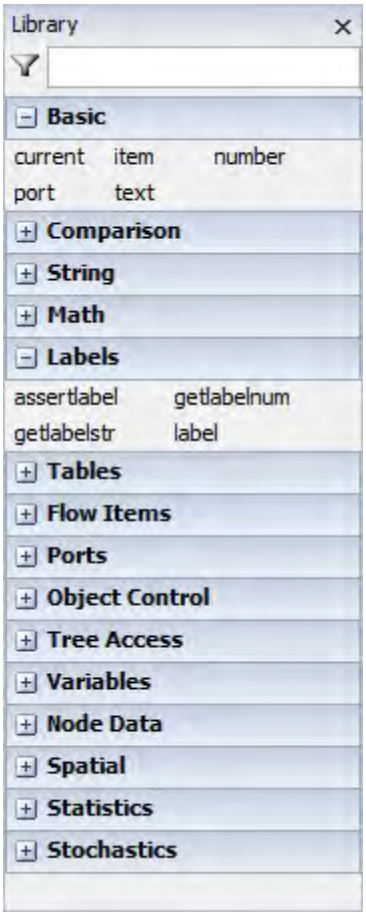
parnode(1);

For more information about item and current see the Item and Current section.

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.

Code Builders

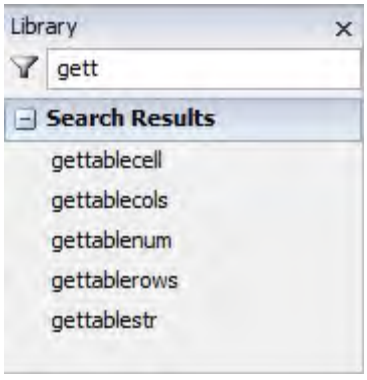
When you are editing code in the Code Editor, or entering values in a picklist field, the Library Icon Grid changes to display a list of Code Builders.



These Code Builders can be dragged into your Code Editor or picklist field to give you the correctly formatted command. Use the tab key to select each commented section of the template code. ie /*"labelname"*/

```
1 treenode item = parnode(1);
2 treenode current = ownerobject(c);
3
4 setlabelstr(current, /*"labelname"*/, /*value*/);
```

You can also filter the Code Builder list by typing in the  field.

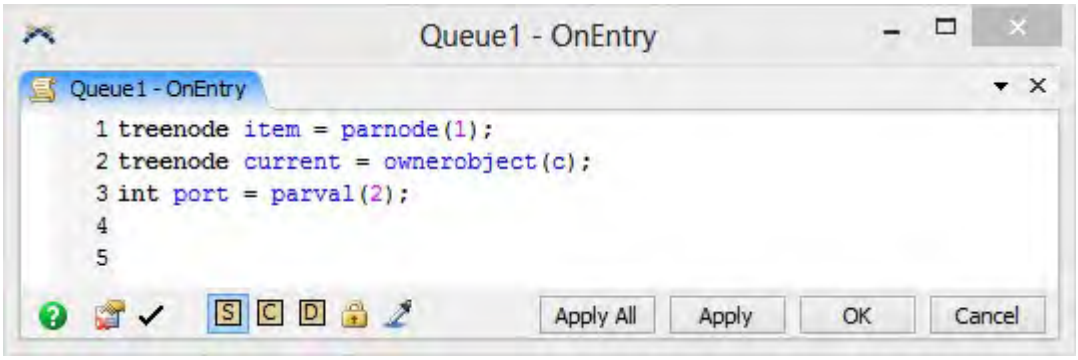


DLL Functionality

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 modellers 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 click the to lock the code state of the given field. This would also be important if you chose one of the menu options: Make all code C++ or Make all code FlexScript.



Tab Bar (Queue1 - OnEntry) - Displays the current object and trigger/picklist being edited.

- Displays this help page.

- Removes all template code. Template code may be found in picklists and triggers and takes the form of: /***popup:Conditional*/ and /** \nCondition: *//***tag:expression*//**/ - Checks syntax for compile errors.

- Toggles the current code as Flexscript.

- Toggles the current code as C++ code. Editing C++ code requires compiling the model. See When to Compile.

- Toggles the current code as DLL linked.

- Locks the toggled state of the code. This does not lock the code from being edited, but rather locks the Flexscript, C++ and DLL toggle. This can be necessary if you want your object triggers to be C++ code as the property editors automatically toggle the code to be Flexscript.

- The Sampler allows you to insert code into your code editor to reference objects, set labels, get values etc. For more information see the Sampler page.

Apply All - Saves all changes to all code editors currently docked in the same window.

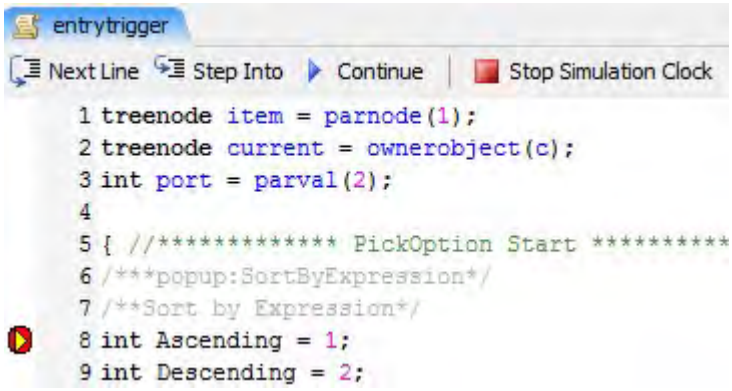
Apply - Saves changes to the currently active code.

Ok - Saves changes and closes the currently active code.

Cancel - Cancels any unsaved changes and closes the currently active code.

How It Works

Within the FlexSim code editor, there is a margin on the left side of the line numbers. By clicking in the margin, you can add a breakpoint to that line of code. The breakpoint will appear as a in the margin. You can delete the breakpoint by clicking the . When a line of code with a breakpoint is executed, FlexSim will enter debug mode. While debugging, you will only be able to interact with certain areas of the program including the debugger tools the tree window, output consoles and a limited number of other windows. The code window will change to give you tools for debugging.



The debug() Command

The debug() command can be placed in any flexscript code. It acts just like a breakpoint; it will pause the model and open the debug window when it is executed. However, it will pause the model even when breakpoint debugging is disabled.

This command makes it possible to have conditional breakpoints.

```
16 if (curitemtype > 0)
17     debug();
18
19 for (int i = 1; i <= content(current); i++) {
```

Controls

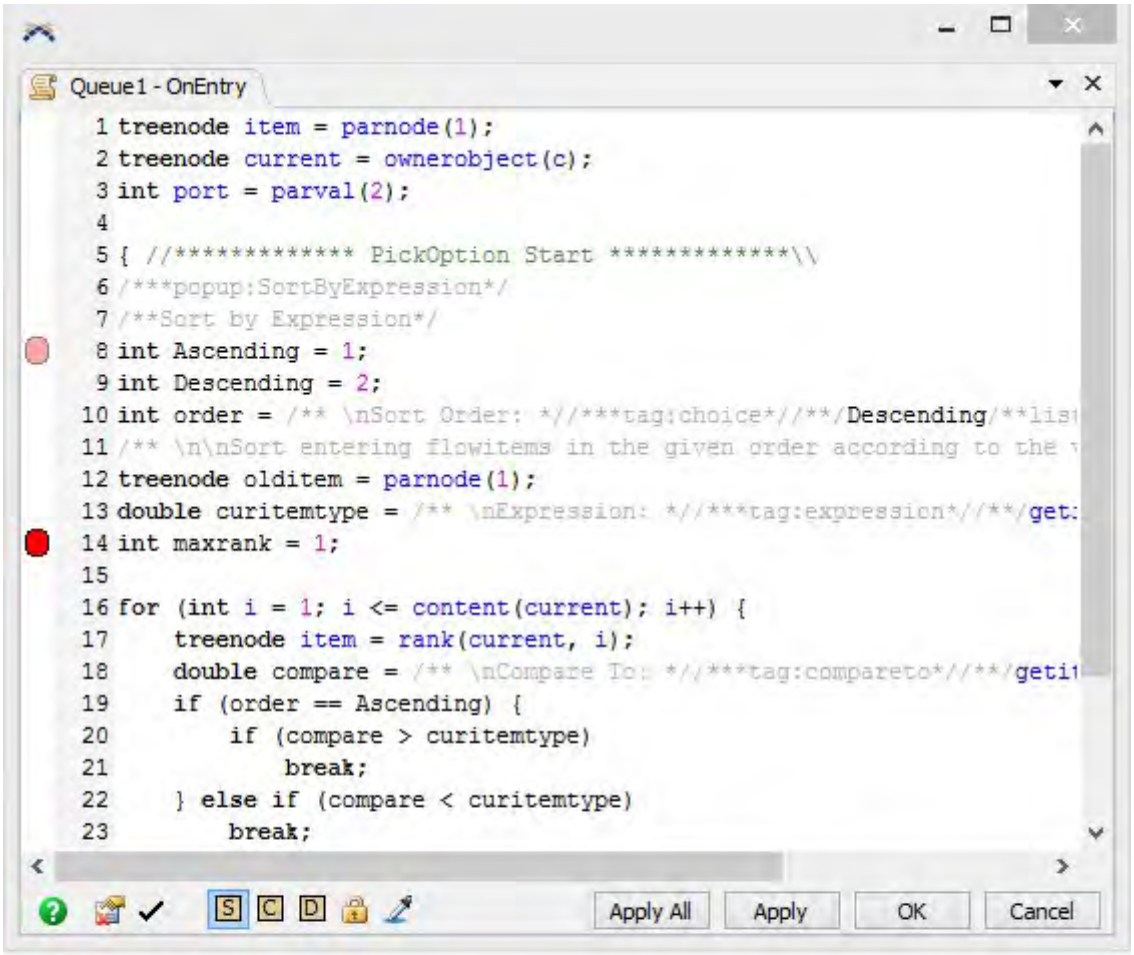
Next Line - The yellow arrow shows you what line will be executed next. By pushing the Next Line button, that line will be executed and the yellow arrow will move to the next line of code to be executed.

Step Into - This button can be used when a line of code contains certain function calls. It allows you to follow the code execution into the function and proceed line by line. When the function is finished, the

debugger will return to the code that called the function. The only functions you can follow in this way are nodefunction, executefsnode, and any custom user commands.

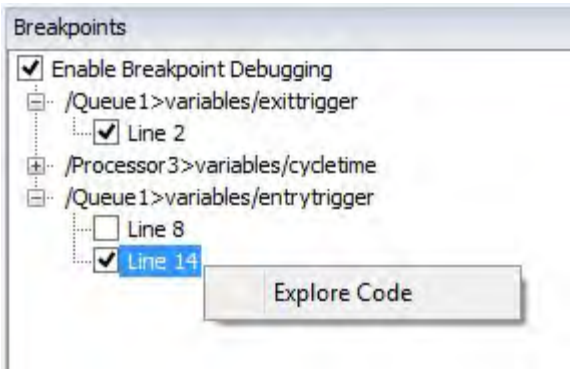
Continue - This will cause the FlexScript execution to continue until it reaches another breakpoint. If the code currently being executed finishes, then it will leave debug mode.

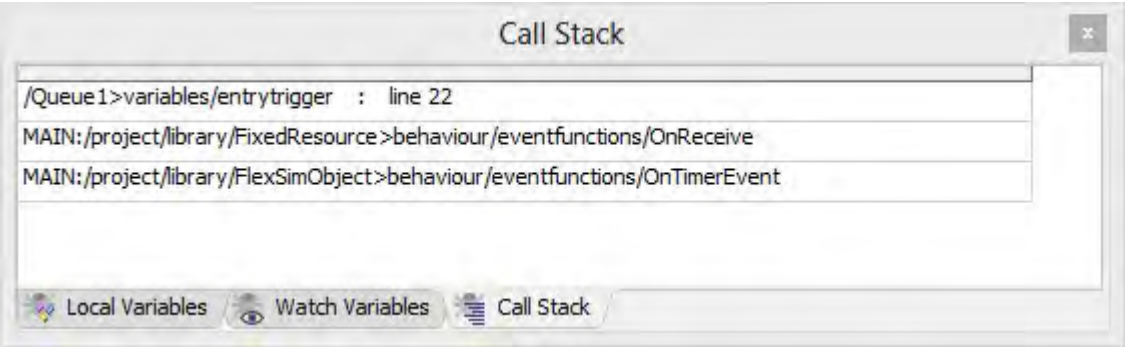
Stop Simulation Clock - This button will stop the model's simulation clock. This is particularly useful if the model is running quickly, as the Continue button may cause FlexSim to re-enter debug mode almost instantly when it continues.



The Breakpoints window is available through the Debug menu. It is a treeview with checkboxes showing you what breakpoints have been added to code in the model. You can disable breakpoints by clicking the checkbox next to the line number where they are. You can disable all the breakpoints by unchecking the "Enable Breakpoint Debugging" box. Checking and unchecking these boxes will only affect whether a breakpoint is enabled or disabled, it will not actually delete the breakpoint. To delete a breakpoint, you can highlight it in this window and press the delete key or click on its in the code window. Disabled breakpoints will appear as a in the margin and will not cause the FlexScript execution to stop for debugging.

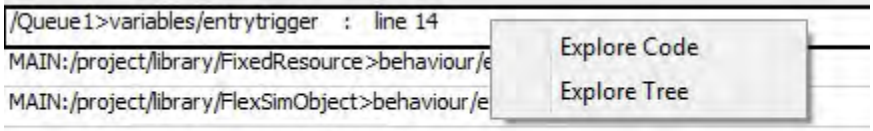
You may right click on a breakpoint in the breakpoints window to explore the associated code.



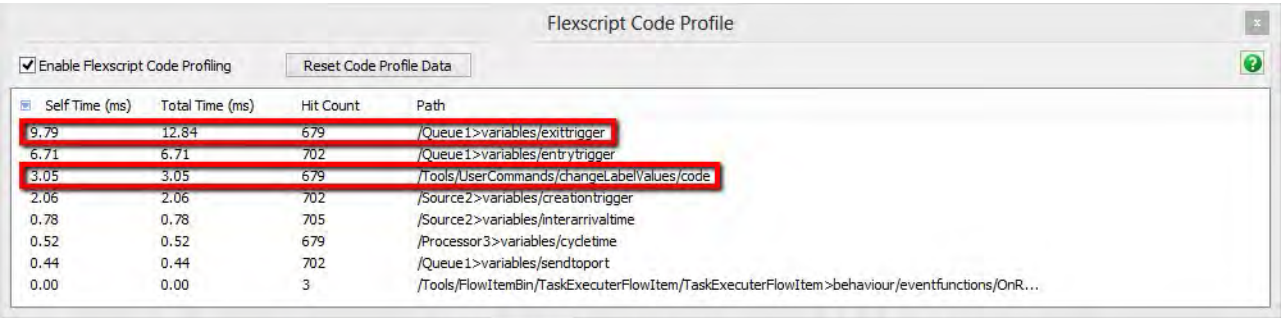


This area shows the current call stack, which is a function call history. The most recent call is always at the top of the list. In this case, the OnEntry function was called by the OnRecieve function, which was called by the OnTimerEvent function.

You may right click on any line of the call stack to explore the associated code or node in the tree.



Code Profiler



The Code Profile window profiles all the flexscript code being executed in your model.

Enable Flexscript Code Profiling- Check this box to enable Code Profiling. Code profiling will remain enabled even if the Code Profile window is closed.

Reset Code Profile Data - Clears the currently accumulated profile data.

Self Time (ms) - This is the total amount of time in milliseconds that the code has taken to execute since the model began.

Total Time (ms) - This is the Self Time plus any time spent calling other functions like User Commands within the code.

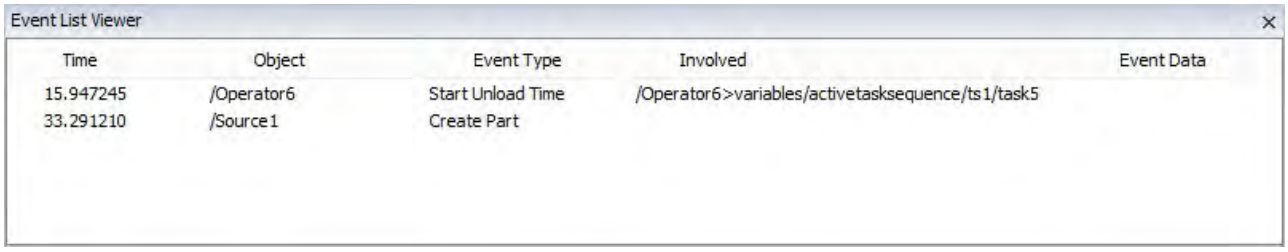
Hit Count - This is the total number of times the code has been executed. Path -

The path to the flexscript node being executed.

Self Time vs Total Time

In most cases the Self Time and Total Time will be equal. However, you'll notice above that the Self Time and Total Time of the Queue1>variables/exittrigger are different. Within the exit trigger, a user command called changeLabelValues is called. The Total Time of the Queue's exit trigger is equal to it's Self Time + changeLabelValue's Self Time.

Event List



The Event List is accessed from the Debug menu > Event List.

The Event List shows all the pending events for the model. It is useful for seeing when different events will occur in order to debug modeling issues. If you have a problem that happens during a particular event, the Event List is useful for seeing information about that event to help track down the source of the problem. If you want to only view the events for a particular object, you can right-click on the object in the 3D View and select View > View Object Events .

Time - This is the time that the event occurs.

Object - This is the path to the object, relative to the model, that the event affects.

Event Type - This is the type of event. It is the event code and will show a number value for event codes without registered names. You can use the "seteventlistlegendentry" application command to register a name for custom event types.
For example: applicationcommand("seteventlistlegendentry", 102, "My Event Type", 0); will set event code 102 to show "My Event Type" as its name in the list.

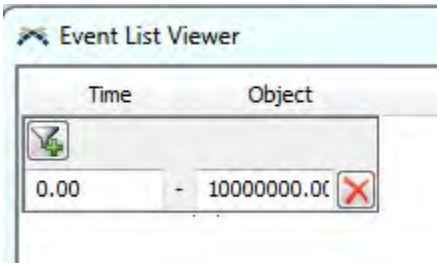
Involved - This is the path to the involved object for the event.

Event Data - This value's use depends on the event and may not be used for all event types.

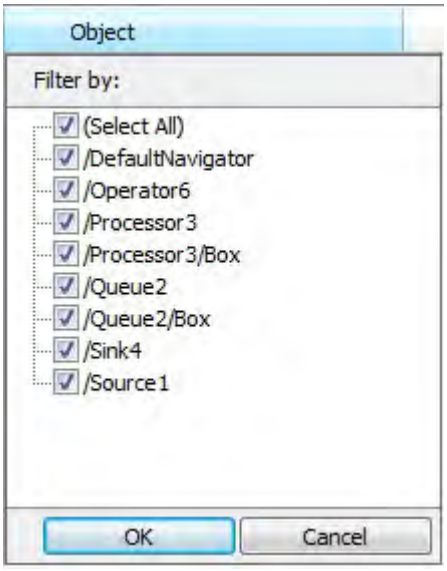
Filters

The Event List can be filtered based on the Time, Object and Involved columns. Columns with an active filter will display a . To add/edit a filter, left-click on the header name for the desired filter.

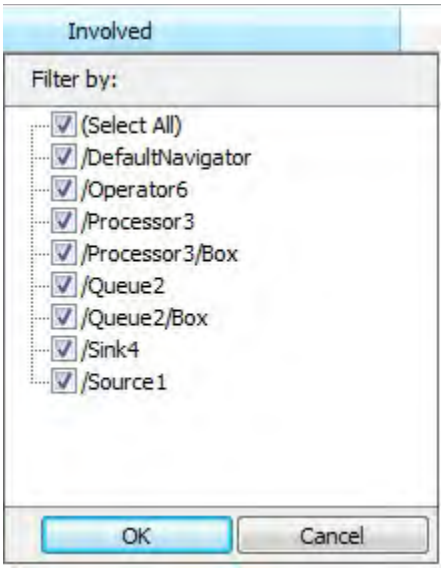
Time - Each time filter has a begin (left) and an end (right) field. Only events that occurred within those two times will be displayed. To add a filter, click the icon. To remove a filter, click the icon.



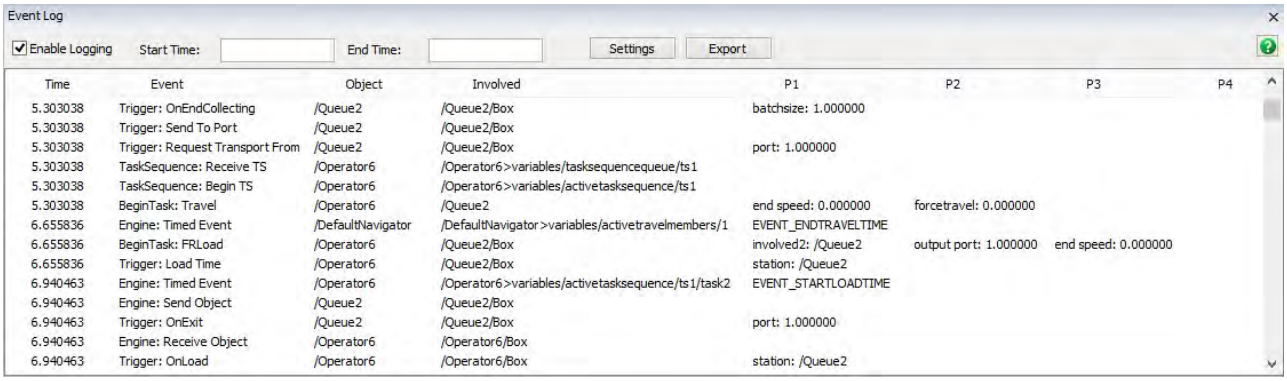
Object - This list allows you to filter the event list by which object generated the event. To include or exclude an object in the list, check or uncheck the box next to its name.



Involved - This list allows you to filter the event list by which object is involved in the event. To include or exclude an object in the list, check or uncheck the box next to its name.



Event Log



The Event Log is accessed from the Debug menu > Event Log.

When Enable Loggin is checked, the Event Log will create a record of events that occur in the model. It is useful for seeing the order in which certain events took place. For each event that happens in the model, multiple entries may be made in the Event Log to explain what happened during that event. These multiple entries will all have the same time and all be logged simultaneously when you press the Step button. The event log will be cleared when the model is reset.

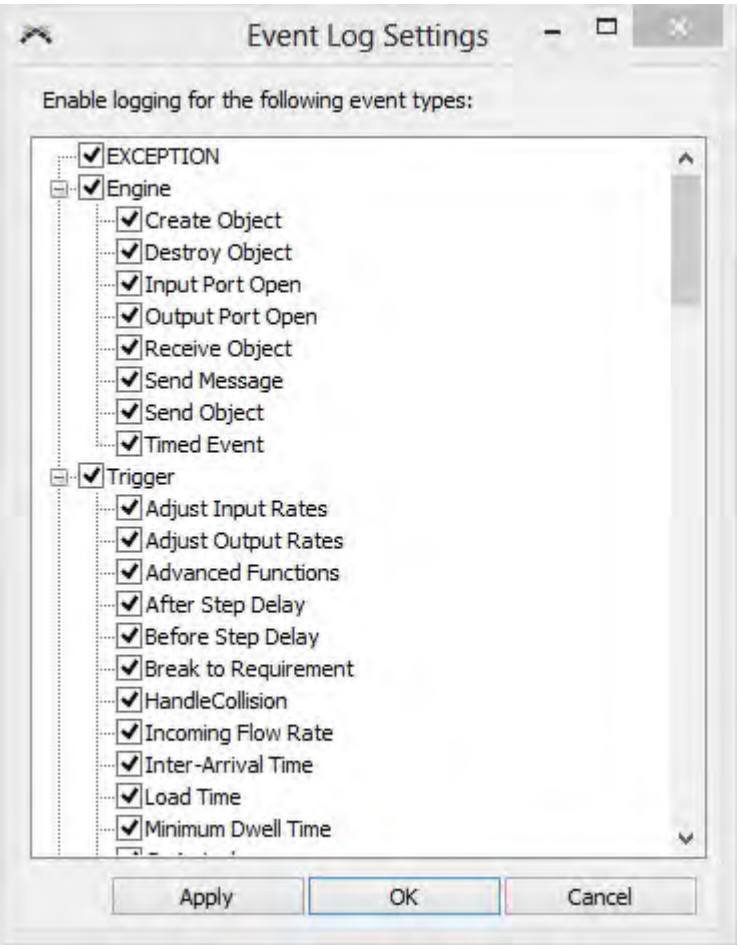
Some exceptions will be recorded in the event log. The entry immediately preceeding the exception entry will give you a clue as to where the code is that caused the exception to happen. This is particularly useful if the exception was caused by improper code in an object's trigger. More information about the exception may be available in the System Console. The model may not be behaving correctly if there are exceptions happening in the code.

Enable Logging - This will enable or disable event logging. The model will run much slower when logging is enabled so you should disable logging when you are finished using the event log.

Start Time - If you only want to log a specific time period, you can enter a start time for when the logging will begin. This will automatically be applied after editing this field without having to reset and rerun the model.

End Time - Optionally, you can specify an end time for when you want the logging to stop. If the end time is less than or equal to the start time, it will be ignored.

Settings - Within the settings window, you can set up which events should be recorded in the event log. Events that have already been recorded will not be affected by changing these settings. Events that occur after changing these settings will only be recorded if they are enabled here.



Export - This will export the Event Log as a csv file. If will only export valid events, ignoring any events that have been filtered out.

The Table

Time - This is the time that the event happened. The entries happened in order from top to bottom. Entries recorded with the same time happened in the order shown and may have happened during the same model event.

Event - This is the type of event. You can enable or disable logging for certain event types in the Settings window.

Object - This is the path to the event's object.

Involved - This is the path to the involved object for the event.

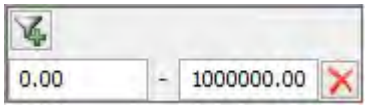
P1 - P4 - These values depend on the event and may not be used for all event types. Usually they give you information about what parameters were passed into the event or more information about the event type. This is useful for debugging if parameter values are not what you expect them to be.

Filters

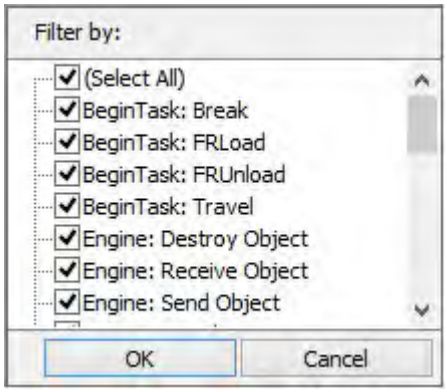
The Event Log can be filtered based on the Time, Event, Object and Involved columns. Columns with an active filter will display a . To add/edit a filter, left-click on the header name for the desired filter. Event log

entries that are no longer displayed because they have been filtered will not be exported with the Export button.

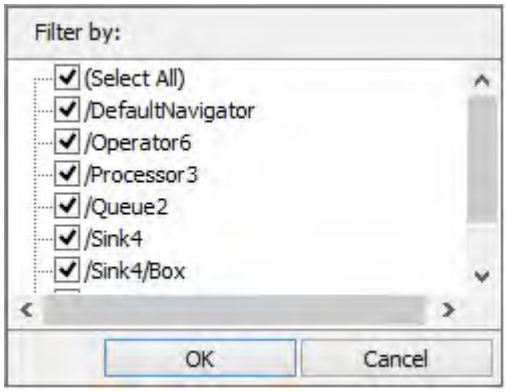
Time - Each time filter has a begin (left) and an end (right) field. Only events that occurred within those two times will be displayed. To add a filter, click the icon. To remove a filter, click the icon.



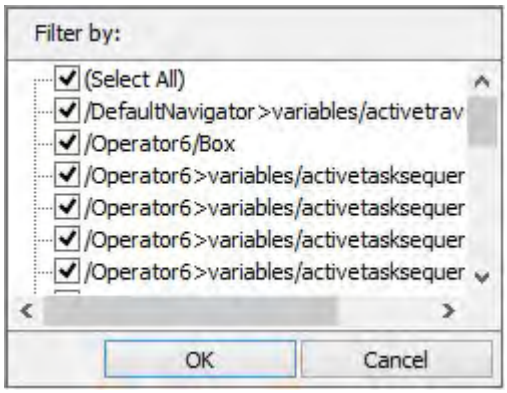
Event - This list allows you to filter the list by which event or trigger the event was associated with. To include or exclude an event from the list, check or uncheck the box next to its name.

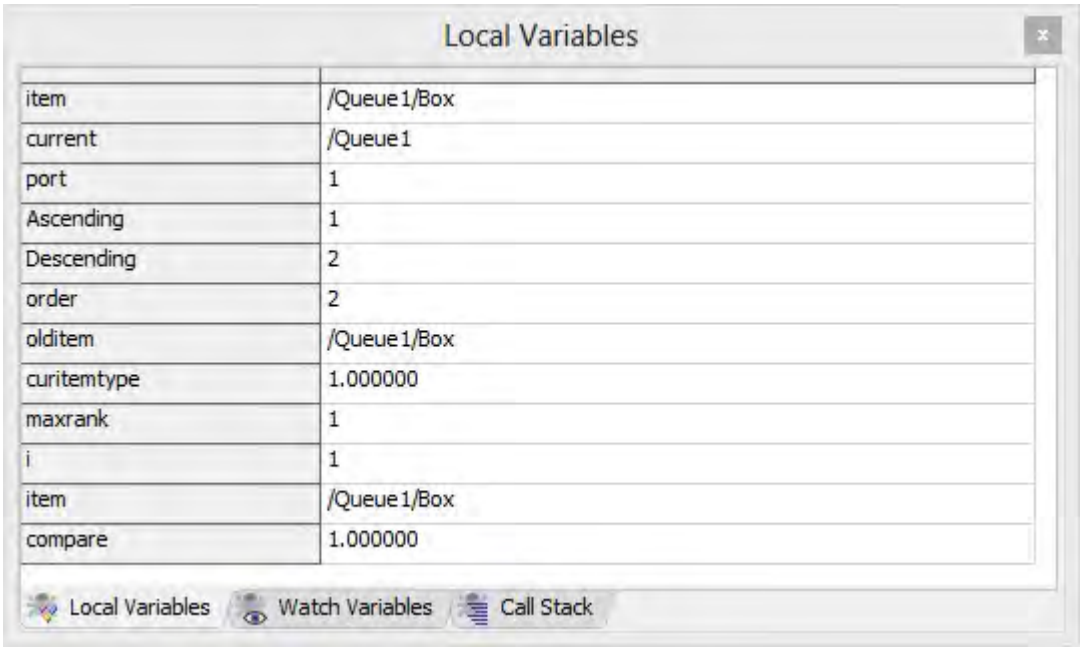


Object - This list allows you to filter the list by which object generated the event. To include or exclude an object in the list, check or uncheck the box next to its name.



Involved - This list allows you to filter the list by which object is involved in the event. To include or exclude an object in the list, check or uncheck the box next to its name.





This area shows you the current values of any locally defined variables. As you step through the code, these values will update immediately so you can see what is happening. Often, models may not behave correctly because variables in code are not what they are expected to be. This window allows you to see exactly what the variables are.

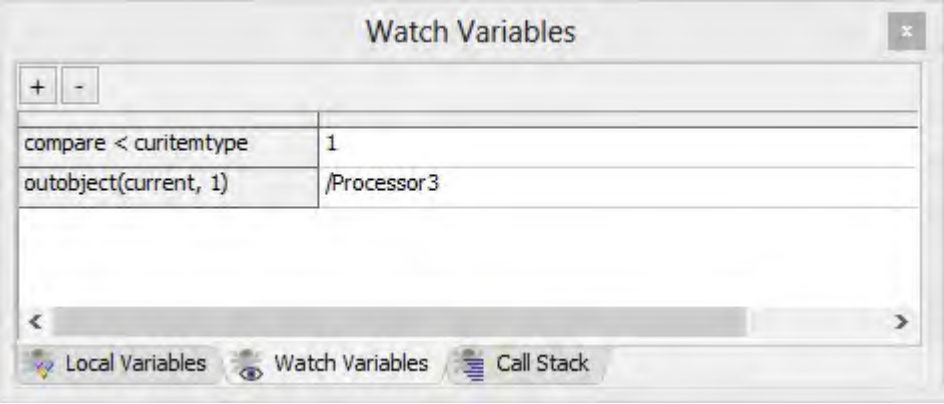
Alternatively, you can mouse over variables in the Code Edit window to see their current value.

```
 8 int Ascending = 1;
 9 int Descending = 2;
10 int order = /** \nSort Order: *//***tag:choice
11 /** \n\nSort entering flowitems in the given o
12 treenode olditem = parnode(1);
13 double c olditem: /Queue1/Box ression: *//***t
14 int maxrank = 1;
15
16 if (curitemtype > 0)
17     debug();
18
19 for (int i = 1; i <= content(current); i++) {
```

Keep in mind, the yellow arrow is pointing to the next line to be executed, so if a new variable is being initialized, you will not be able to see its value until after that line has been executed.

```
19 for (int i = 1; i <= content(current); i++) {
20     treenode item = rank(current, i);
21     double compare = /** \nCompare To: *//***tag:compareto*//**/getitemtype(item)/**/;
22     if (or compare: Out of scope
23         if (compare > curitemtype)
```

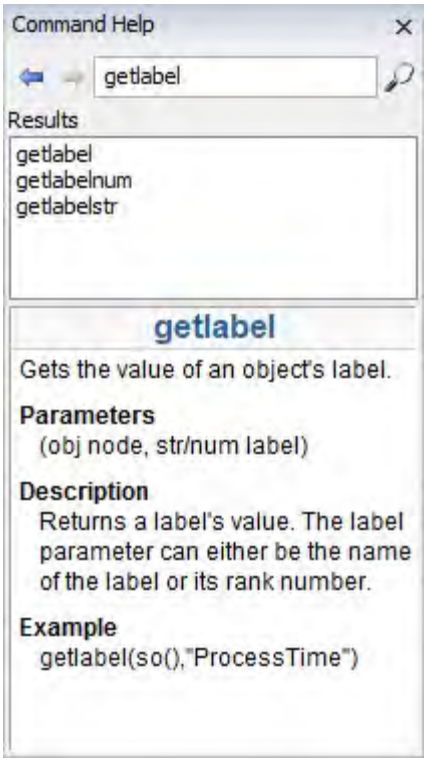
Watch Variables



This area allows you to specify other variables or expressions that you want to see, such as global variables. By pushing the + button, you can increase the number of lines in the table. The - button will delete the row that was last clicked in the table. You can double-click on a gray area of the table to enter a variable or expression. Its value will be displayed to the right This can help explain why certain conditional statements, such as used in "if" statements aren't behaving as expected. It also allows you to see global variables that otherwise are not visible on the Local Variables tab.

The Command Helper is a quick reference for flexscript commands. It can be accessed by two methods:

1. From the Help menu.
2. By pressing F1 while hovering the cursor over a flexscript command.



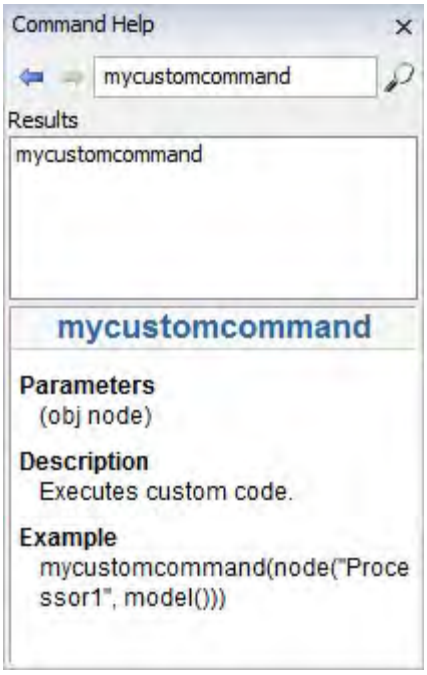
- Go to the previous command topic.

- Go to the next command topic.

- Search the command list.

Results - Displays search results.

User Commands that the modeler creates will also appear in the Command Helper and in the Command Reference.

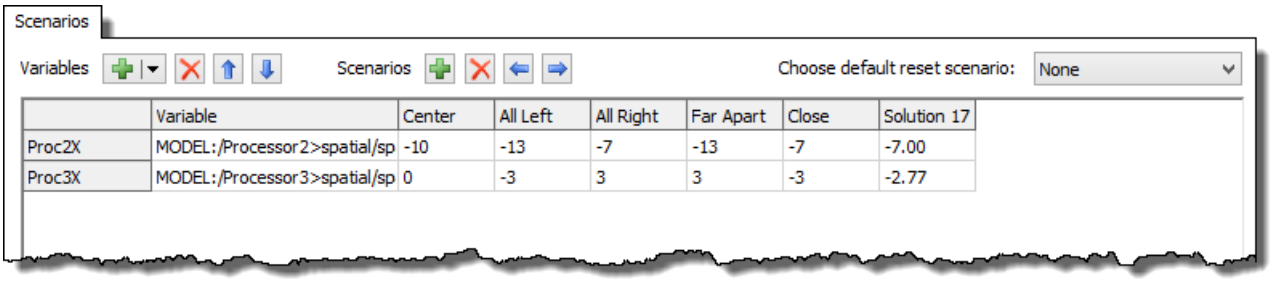

Command Help
mycustomcommand
Results
mycustomcommand
mycustomcommand
Parameters
(obj node)
Description
Executes custom code.
Example
mycustomcommand(node("Proce
ssor1", model()))

The Experimenter is used to define, run and analyze experiments on defined model scenarios. See the Experimenter Example for an example of how to use the Experimenter.

Topics

- Variables/Scenarios
- Performance Measures
- Analyzing Results

Variables/Scenarios



To create an experiment run, variables are added to the experimenter. These variables are things in the model that you want to change as part of a given experiment. They may be simple values in labels or global tables or they may be the number of Operators for a given team or the position of a FixedResource object. You create and edit Experimenter Variables in the Scenarios Tab. Each variable is a row in the Scenarios table.

Experiment scenarios are associated with experiment variables. A scenario is a specific configuration of the set of variables that you have defined. Variables must have at least one scenario, which is a numeric value that is assigned to the associated variable. In the image above, the scenarios are Center, All Left, All Right, etc. and specify the x position of Processor2 and Processor 3. When the experimenter runs, it will run a defined number of replications for each scenario.

You can have any number of variables and scenarios in an experiment.

Default Reset Scenario

The default reset scenario option allows you to set up your model based on a given scenario. Select a scenario from the drop down and then hit the Reset button to reset the model. The values assigned to the different variable will be applied to the model.

Performance Measures

Performance measures allow you to get statistical data from your model and use it to assess and compare the results of your scenarios. The statistics may include things like throughput, average wait time, or values from dashboard widgets, etc.

Analyzing Results

Once the experiment is finished (all bars in the Experiment Status window are green) you can analyze the results of your performance measures by clicking the View Results button of the Experiment Run tab. This will open the Performance Measure Results window.

The data in this window allows you to compare the results of different scenarios. There are several options for how to display the data including a Replications Plot, a Frequency Histogram, a Correlation Plot (for examining correlations between multiple performance measures), a Data Summary, and a Raw Data view. This window also displays the output of each replication and results from dashboard widgets. Performance measure results and dashboard widgets may then be exported to an HTML format for distribution.

What is Optimization?

Optimization begins with a model. For optimization in general, this model can be any system that accepts a set of inputs and then produces outputs. Optimization is the search for a set of inputs that produces the best outputs for a given model.

Within FlexSim, it is possible to optimize a simulation model. The inputs can be thought of as configuration options: How many people should be hired? Where should the storage racks be placed? What kind of machines would work best? The outputs can be thought of as results: What was the utilization of the workers? How long did items spend in the model? What was the total cost per item? FlexSim provides easy-to-use tools that allow you to specify the inputs and outputs for your model. Once this is done, FlexSim can intelligently search through the possible configurations to find the best possible outputs; in other words, it can find an optimal configuration of your model.

Note: This document contains a lot of information. If you are new to optimization in FlexSim, a good learning strategy would be to read this document, then do the optimizer tutorial, then read this document again.

Configurations, Scenarios, and Solutions

In FlexSim, the Simulation Experiment Control tool allows users to define scenarios for a model. A scenario consists of one value for each experiment variable, which results in a configuration of a model. The optimizer also generates configurations for a model based on the same set of variables used by the experimenter. In order to distinguish this kind of configuration from a scenario, it is called a solution. Users generate scenarios, and the optimizer generates solutions. Both configure a model based on the variables defined in the Experiment Simulation Control.

General Algorithm

When the optimizer is running, it is searching for better and better configurations for the model. Here's a basic version of that search algorithm:

1. Generate a model configuration (a solution).
2. Set the current model to match that configuration.
3. Run the simulation. This is also called evaluating the solution. 4. Get the outputs from that run 5. Rank the solution.
6. Generate a new solution based on the results from all evaluated solutions.
7. Repeat from step 2.

The above algorithm repeats until the optimizer runs out of time, evaluates a maximum number of solutions, is stopped by the user, or until it evaluates all possible solutions of the model.

Defining an Optimization

Variables

Optimization variables are the options for your model. When the optimizer generates a solution for your model, it is simply setting each of the variables to a specific value. There are different kinds of variables based on the option that is being represented. There is no limit to the number of variables that an optimization can have.

To add a variable to an optimization, add a variable to an the Scenario Table on the Simulation Experiment Control. If that variable can be manipulated by the optimizer, it will be automatically added to the list of variables for the optimizer. If necessary, the name of the variable will be modified to match the optimizer's variable name requirements.

Type	Description	Examples
Continuous	These variables represent options with a range of possible values. Any value within that range is a valid option for a model (4.354, 6.0, -10.45, etc.).	Values that can be very finely tuned (such as positions, lengths, and times) are often represented as continuous variables.
Integer	These variables represent options with a range of possible values. Any integer value within that range is a valid option for a model (-3, 5, 7, 25, etc.).	Quantities of discrete objects (number of people, items, etc.) are often represented as integer variables.
Discrete	These variables represent options with a range of possible values and a step between possible values. Only values that are exactly *n* steps from the lower bound are valid.	Values with discrete steps (part sizes, paired items, etc.) are often represented as discrete variables.
Binary	These variables represent options that only have two possible values: 0 and 1.	Values that represent options like yes/no, on/off, present/not present, etc., are usually represented as binary variables.
Design	These variables represent options that have a range, but a higher value does not represent "more" or "further." Instead, it just represents a different option. The optimizer will not assume that increasing this kind of variable will have a predictable effect on the system.	Values that represent options like machine type, overall floor layout, or packing strategy are usually represented as design variables.

Permutation	These variables come in groups; a single permutation variable is pointless. Permutation variables can have a value from 1 to *n*, where *n* is the number of	These variables are usually used to represent an order. For example, a part route could be represented with a permutation variable. In one

	permutation variables in a specific group. However, each value is guaranteed to be distinct. If there are three permutation variables in a single group, they can have the values like 1, 2, 3 or 3, 1, 2; they cannot have values like 1, 1, 2 or 3, 1, 3.	configuration, Station 1 could be first, Station 2 second, and Station 3 third. In another configuration, Station 3 could be first, Station 1 could be second, and Station 2 could be third.

Performance Measures

Performance measures are the results of a model run. They usually represent things such as cost, revenue, throughput, risk, average time in system, utilization, etc. There is no limit the the number of performance measures monitored by an optimization.

All performance measures added to the Performance Measures tab in the Simulation Experiment Control are automatically added to the optimizer. If necessary, the name of the performance measure will be modified to match the optimizer's variable name requirements.

Constraints

Constraints are a way of specifying additional conditions that a given model must maintain. Because of the nature of simulation, many constraints are naturally enforced by the model. Occasionally, however, it is necessary to enforce additional relationships.

Constraints are mathematical expressions that result in boolean values. They can be composed of variables, performance measures, literal values, and basic mathematical functions. Examples of valid constraints are shown below:

```
processorPositionX <= 30  numberOfEmployees * 500 >=
0.5 * totalRevenue
0 <= numberOfEmployees + numberOfTrucks <= 30
```

Valid Mathematical Comparisons
When using mathmetical comparisons within constraints, only >=, <= or == are valid. > and < will be evaluated as >= and <= respectively.

If a given configuration breaks one of these constraints, it is marked as infeasible. The optimizer will only mark a configuration as optimal if it is feasible.

A constraint can have a comma-separated list of values; for example:

```
processorPositionX <= (30, 35, 40)
```

If a constraint like this exists, the optimizer will basically run the optimization routine for each value in the list, but report the results as one set.

Note: The graphical display for the optimizer may not display correctly if a constraint list exists and multiple objectives are searched. The optimizer will correctly perform the search, but a different utility (such as Excel) will need to be used to visualize the results.

Objectives

Objectives are functions used to determine whether one configuration is better or worse than another. They are very similar to constraints, except they return an actual value rather than a boolean result. Objectives also have a direction; they can be maximized or minimized. For example, an objective function called "cost" might be something like:

```
machineCount * 5000 + throughput * 50
```

If cost were minimized, the optimizer would look for solutions with the lowest cost. If cost were maximized, the optimizer would look for solutions with the highest cost. It is very important to set the correct direction for each objective.

Search Modes

The optimizer has three different search modes. A search mode is the way the optimizer will use the objective function while it runs. The search modes are as follows:

- Single - The optimizer will try to maximize or minimize a single objective. The best solution found in the time given will be returned.
- Weighted - The optimizer will try to maximize the cumulative value of all given objectives. Each objective is given a weight and a direction. If a particular objective is to be minimized, its value is multiplied by -1 before it is added to the cumulative objective. The best solution found in the time given will be returned.
- Pattern - The optimizer will search for solutions that optimize all given objectives. This search mode returns a set of optimal solutions. They are optimal in that for each one, improving any objective worsens the others. For example, a given optimization may maximize profit and minimize cost. For a solution to be optimal, no other solution can exist that increases profit without increasing cost. This is also called "Pareto Optimal."

There are more options for weighted searches, including an objective min, max, and goal. These are not well documented, even by OptQuest; use at your own discretion. Leaving blank or at zero in FlexSim will make sure they are ignored.

Other Optimization Options

Because the optimizer must run a simulation to evaluate each solution, there are many options about those simulations that can be set, such as:

- The model run time per solution evaluation
- The model warmup time per solution evaluation
- The number of replications per solution
- The kinds of data that should be saved per replication

There are also many options about the optimizer itself that can be set, such as:

- The wall time - the amount of real time the optimizer will spend searching for an optimal solution
- The maximum number of solutions to evaluate • The use of experimenter scenarios as solutions.

Running an Optimization

Once the optimization has been properly defined, the hard part is done. You can click the Optimize button on the Optimizer Run tab of the Experiment Simulation Control. At this point, FlexSim will use the same multi-threaded capability found in the experimenter to evaluate multiple solutions simultaneously. This

allows FlexSim to rapidly search through generated solutions, as well as provide meaningful feedback about the progress of the optimizer.

Interpreting the Results

When the optimizer is finished, one (or several) solutions will be marked as optimal solutions. Unless the optimizer evaluated all possible solutions (which is generally impossible), then the optimal solutions are simply the best solutions that the optimizer found. They are not guaranteed to be the absolute best solutions. They are only guaranteed to be better than all other solutions that were evaluated.

The true value of the optimizer, however, will be missed if you only look for those few optimal solutions. As the optimizer searches for optimal solutions, it creates a database of each solution and its effects on the model outputs. This database can be used (with the proper application of statistical techniques) to find previously unknown relationships between model variables, or to determine which constraints were most restrictive. Using this data appropriately can lead to new insight into the behavior of the model.

Under the Hood

FlexSim uses the OptQuest engine from OptTek Systems, Inc. OptQuest is a trusted, industry-standard optimization engine that uses the latest evolutionary algorithms to effectively search complicated design spaces. More information on optimization with this engine can be found in the OptQuest documentation here.

Experimenter/ Optimizer Example

Introduction

This tutorial introduces the experimenter and optimizer found in FlexSim. These tools each help to answer the "what-if" questions for a model: What if the milling station were placed differently? What if more people were hired for a given shift? What if the order of these processes was changed? The experimenter allows you to try many different versions of your model and evaluate the performance of each one. Alternatively, the optimizer can automatically search through many different versions of your model, looking for the best one. They are both very powerful tools that enable you to learn about and improve your system. Note: Using the optimizer requires the OptQuest license. Contact FlexSim for more information about obtaining this license.

What You Will Learn

This tutorial will walk you through the following steps:

- Creating an experiment
- Running an experiment
- Viewing experiment results
- Designing an optimization
- Running an optimization
- Viewing optimization results

Along the way, we will cover the definitions of related terms, such as:

- Experiment variables
- Performance measures
- Scenarios
- Optimizer variable types
- Constraints
- Objectives
- Solutions

Step-By-Step Model Construction

For this tutorial, let us examine a very simple situation. A single worker must carry an item from a source to a processor. After the item finishes processing, the worker must carry it to a second processor that takes longer than the first. After the the second processor finishes the item, the worker then carries it to the sink. Now let us suppose we want to maximize the throughput (which is also tied to revenue) of this system by adjusting the position of the processors. If each processor could be moved up to three meters right or left, where should each be placed? It would be very difficult to intuitively know how to place both processors to maximize throughput. In order to solve this problem accurately, we will use the experimenter and optimizer. Now, obviously this is a drastically simplified scenario, but often in real life you have

situations where you want to see how various layouts affect overall throughput. This is a very simplistic implementation of such an experiment.

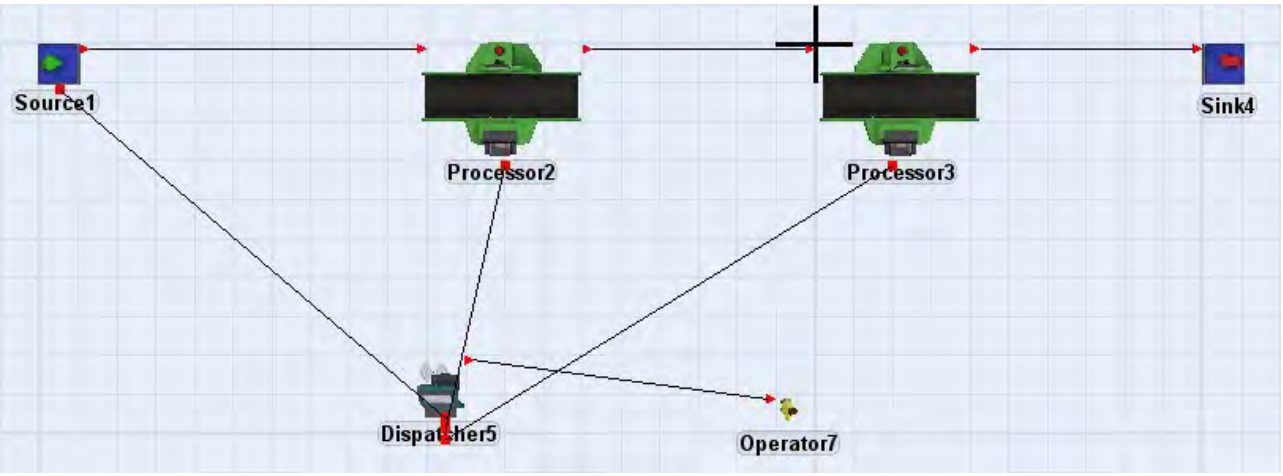
Step 1: Building the Model

Model

Create a new model using Seconds, Meters, and Liters for units.

Objects

Create a Source, two Processors, a Sink, a Dispatcher, and an Operator. Lay these objects out as shown below.



Positions

Set the location of the objects according to the table below:

Object	X Position	Y Position
Source1	-20.0	0.0
Processor2	-10.0	0.0
Processor3	0.0	0.0
Sink4	10.0	0.0
Dispatcher5	N/A	N/A

Operator7	N/A	N/A

Dispatcher5 and Operator7 do not need to be in a particular place, but they should not be in line with the rest of the objects.

Logic

Set the following logic:

- Set Source1, Processor2, and Processor3 to Use Transport (available in the Quick Properties menu).
- Set the process time of Processor2 to normal(10, 2) (also available in the Quick Properties menu).
- Set the process time of Processor3 to normal(12, 3).
- Set the reset position of Operator7 to its current position.

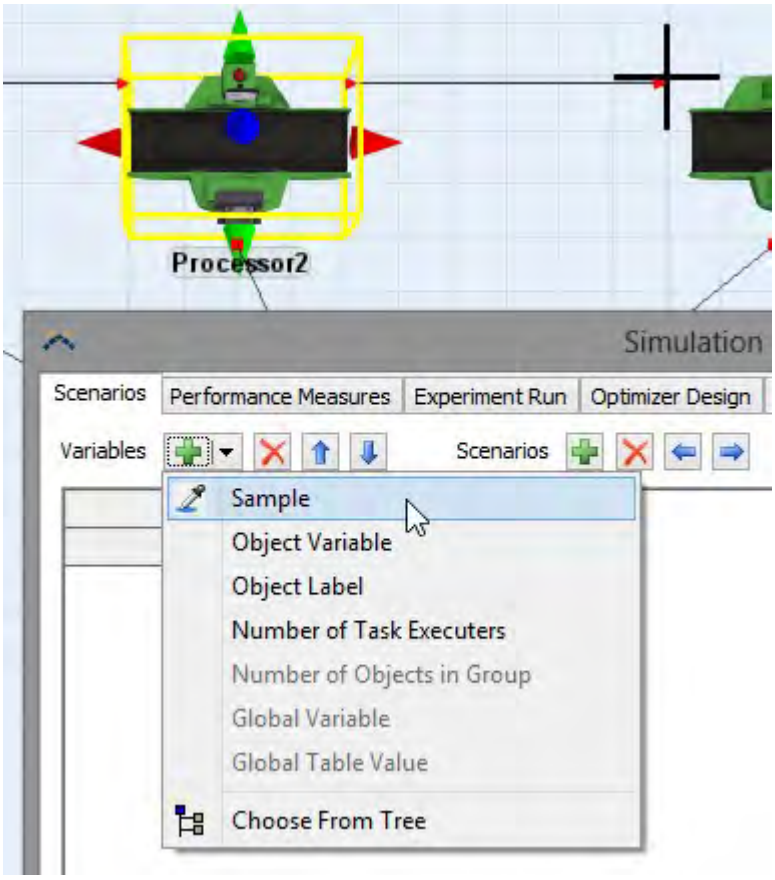
Step 2: Defining the Experiment

The remainder of this tutorial deals with using the Experimenter, found in the Statistics menu. The optimizer uses most of the functionality already present in the experimenter.

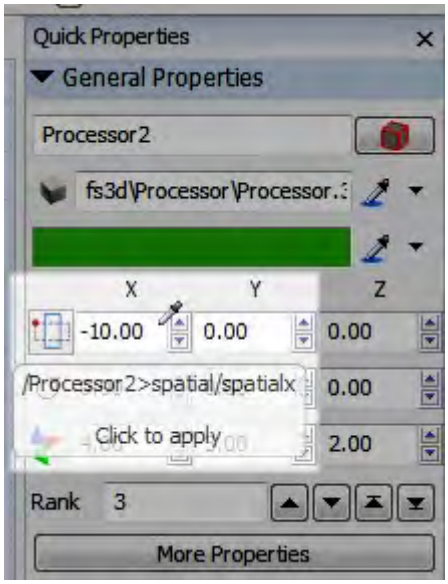
Creating Variables

Open the Experimenter window. Position the window so you can see the processors in the model as well as the window. Then, for Processor2 and then Processor3, follow these steps:

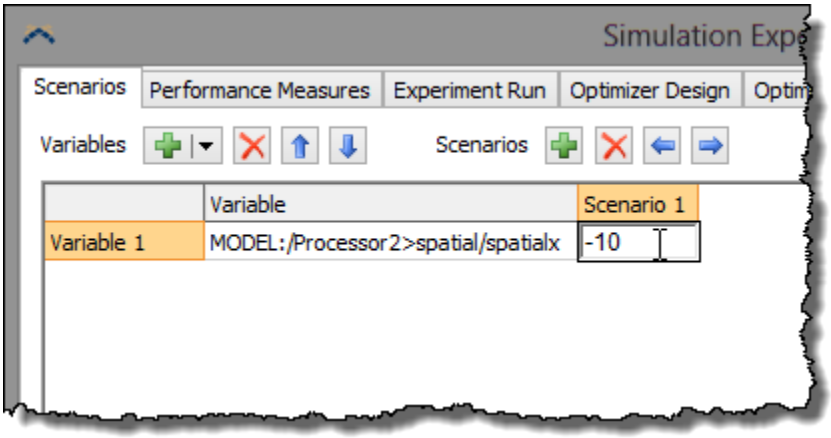
- Click on the processor in the 3D view.
- Click on the Position Reference button  in the Quick Properties and set the position reference to Direct Spatials.
- Click the down arrow on the  button.
- Select Sample from the popup menu. This puts the cursor in sample mode.



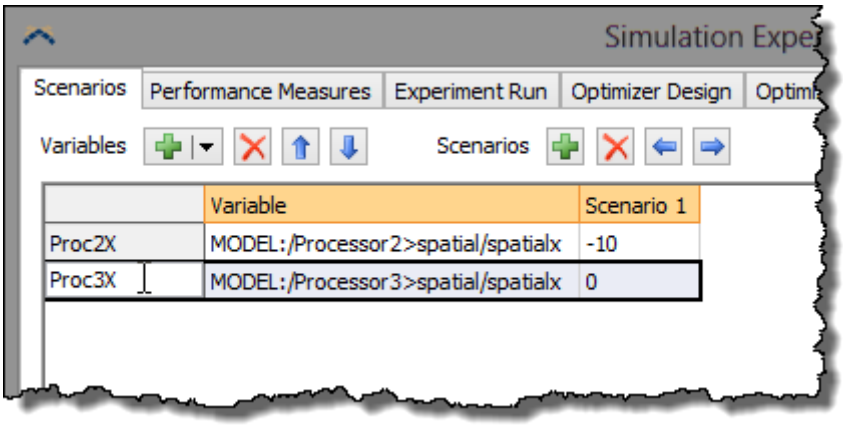
- Sample the X position field in the Quick Properties menu by clicking on it. This adds a new experiment variable.



- Set the value of Scenario 1 for the new variable by double-clicking the cell and entering the new value.



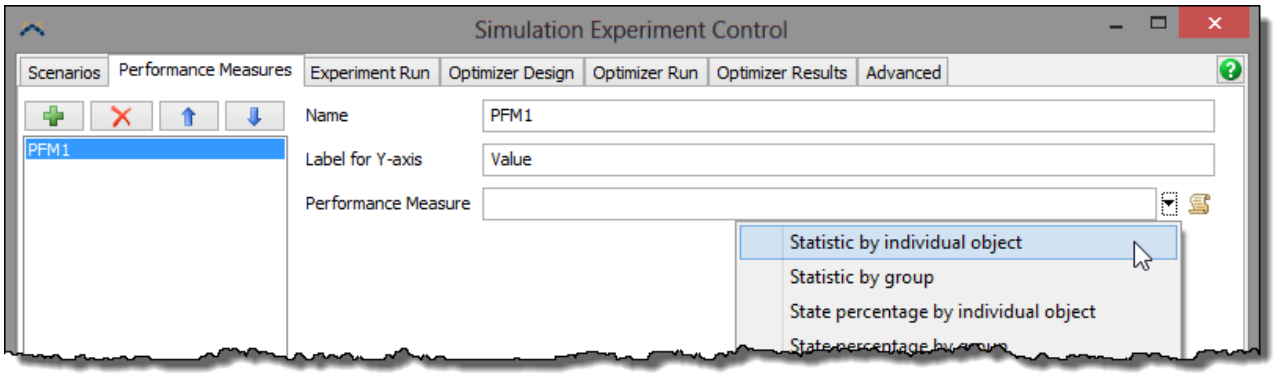
- Set the name of the variable by double-clicking on the current name. Set the name to Proc2X for Processor2 and Proc3X for Processor3.



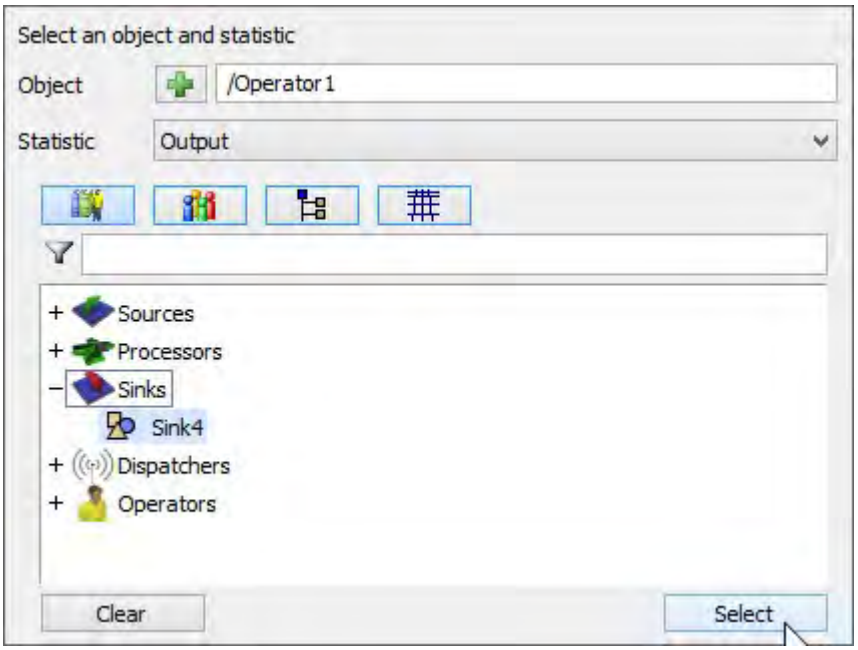
Creating Performance Measures

Go to the Performance Measures tab in the Experimenter window. From there:

- Click the  button to add a new performance measure.
- Name the new performance measure Throughput.
- Click the ▾ button and select the first option. A popup will appear.



- Select Sink4 for the object and Input for the statistic. To select the object simply type "/Sink4" (no quotes) in the object field or do the following:
    - o Click the  button. o Select Sink4 from the list of objects. o Click the Select button when you are finished.



Designing the Experiment

Now that we've created the variables and performance measures, we'll set up some scenarios for our experiment. Go back to the experiment tab:

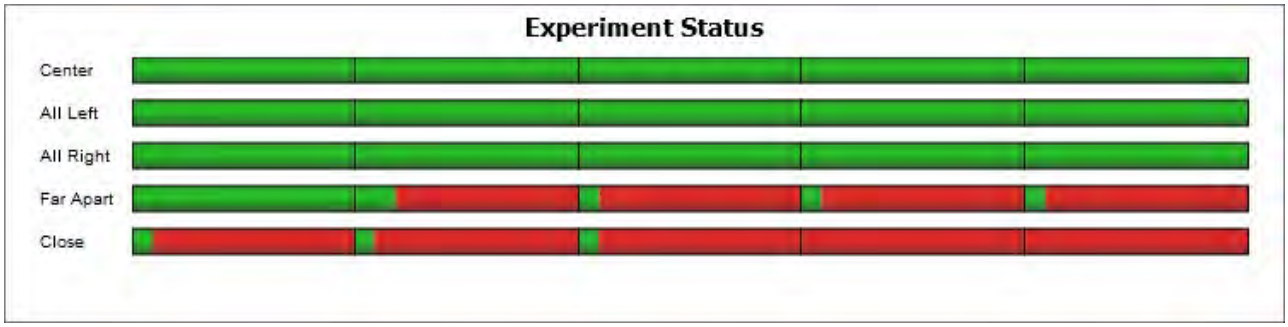
- Create 5 scenarios by clicking on the Scenarios  button 4 times.
- Enter scenario names and values as follows:

	Variable	Center	All Left	All Right	Far Apart	Close
Proc2X	MODEL:/Processor2>spatial/spatialx	-10	-13	-7	-13	-7
Proc3X	MODEL:/Processor3>spatial/spatialx	0	-3	3	3	-3

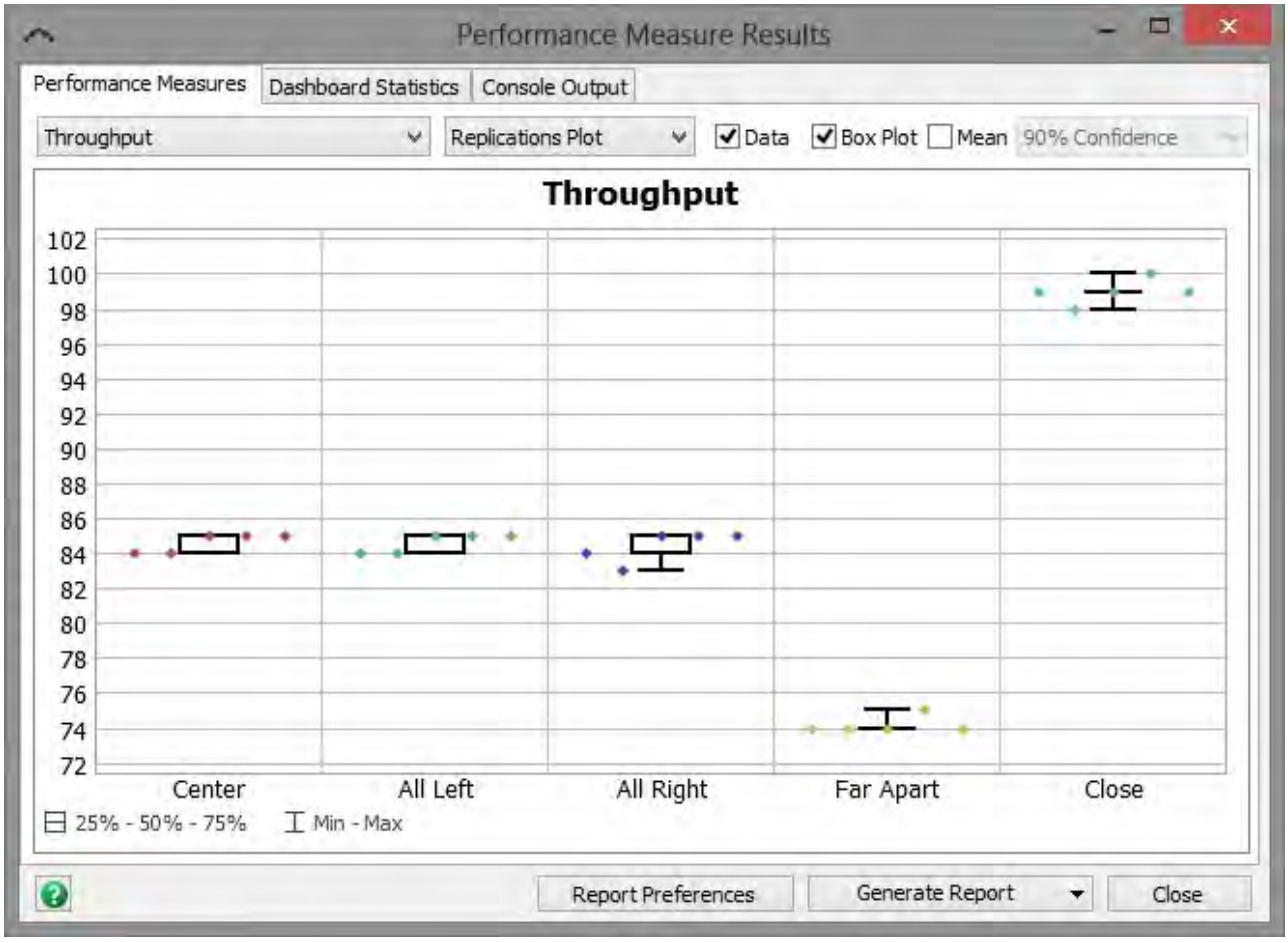
Running the Experiment

Go to the Experiment Run tab. Hit the 

button. Each scenario will be run 5 times and the results of the Throughput performance measure will be collected at the end of each run. The status window will show which scenarios/replications are currently being run. FlexSim will run multiple scenarios simultaneously if your computer has a multi-core cpu.



Once the experiment is finished, click the View Results button at the bottom. This will open a window where you can get data on the performance measures for the scenario. In this example we only have one performance measure, but if you had multiple you could see the results for each in this window. There are several options for how to display the data, including a Replications Plot, a Frequency Histogram, a Correlation Plot (for examining correlations between multiple performance measures), a Data Summary, and a Raw Data view.



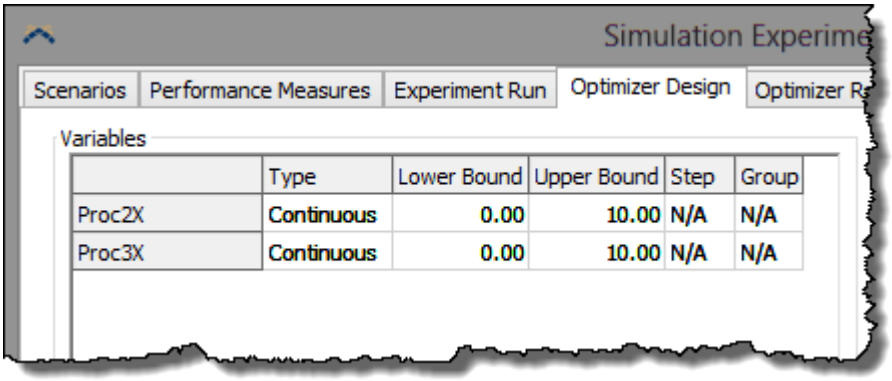
In this experiment, the best scenario was the "Close" scenario, which averaged right around 99 parts of throughput. The worst scenario was the "Far Apart" scenario, which averaged about 75 parts throughput.

Optimization

In addition to using the Experimenter to explicitly define scenarios, you can use the Optimizer. The optimizer will automatically create scenarios and then test those scenarios, trying to find a scenario that best meets an objective.

Designing the Optimization

Go to the Optimizer Design tab in the Experimenter window. You will see that the two variables created earlier are present; this is because the experimenter and optimizer share variables.

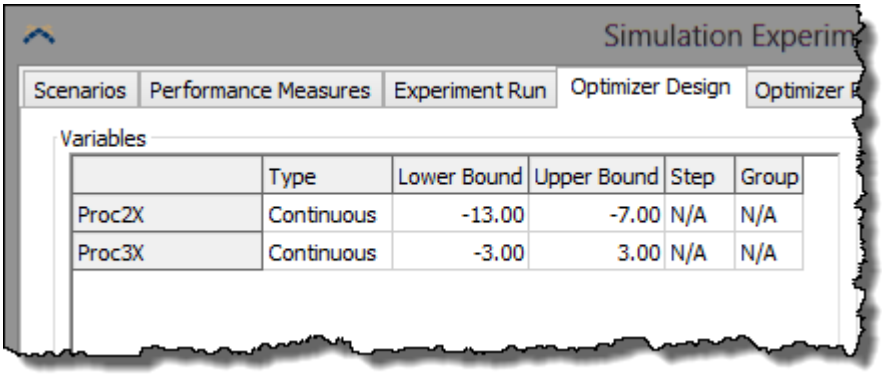


	Type	Lower Bound	Upper Bound	Step	Group
Proc2X	Continuous	0.00	10.00	N/A	N/A
Proc3X	Continuous	0.00	10.00	N/A	N/A

However, the optimizer needs additional information about those variables. Specifically, you must specify:

- Type - The type of a variable dictates what kinds of values are possible for a given variable. Continuous variables can have any value between the upper and lower bound.
- Lower Bound - The lower bound specifies the lowest possible value the optimizer can set for the variable.
- Upper Bound - The upper bound specifies the highest possible value the optimizer can set for the variable.
- Step - For Discrete and Design variables, the step specifies the distance between possible values, starting from the lower bound.
- Group - For permutation variables, the group specfies which set of permutation variables this particular variable belongs to.

For this optimization, we want to allow the processors to move three meters to either side. Since we are not limited to specific positions within that range, both position variables are Continuous. However, we need to set the lower and upper bounds of each variable. To edit values in the table, double-click on the cell of interest and enter in the new value. Enter in the values shown below:



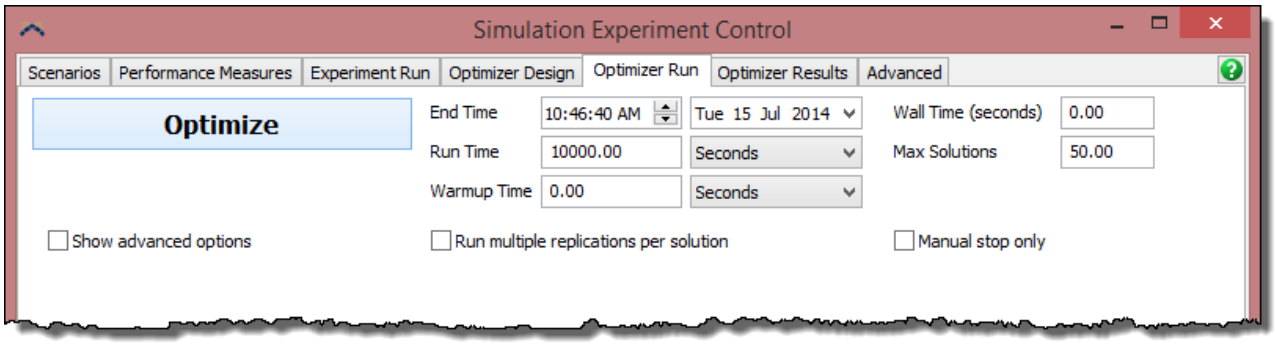
	Type	Lower Bound	Upper Bound	Step	Group
Proc2X	Continuous	-13.00	-7.00	N/A	N/A
Proc3X	Continuous	-3.00	3.00	N/A	N/A

The final design step is to set the objective function. The objective function is present, but blank. Edit its name to be Revenue. Then click on the function field. A  button will appear on the right side. Click on this button to bring up a list of all variables and performance measures. The objective function is a value derived from any or all of these values. Select Throughput; this will add that perfomance measure to the objective function, and put the cursor right and the end. Add the text * 500 so that Revenue is equal to Throughput * 500. Leave the direction on Maximize, because we want to maximize Revenue. Since we only have one objective, the search mode can remain on Single.

Step 3: Running the Optimization

Go to the Optimizer Run tab in the Experimenter window. Then:

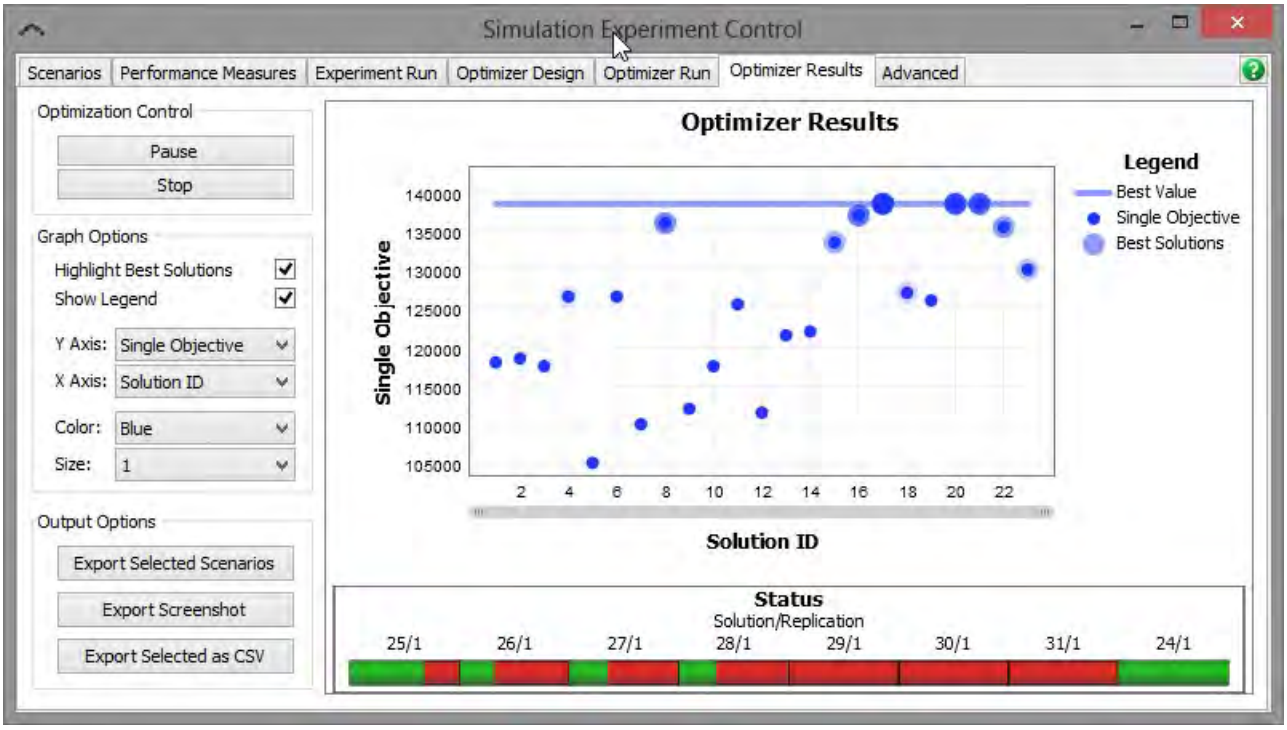
1. Set the Run Time to 10000. This is how long the optimizer will run each model configuration (called a solution) to evaluate it.
2. Set the Wall Time to 0. This usually means how long the optimizer is allowed to run in real time. The value 0 means it has no time limit.
3. Set Max Solutions to 50. This means the optimizer will try no more than 50 different solutions to find the optimal solution.
4. Click the Optimize button.



The Experimenter window will automatically switch to the Optimizer Results tab. The optimizer then begins running through the following loop:

1. Determine values for Proc2X and Proc3X.
2. Run a model with those values for 10000 seconds.
3. Evaluate the performance measures.
4. Calculate the objective function.
5. Rank this solution.
6. Use the information from this solution to create a new solution - new values for Proc2X and Proc3X.
7. Repeat from Step 2.

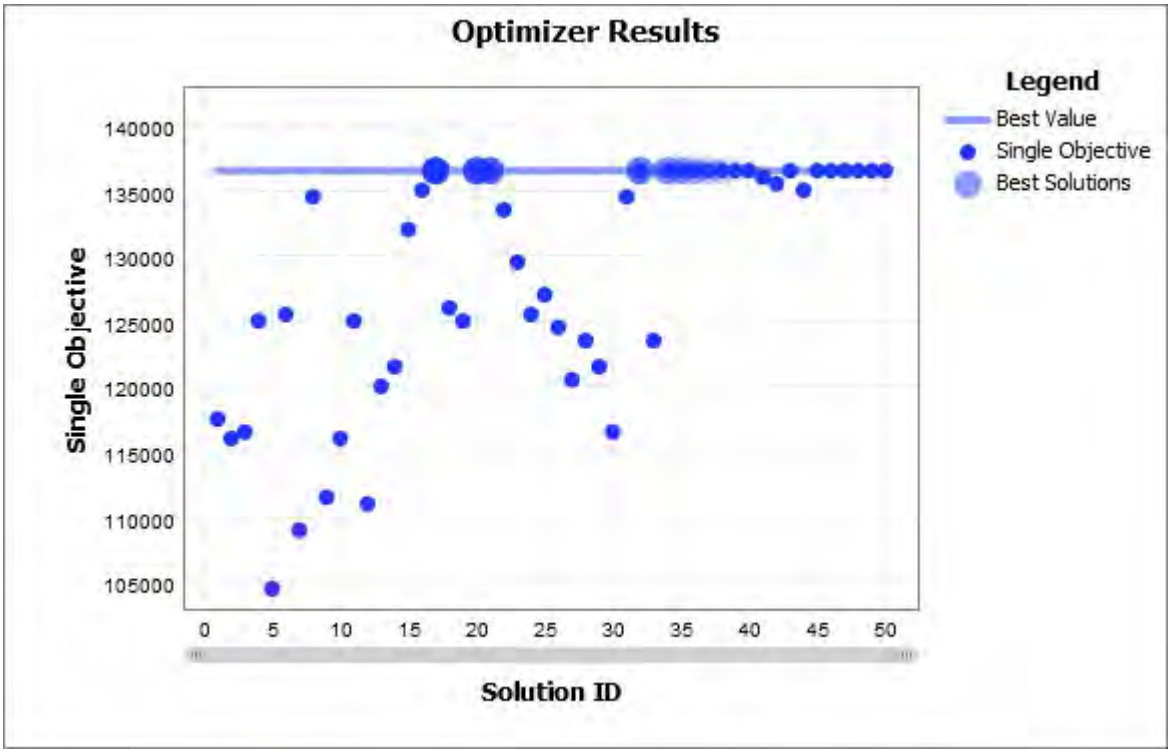
Because the optimizer shares the multi-threaded capability of the experimenter, it can evaluate multiple solutions at the same time. As the optimization progresses, the Optimizer Results graph will update and show the optimizer's progress.



Once the optimizer evaluates 50 solutions, a message will appear stating why the optimizer stopped. In this case, it will say that the optimizer reached the maximum number of solutions. If something went wrong, the message will contain information about the error.

Step 4: Analyzing the Results

When the optimization is finished, the Optimizer Results chart should look something like this:



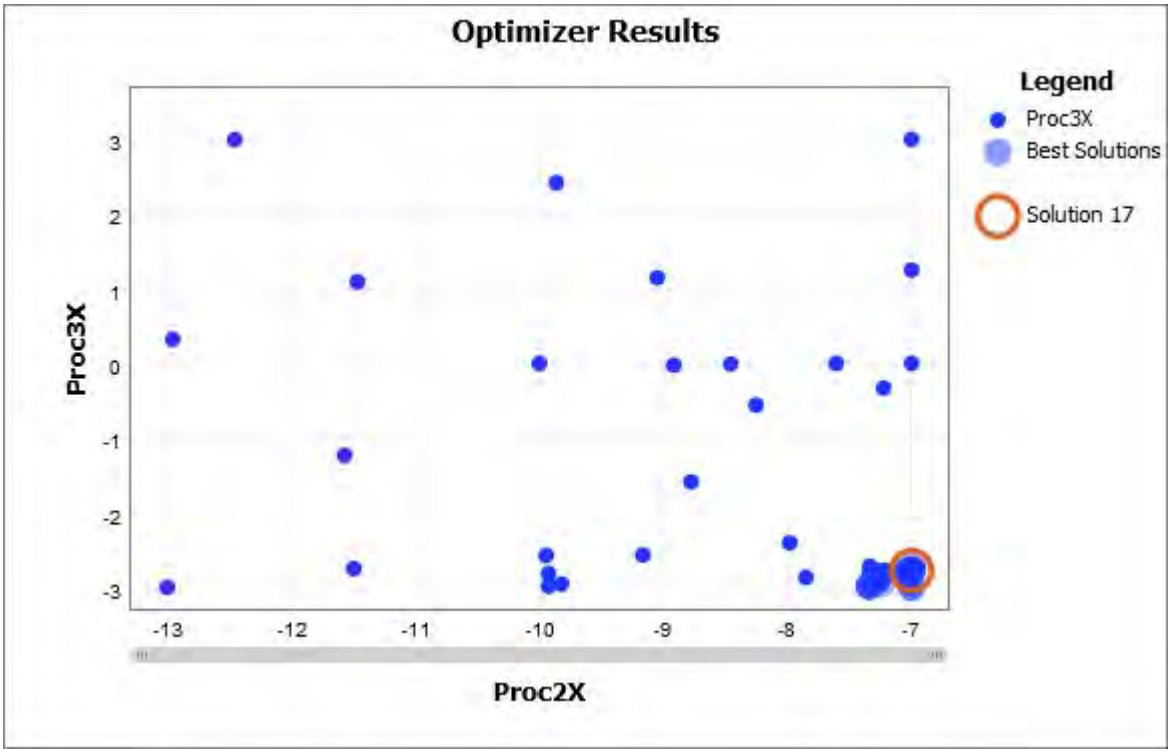
The Y Axis is called "Single Objective." For this example, it is synonymous with Revenue. The best solutions are highlighted. The circles with a lighter border around them represent better solutions. For a single objective, the top 10 solutions are marked this way.

As the optimization progressed, the optimizer got better and better at creating good solutions, so that the last 15 solutions were all very good. This is called convergence, and it is one way to tell if an optimization is finished; if the objective value has not improved for a while, it may be that it will not improve with further searching, and the current best solution should be used.

Answering the Original Question

The goal of this optimization was to figure out where to put the two processors. We can now very easily find the answer to this question. Hover over the best solution (the largest blue circle) on the chart; a small popup will appear.

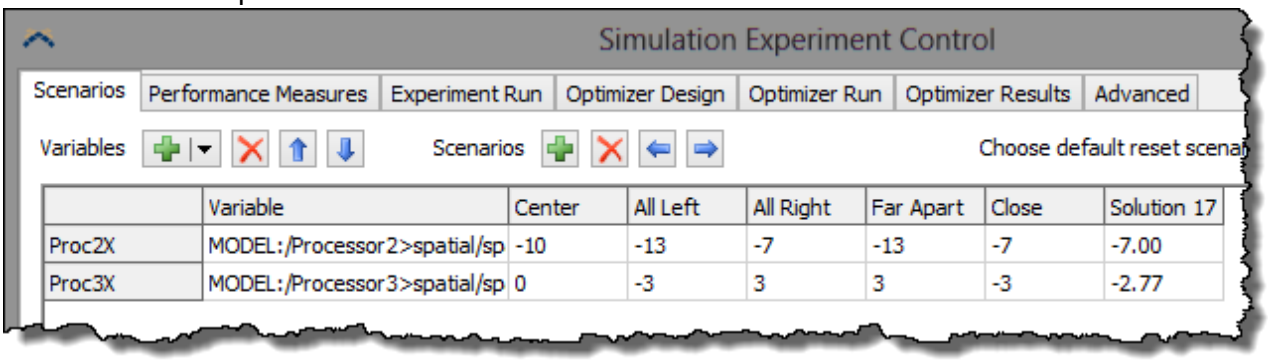
Click on this solution to select it. Now, in the Graph Options panel, change the Y Axis to Proc3X, and the X Axis to Proc2X.



The best solution (and all the other best solutions) is found where Proc2X is greatest, and where Proc3X is least. Remember that all top 10 solutions produced the same results; in this case, having the two processors right next to each other is the best configuration for this model.

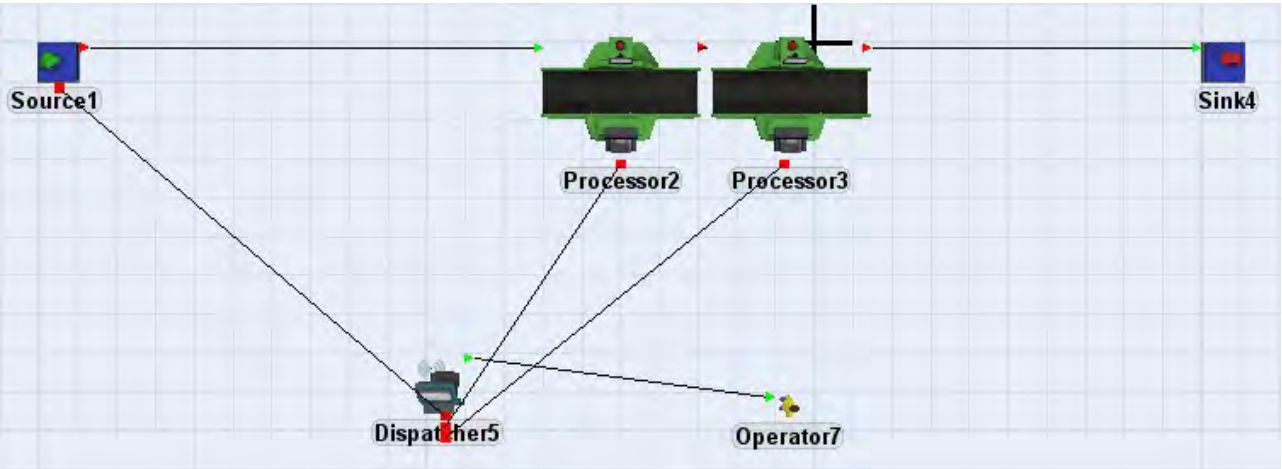
Setting the Model to the Best Solution

It can be very useful to set the model to match the best solution. To do this, click the Export Scenarios button. This takes all the selected solutions and creates experimenter scenarios for them. Go back to the Scenarios tab on the Experimenter window to view the new solution.



	Variable	Center	All Left	All Right	Far Apart	Close	Solution 17
Proc2X	MODEL:/Processor2>spatial/sp	-10	-13	-7	-13	-7	-7.00
Proc3X	MODEL:/Processor3>spatial/sp	0	-3	3	3	-3	-2.77

Now, from the "Choose default reset sceanrio" drop-down on the far right, select the new scenario. Then reset the model to apply those values to the 3D model.



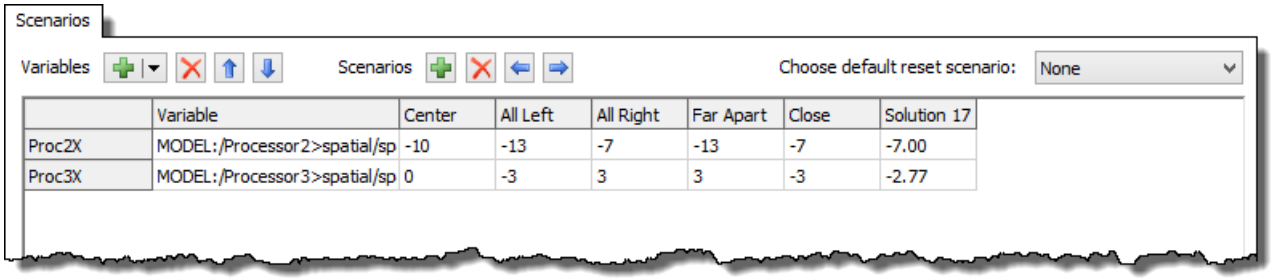
Experimenter / Optimizer Reference

Pages

- Scenarios
- Performance Measures
- Experiment Run
- Performance Measure Results
- Optimizer Design
- Optimizer Run
- Optimizer Results
- Advanced

Scenarios

The Scenarios tab is where you define the variables and scenarios associated with an experiment.



	Variable	Center	All Left	All Right	Far Apart	Close	Solution 17
Proc2X	MODEL:/Processor2>spatial/sp	-10	-13	-7	-13	-7	-7.00
Proc3X	MODEL:/Processor3>spatial/sp	0	-3	3	3	-3	-2.77

Variables

 - Add an experiment variable. Select the variable type from the drop down. -  Remove the selected variables.

 - Move the selected variables up or down in the list.

Once variables have been added, you can change them by clicking in the "Variable" cell for that variable. The  buttons will appear. Use these to change the variable, or type in a path manually. Define the name of the variable in the header column of the table on the left side.

Scenarios

 - Add a scenario. Once added, enter the value for each variable in that scenario's column in the table.

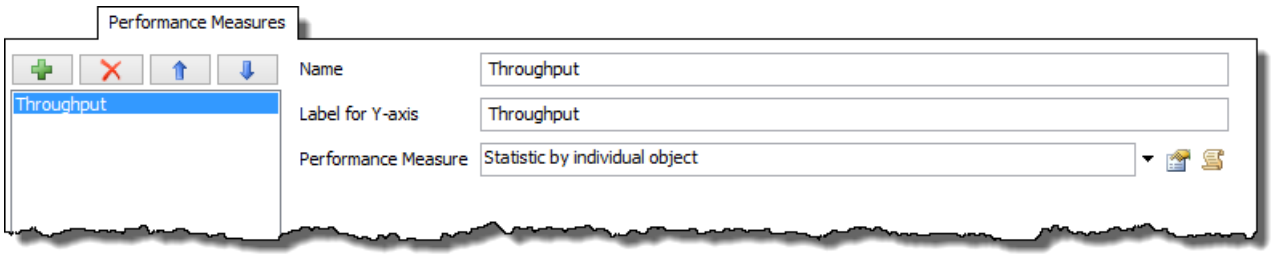
 - Remove the selected scenarios.

 - Move the selected scenarios left or right in the list.

You can rename the scenarios by changing the name of the table's column header.

Choose Default Reset Scenario - This allows you to define a scenario for your model to go to when you reset your model outside the experimenter. Once chosen, whenever you reset, the model will go back to that scenario. Choose None if you don't want the model to set its scenario when reset.

Performance Measures



 - Add a Performance Measure. If there are dashboards in the model, you can select a statistic from a dashboard as your performance measure.

 - Remove the selected performance measure.

 - Move the selected performance measure up or down in the list.

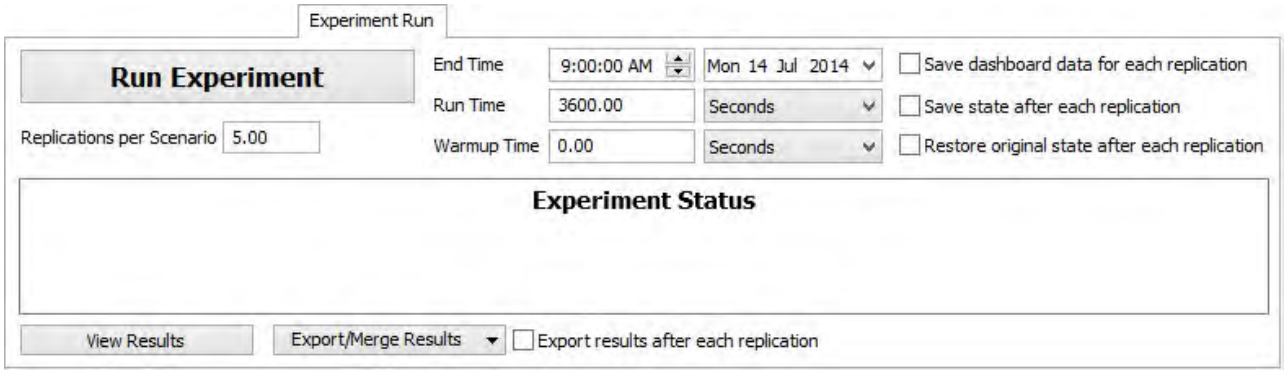
Select a performance measure from the list to edit it. If it is a standard performance measure, controls will display on the right (as seen in the above image). If the performance measure is associated with a dashboard statistic, a Properties button will appear. This will open the dashboard widget's properties window.

Name - The name of the performance measure will be used in the Performance Measure Results window and can be used in the Optimizer Design tab when defining Constraints and Objectives.

Label for Y Axis - Defines the Y axis title in the Performance Measure Results window.

Performance Measure - This picklist specifies what information will be gathered. Choose from the available picklist options or write your own custom code.

Experiment Run



The Experiment Run tab is where you define parameters for the experimenter and run it.

Run Experiment - Starts the experiment run.

Replications Per Scenario - The number of replications that will be run for each scenario.

End Time - The date and time the simulation will end. Based upon the Model Start Date and Time as defined in Model Settings.

Run Time - The total simulation time that each experiment will run to.

Warmup Time - The simulation time that each replication will run to before resetting their statistics. Statistics will thus only be collected for the time period (Run Time - Warmup Time).

Save Dashboard Data for Each Replication - If checked then at the end of each replication FlexSim will save the data for each dashboard statistic in the model so they can later be viewed as part of the results.

Save State After Each Replication - If checked, each replication's full simulation state will be saved to a file at the end of the replication. This allows you to open the replication in the state where it finished. This can be especially useful if one of your replications fails to give valid results.

Restore Original State After Each Replication - If checked, FlexSim will completely reload the model between execution of each model. You might check this box if your model doesn't properly reset to the same exact state every time you reset, and you don't want that "spill-over" state affecting subsequent replication results. However, because it is completely reloading the tree, it may increase the time it takes to run each replication.

Export results after each replication - Saves results to a file after each replication.

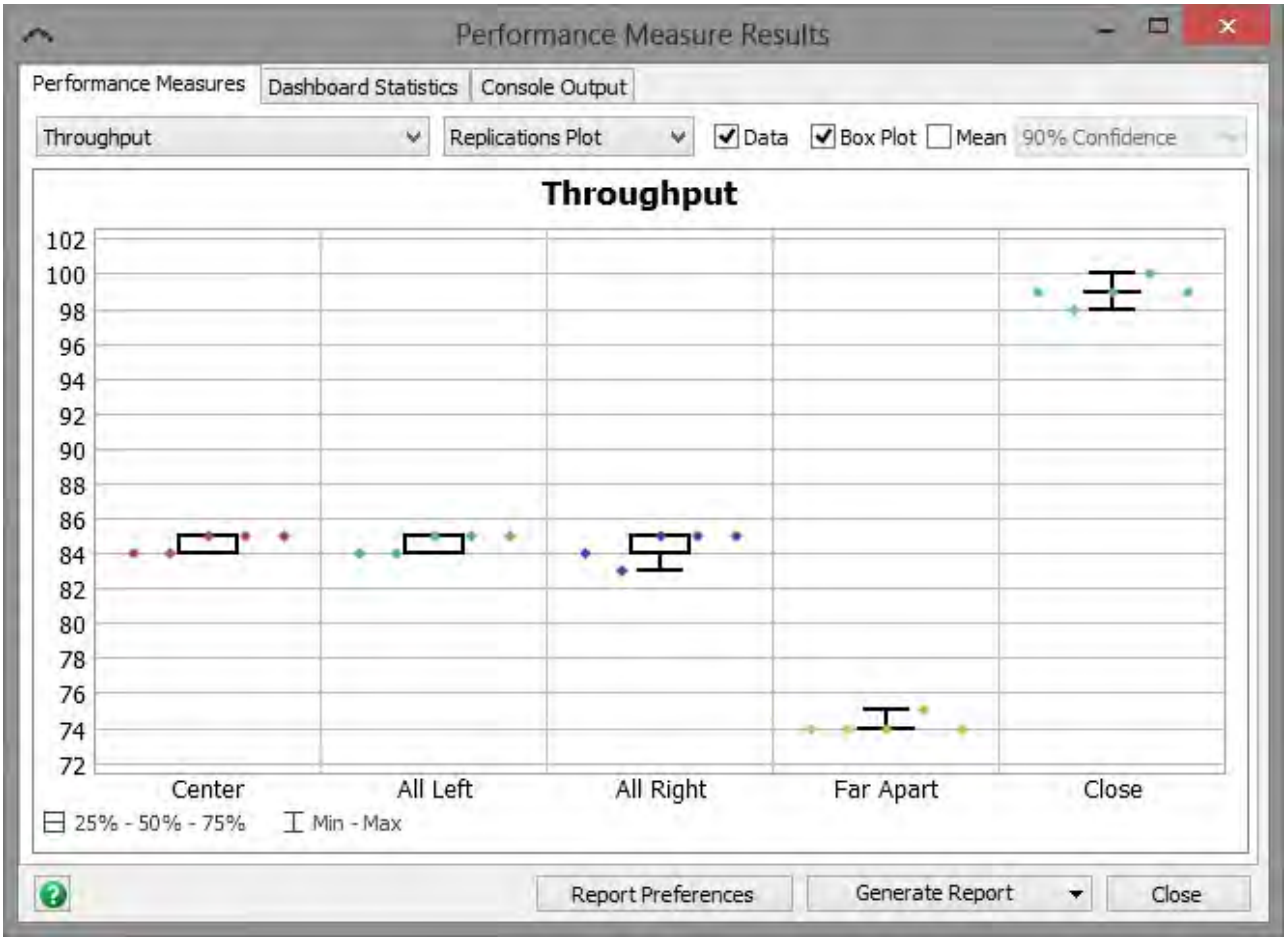
View Results - Opens the Performance Measure Results window. Click this button once an experiment run has finished.

Export / Merge Results - This button allows you to save your result to a file, as well as load / merge results from a saved file into your current results.

- Export will save the results to a .t file.
- Load will load the results from a saved file into your model, replacing any current results.
- Merge will load results from a file, and then merge those results with your model's current results.

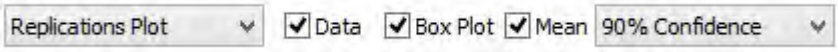
Export results after each replication - Saves results to a file after each replication.

Performance Measure Results



Performance Measures Tab

This tab allows you to compare performance measures between the different scenarios. Select the Performance Measure and graph type. The graph types available are: Replications Plot, Frequency Histogram, Correlation Plot, Data Summary, and Raw Data. Replications Plot

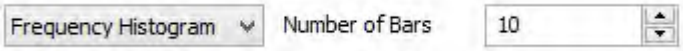


Data - If checked, the replications plot will plot the data points for all replications.

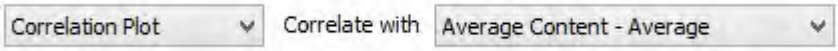
Box Plot - If checked, the replications plot will display a box plot.

Mean - Check the box to display the mean confidence interval for each scenario. You can specify the percentage as 90%, 95% or 99% Confidence.

Frequency Histogram



Number of Bars - Specifies the size of range to display. Correlation Plot



Correlate with - Select another performance measure to plot the current performance measure against.

Data Summary

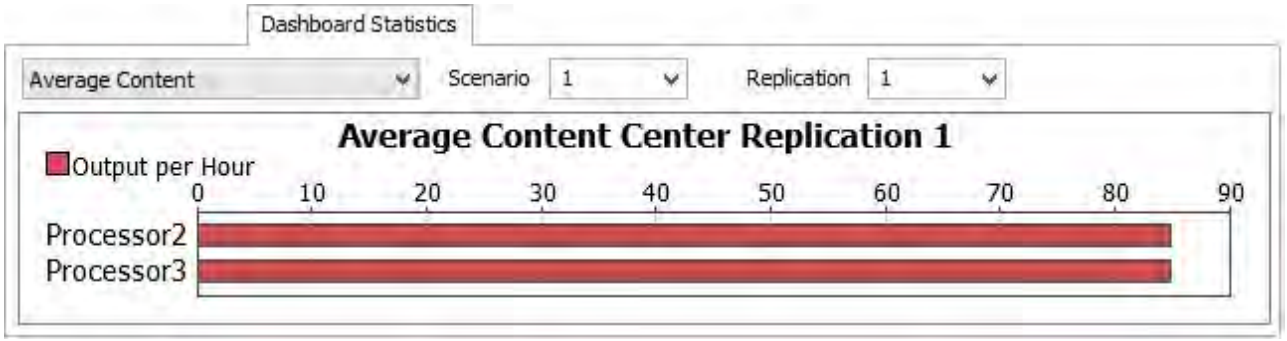
Mean Based on - Displays values based on the selected confidence interval.

Report Preferences - Opens the Performance Measures Report Preferences window. This allows you to prune what data you want included in the report.

Generate Report - There are two options for generating reports:

- Report Format This creates an html file with full results for all performance measures.
- Web Viewer Format This creates an html file that is interactive. When you open the html file you can choose which graphs you'd like to view.

Dashboard Statistics Tab

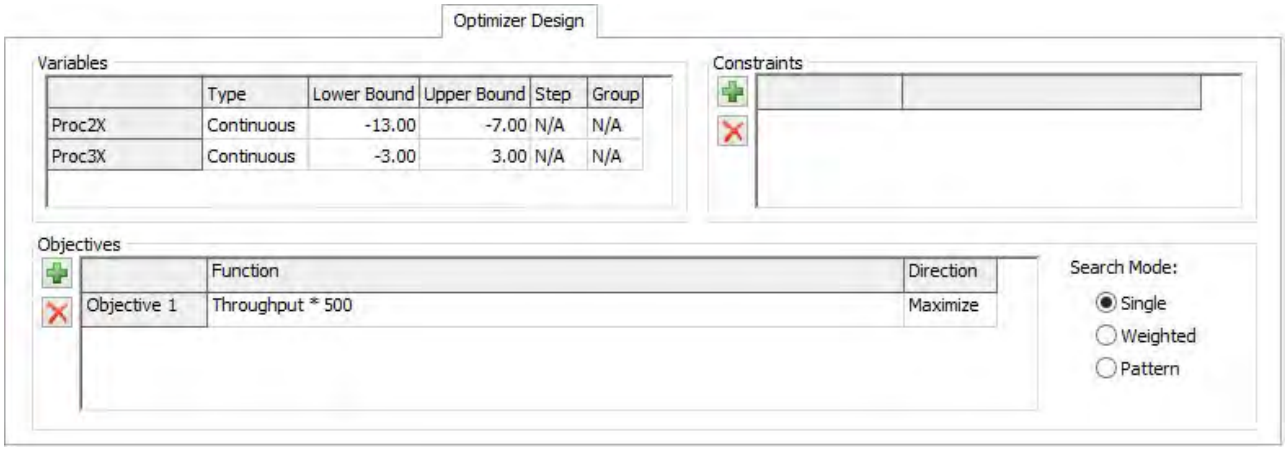


If you have have checked the box *Save Dashboard Data for Each Replication*in the Experiment Run tab before running the experiment, then you can go to the Dashboard Statistics tab to view individual replication results for various dashboard statistics. Choose the statistic, replication and scenario, and the statistic's associated graph will be shown.

Console Output Tab

FlexSim also saves the text output of the Output Console and the System Console for each replication. This can be used for debugging/analysis of each replication. Choose Output or System Console and the scenario and replication you want to view.

Optimizer Design



Variables

Variables Table - This table displays the variables that are listed in the Scenarios tab. Click in one of the Type cells to display a ▾ to change a variable's type. Setting the type will change the cells of the table to reflect which options are applicable.

Constraints

- Add a constraint.

- Remove the selected constraint.

Constraint Table - This table displays all of the constraints for the optimizer. Define an equation using the available variables, boolean and mathematical operators.

Objectives

- Add an objective.

- Remove the selected objective.

Objectives Table - This table displays all of the objectives for the optimizer. Define a function using the available variables, boolean and mathematical operators.

Search Mode
There are three search modes:

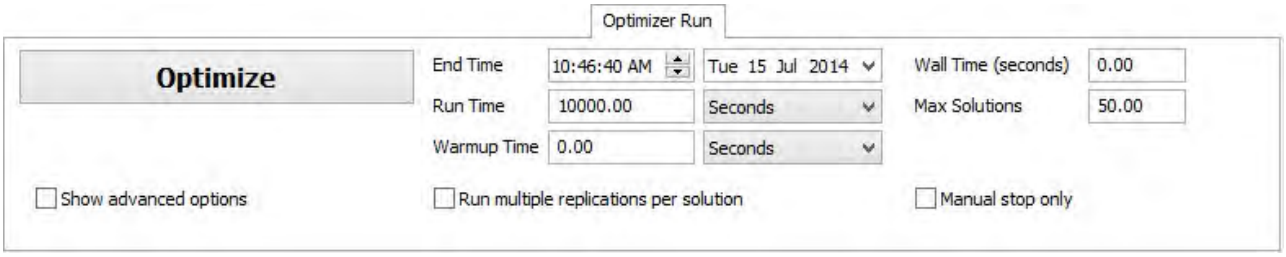
- *Single* - The optimizer will try to maximize or minimize a single objective. The best solution found in the time given will be returned.
- *Weighted* - The optimizer will try to maximize the cumulative value of all given objectives. Each objective is given a weight and a direction. If a particular objective is to be minimized, its value is multiplied by -1 before it is added to the cumulative objective. The best solution found in the time given will be returned. Selecting weighted will change the objectives table to display the following:

	Function	Weight	Goal	Direction	Lower Bound	Upper Bound
Objective 1		0.00	0.00	Maximize	0.00	0.00

- *Pattern* - The optimizer will search for solutions that optimize all given objectives. This search mode returns a set of optimal solutions. They are optimal in that for each one, improving any objective worsens the others. For example, a given optimization may maximize profit and minimize cost. For a solution to

be optimal, no other solution can exist that increases profit without increasing cost. This is also called "Pareto Optimal."

Optimizer Run



Optimize - Begins the optimization process.

End Time - The simulation date and time each solution will end. Based upon the Model Start Date and Time as defined in Model Settings.

Run Time - This is the time in model units that each solution will run.

Warmup Time - The simulation time that each solution will run to before resetting their statistics. Statistics will thus only be collected for the time period (Run Time - Warmup Time).

Wall Time (seconds) - This is the maximum real time the optimizer will spend generating solutions. Once the wall is hit, the active solutions will finish running and then the optimization run will stop.

Max Solutions - The maximum number of solutions the optimizer will generate and test.

Run multiple replications per solution - If checked, the optimizer will run multiple replications for each solution that is generated.

Manual stop only - Sets the optimizer to run until the user stops it.

Advanced Options

Check Show advanced options to display the following:

Use scenarios as possible solutions - Use manually-defined scenarios (from the Scenarios page) as initial search points. This gives the optimizer a place to begin it's optimization process and generally shortens the optimization time.

Save dashboard data for each replication - Check to save data from dashboard statistics at the end of each replication for post-optimization reports/analysis.

Save state after each replication - Check to save the full state of the model to a file at the end of each replication.

Restore original state after each replication - Check to restore a clean copy of the model after each replication.

Evaluate solutions in order generated - Check to ensure solutions are evaluated in the order they are generated. This is slower, but ensures that the optimization search is repeatable.

If *Run multiple replications per solution* is checked, the following options are made available:

Minimum number of replications - The minimum number of replications that will always be performed for each solution.

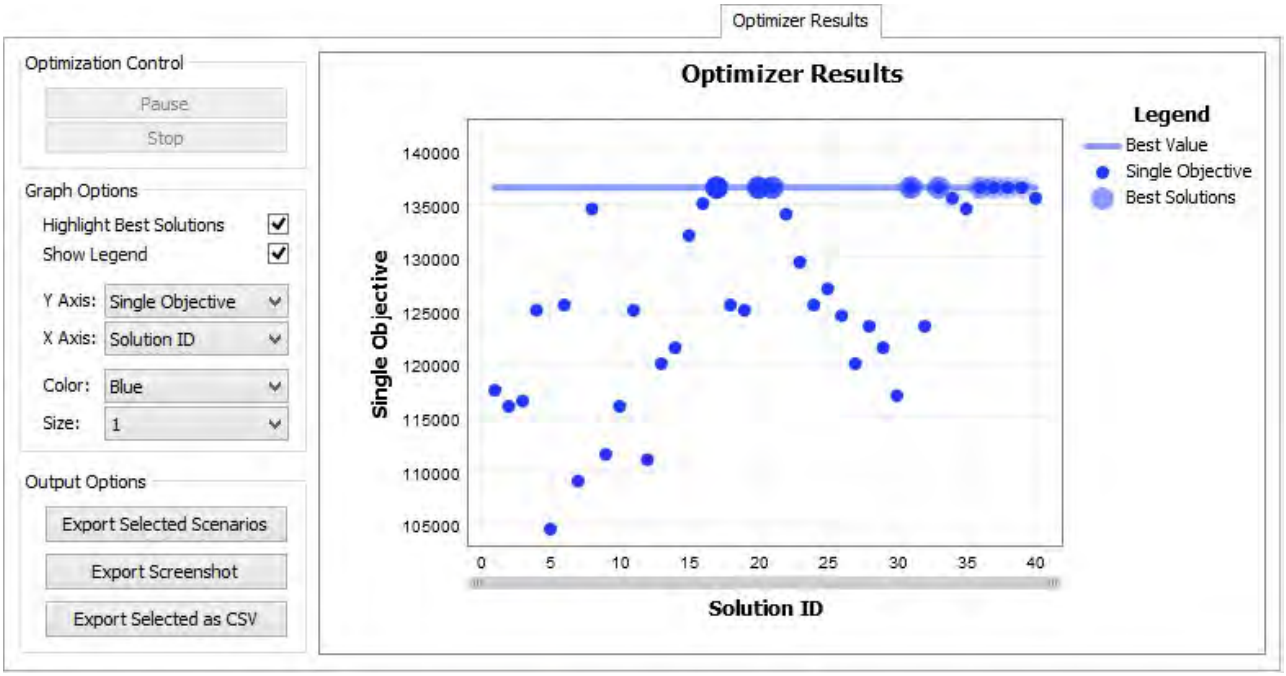
Maximum number of replications - The maximum number of replications that will always be performed for each solution. If set to -1, no maximum will be used.

Run replications until - Each solution will run replications until the chosen condition is met or the maximum number of replications is reached (if not set to -1).

Percent confidence - Percent confident that the solution falls within the specified error percent of the mean value of the replications.

Error percent - Determines the range about the mean for which the percent confidence applies. Must be a value between 0 and 100.

Optimizer Results



Optimization Control

If the optimizer is currently running through scenarios, you can pause or stop the optimization process.

Graph Options

These options allow you to customize the look of the graph and define what values to plot against the X and Y axes.

Output Options

Export Selected Scenarios - This option will take the selected scenarios (click on a scenario in the graph to select) and export their data to the scenarios table of the Scenarios tab. This allows you to either run further experiments on the chosen scenarios, or to set your model to one of the selected scenarios by using the *Choose default reset scenario*.

Export Screenshot - Saves the currently displayed results graph as a PNG file.

Export Selected as CSV - Exports the data from the selected scenarios to a CSV file.

Optimizer Results Graph

Optimization Control - If the optimizer is currently running through scenarios, you can pause or stop the optimization process.

Advanced

For more information see the Experimenter Picklists.

Repeat Streams of Replication - Specify a replication to set the manual model runs (not experimenter replications) to use the same random number streams. You must be repeating random streams for this to take effect.

Preparing a 3D File

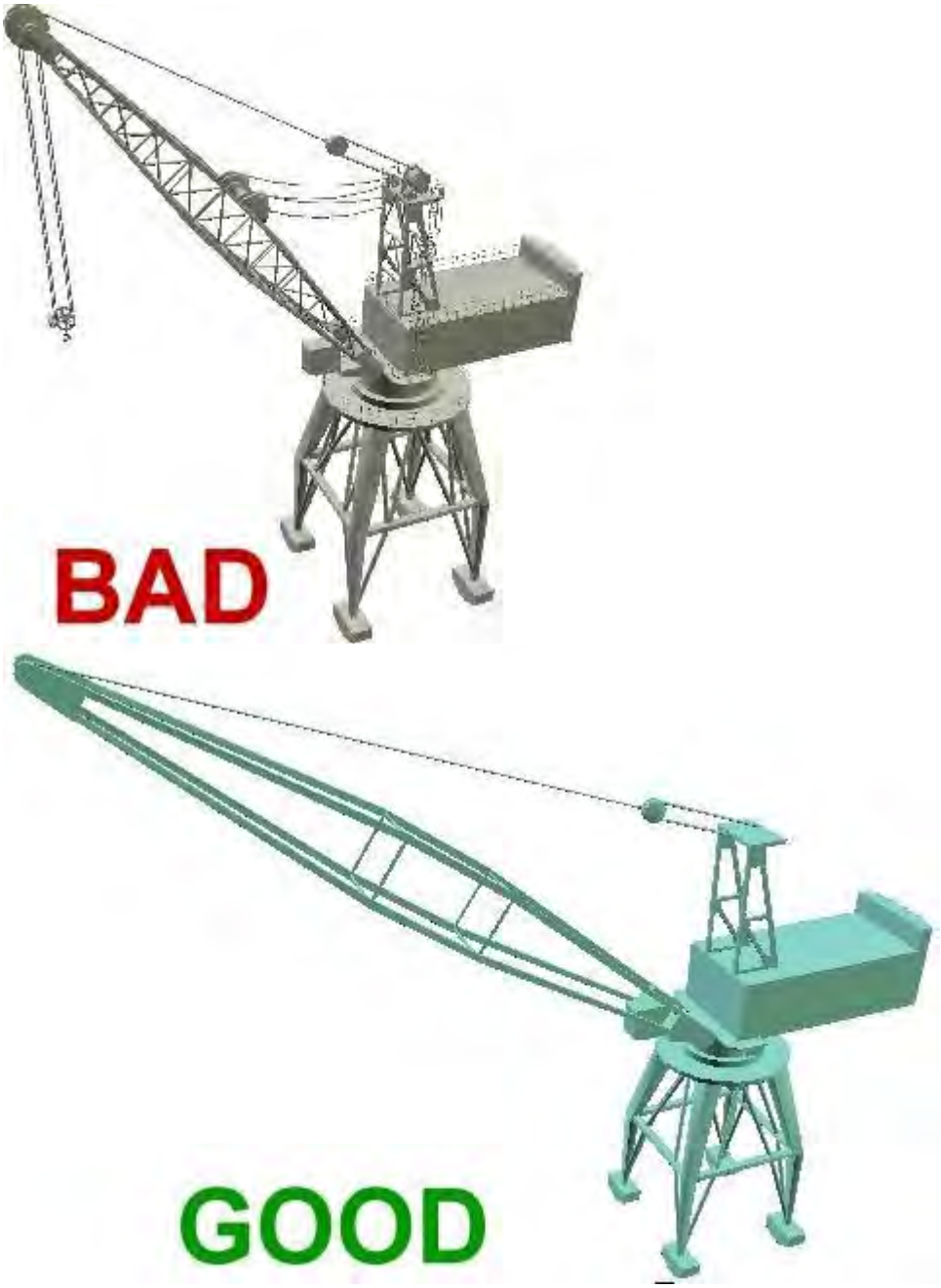
We suggest using AC3D for 3D media creation. A license is relatively cheap and AC3D has a lot of capability coupled with a decent UI that's pretty quick to learn. Another large advantage is, FlexSim is capable of directly importing the raw .ac files created from AC3D. You can get AC3D from www.inivis.com. If you're not using AC3D, we suggest exporting media to the 3ds file format. The 3ds file format allows for very fast drawing speeds and fast load speeds. With several 3D formats, including .3ds and .ac files, you can:

- Have the FlexSim object's color show through on certain parts of the shape
- Use the FlexSim object's defined texture show through on certain parts of the shape
- Auto-scale the FlexSim object based on 3D file dimensions

The following steps should be performed when preparing a 3D file for import:

Reduce the Number of Polygons

3D 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. Also, where possible you should consolidate objects in your 3D creation software and share the same texture among multiple objects. This reduces the number of OpenGL state changes required, which can significantly improve performance.

Transparency

Transparency in FlexSim is simple. It is just a matter of using textures that have transparency in them. You'll need to create your textures with a file type that supports transparency like a PNG file. FlexSim will read the transparency of the image and display it properly.

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 in FlexSim that the file is imported into.
3. Use an xrl file for wrl files.

If you are using 3DS or .ac, then you can just make sure the object is scaled properly in whatever 3D software you use. Then import it into FlexSim, and it will automatically import to the correct size. Just make sure that your units in your software are the same as the units you are going to use in FlexSim.

Getting the Object's Color to Bleed Through

For .3ds and .ac files, you can have the FlexSim object's color bleed through on certain shapes in the file. You can do this in one of two ways: (1) give the shape an ambient color value of rgb: (0.235,0.235,0.243) in your 3D software, (2) append "_fsclr" to the end of the shape's material name in your 3D software.

Getting the Object's Texture to Bleed Through

For .3ds and .ac files, you can also have the FlexSim object's defined texture show up on the shape instead of the texture defined in the file. To do this, just add a texture named "fstx.png" to the object in your 3D software.

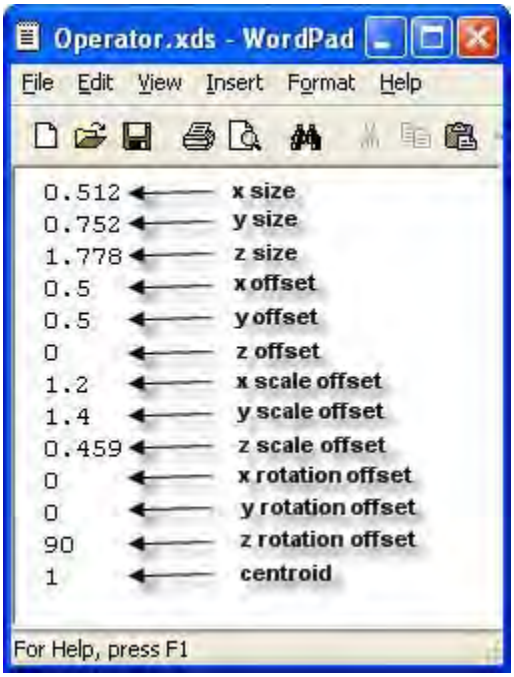
XRL Files

xrl files are used to make imported wrl shapes conform to the object they are imported into. xrl files must have the same names as the objects they modify. For example:

- crane.wrl – crane.xrl

xrl file makeup

The 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.



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.

FlexSim can import several types of 3D media. These include:

- .wrl, .3ds, .dxf, .stl, .skp, .dae, .obj, .ac, .x, .ase, .ply, .ms3d, .cob, .md5mesh, .irr, .irrmesh, .ter, .lxo, .csm, .scn, .q3o, .q3s, .raw, .off, .mdl, .hmp, .scn, .xgl, .zgl, .lvo, .lvs, .blend, .stp, .igs

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. You can import media using two methods:

1. The most common method is to import media by selecting a 3D file from an object's General Properties of the Quick Properties window.
2. Through the object's General tab page.

The following will help to ensure media is imported correctly:

- Shape Factors
- Letting Color Show Through the 3D Shape
- Level of Detail

The following steps should be performed when preparing an AutoCAD .dxf or .dwg 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:
    1. Measure a known distance in AutoCAD ("_dist").
    2. Apply the following equation: scale factor = FlexSim distance / ACAD distance.

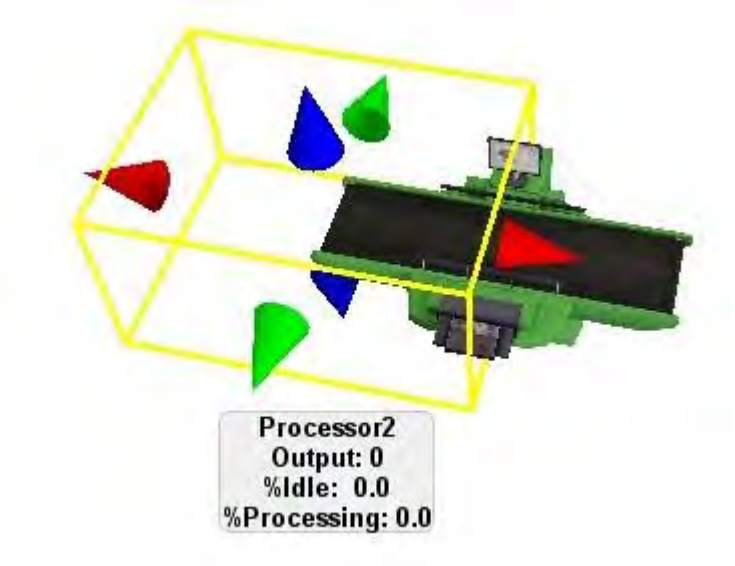
    To scale objects in AutoCAD follow these steps:

    3. Select the objects you want to scale.
    4. Type "_scale" in the command prompt or select the scale command from the menus.
    5. Specify a reference point.
    6. 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:

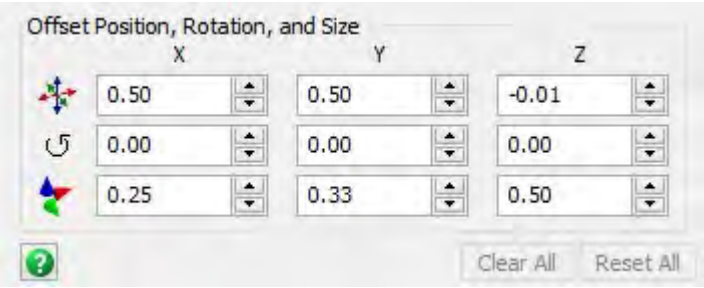
1. Drag a Background object into the model from the Library window under Visual.
2. Follow the steps in the Background Drawing Wizard.

Each media file that is imported has certain scaling and offset settings which may cause the 3D shape that you import to not fit within the object's boundaries. If this is case, you can edit the object's 3D shape factors to fit the 3D shape within the object's yellow bounding box.



The image above shows modified shape factors for a Processor's 3D shape. Notice that the yellow bounding box reflects the true position and size of the processor, but the 3D shape has been offset in the x direction.

Shape factors may be accessed through an object's General tab page.



: Change the position of the 3D shape within the bounding box.

: Change the rotation of the 3D shape within the bounding box.

: Change the size of the 3D shape within the bounding box.

Clear All / Reset All - When a shape frame is added to an object, it is given shape factors so that the shape will display inside the yellow box of the object. Clearing the shape factors from a shape frame will cause the shape frame to draw without regards to the object's yellow box, or the actual size of the 3D shape. Resetting the shape factors will reapply the originally calculated shape factors. By selecting the object's base frame, you can clear or reset all shape frame's shape factors. Or you can select an individual shape frame and clear/reset the shape factors for just that frame.

You can animate an object or flowitem using 3D frames. This is done by creating multiple 3D shapes. Then within FlexSim you can tell an object which shape frame to display. This is useful if you want to change the look of an object like a Flowitem as it progresses through a model. An example of this might be a bottling line. The bottle starts out empty, then is filled and then a cap is placed on top. This could be displayed by using three different 3D shapes.

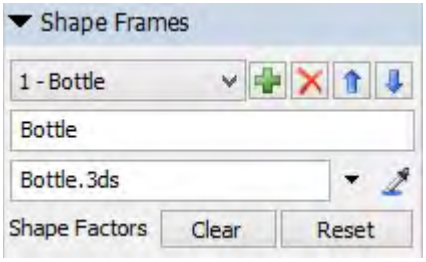
In versions of FlexSim prior to 7, shape frames had to be specified by the filename of a 3D shape. Now, shape frames are assigned manually by:

1. Adding a Shape Frame to the object
2. Setting a name for the frame
3. Specifying the 3D shape path

Once added, shape frames can be changed either by name, or index. See the Commands below. Shape frames can be modified in two places:

- In the Quick Properties window under Shape Frames (see below).
- In the General tab page of an object's properties window.

Quick Properties



The shape frames section of the Quick Properties window does not always display for objects. For a FixedResource, TaskExecuter, or other non-flowitem object, the Shape Frames section will only show up if that object already has shape frames assigned to it through the General tab page.

Flowitems however, always have the Shape Frames section displaying. This is because it is far more common to change the shape of a Flowitem as it changes through the model run, than it is to modify a Processor or Operator.

Shape Factors - When a shape frame is added to an object, it is given shape factors so that the shape will display inside the yellow box of the object. Clearing the shape factors from a shape frame will cause the shape frame to draw without regards to the object's yellow box, or the actual size of the 3D shape. Resetting the shape factors will reapply the originally calculated shape factors.

Commands

To change an object's shape frame or to query an object to find its current shape frame, you'll need to use the following commands:

```
setframe(current, 1); setframe(current,
"Bottle2"); int curframe = getframe(current);
```

For convenience, object triggers like the OnReset, OnMessage, OnEntry/OnExit, OnLoad/OnUnload, etc. have a preset for changing an object's shape frame.

LOD files are 3D shapes that get progressively simpler and 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.

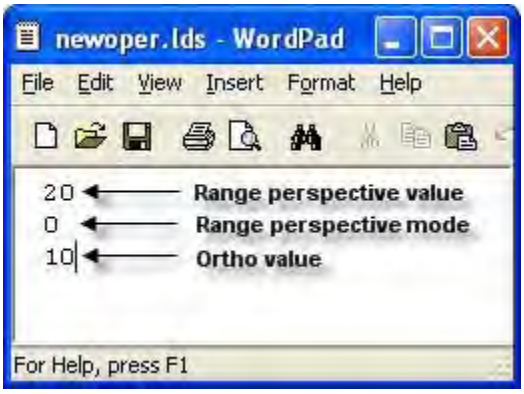
To import an LOD shape just follow the steps used in the section on Importing 3D Media. Use the base object for the name of the file that you select.

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.

This topic describes advanced mechanisms for storing undo history. This is meant mostly for developers of custom user interfaces and libraries.

Topics

- When Advanced Undo is Needed
- Creating an Undo Record Explicitly
- Ignoring Undo
- Undo Tracking on a Custom Object's Drag

When Advanced Undo is Needed

In FlexSim, for the most part, undo/redo just works. You don't need to worry much about what is being recorded to the undo history, because it's all automatic. However, there are some instances where you might need to customize the undo history recording to suit your needs. Some examples of when you might want to customize undo history recording are:

- When you have created a customized object, and that object changes certain variable values as the object is dragged around in the model, i.e. in its OnPreDraw or OnDrag functionality. You may want those values to go back to their original state before the object was dragged.
- If you have custom functionality in a user library's drop script that you do NOT want recorded in the undo history. By default, whenever the user does a drag/drop operation from the library icon grid, FlexSim will automatically start an "aggregated undo" operation, which essentially retools several lowlevel commands to record changes that are made. We'll talk about this in more detail later, but for some dropscripts, you may need to temporarily "turn off" undo recording, then execute your functionality, then turn it back on. For example if you are performing functionality that opens a window.
- If you create a tool window that is used to modify the model, like the Edit Selected Objects tool windows, then you'll need to add code to the buttons of those windows that explicitly does undo recording. In FlexSim, all things that are done in a 3D view, a tree view, or a table view, as well as all operations done in tool windows that focus around those views, are undo-able, so if you design a tool window that is used with one of these views, you'll need to make it undo-capable.

Creating an Undo Record Explicitly

To create an undo record before executing a piece of code, call beginaggregatedundo(). Pass in the view that this undo is associated with. If you pass in NULL as the view, then it will be associated with FlexSim's global undo history (MAIN:/project/undo/). Pass in a description of the operation being done ("ChangeColor"). The beginaggregatedundo() command will return back a unique id for the undo record that you've created. Then execute the code that you want to be undo-able, then call endaggregatedundo(), passing in the view and the undo id.

When you call beginaggregatedundo(), FlexSim will retool several low-level commands, so that subsequent calls that you make to those commands, or calls to those commands by other commands that you call, will record changes to the undo history. The commands that are retooled are as follows:

- nodeadddata()
- setnodenum()/set()
- setnodestr()/sets()
- nodedeldata()
- nodeinsertinto()
- nodeinsertafter()
- nodejoin()
- nodepoint()
- nodebreak()
- setname()
- clearcontents()
- createcopy()

- createinstance()
- switch_cppfunc(),switch_flexscript()... (all switch_... commands)
- transfernode()
- transfernodeobj()
- moveobject()
- setrank()
- destroyobject()
- destroynode()

Ignoring Undo

If you are inside of an aggregated undo and you want FlexSim to ignore a set of commands, call beginignoreundo(), execute the commands, then call endignoreundo(). This is especially useful in dropscripts of a user library. FlexSim automatically creates an aggregated undo record that applies to all functionality executed as part a drag/drop operation from the library icon grid, so all code that executes in a drop script is being recorded as part of an undo record. To bypass this for certain functionality, surround it with beginignoreundo() and endignoreundo()

Undo Tracking on a Custom Object's Drag

You may also want FlexSim to track changes to a custom object's variable value, or the location, rotation and scale of an object that the custom object changes as it is dragged. One example of this being used is in the Crane object in FlexSim's standard library. When you drag on one of the Crane's sizers, there are actually several objects behind the scenes that the Crane resizes as you drag. Also, the Crane has three variable values defining the x,y, and z location of the Crane's frame that change as you drag the Crane to a different position in the model. In order for undo functionality to work on the Crane, it must tell FlexSim to track undo information on those objects that it is changing as the user drags.

To add undo tracking to an object or number variable, call applicationcommand("addundotracking", view, obj_or_variable). The Crane performs this as part of its OnClick event, when the user clicks on the Crane.

The following are snippets of example code using custom undo.

Simple Undo

```
int undoId = beginaggregatedundo(c, "Set Object Rank"); setrank(anObj, 5); endaggregatedundo(c, undoId);
```

Undo with an Ignore

```
int undoId = beginaggregatedundo(c, "Change Object Shape"); setnodestr(shape(anObj), shapePath); int index =
getshapeindex(gets(shape(anObj))); beginignoreundo(); if (getshapeindex(shapePath)==0) {
    autoloadallmedia(anObj);        index = getshapeindex(gets(shape(anObj)));
} endignoreundo(); set(shapeindex(anObj), index);
applyshapefactors(anObj); endaggregatedundo(c, undoId);
```

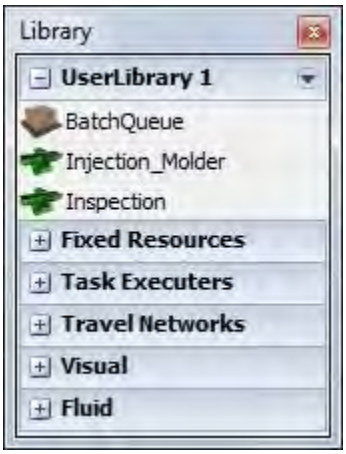
Custom Libraries Concepts

Topics

- Overview
- Adding objects to a custom library
- Editing Library Objects
- Saving / Removing Libraries
- Dropscript and Droppath
- Automatic Install
- Customizing the Library Icon Grid

Overview

FlexSim lets you create and configure special libraries in addition to the standard library set. These are 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. You can create new User Libraries and Open User Libraries from the File Menu.

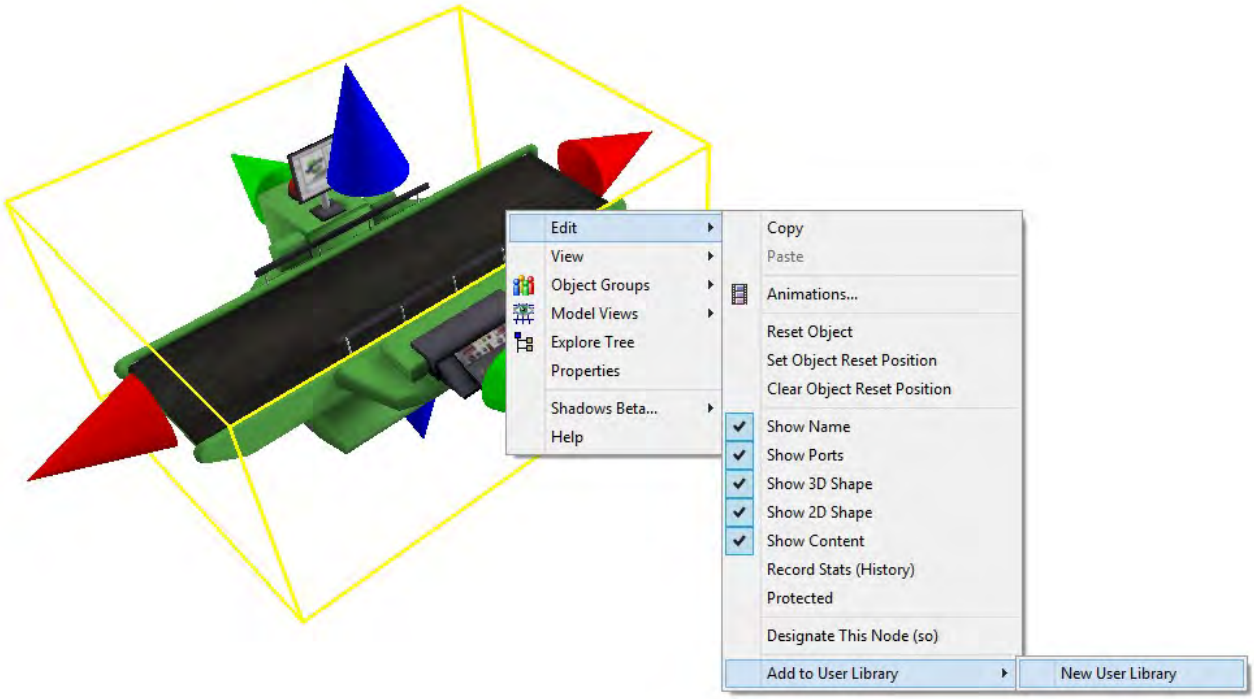


The custom library, or user library, mechanism provides you with flexibility in many areas. The most widely used functionality of the user libraries is to reuse customized objects in a model, but you are not confined to only that functionality. User libraries allow you to automatically install custom 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 dropscript 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.

Adding objects to a custom library

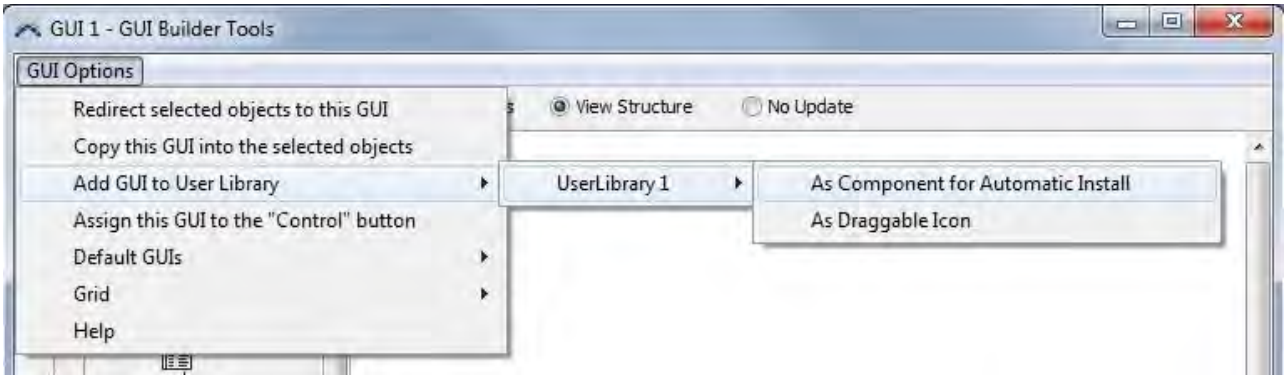
There are several types of objects that you can add to a user library. You will most commonly add a standard object like a Processor or Queue whose properties you have modified to fit a custom modeling

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 submodel, 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 the 3D model view and select Edit > Add to User Library.



If you already have User Libraries loaded, a list of those libraries will appear in the right click menu.

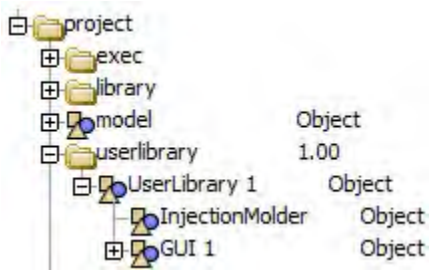
You can also add GUIs, Global Tables, Flowitems, and User Commands to a custom library. You can access this capability from the windows in which you edit these respective objects. The figure below shows a menu option in the GUI editor to add the GUI to a library.



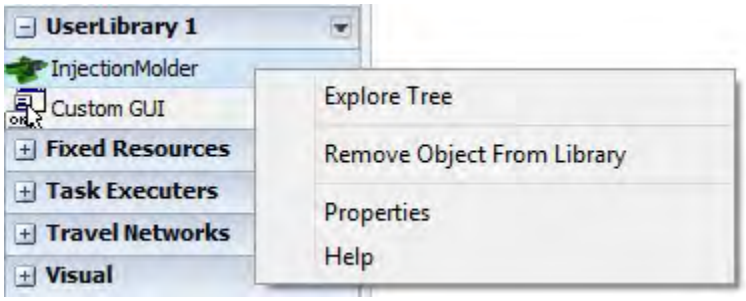
Editing Library Objects

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.

The User Library is stored in the Main Tree in the userlibrary node.

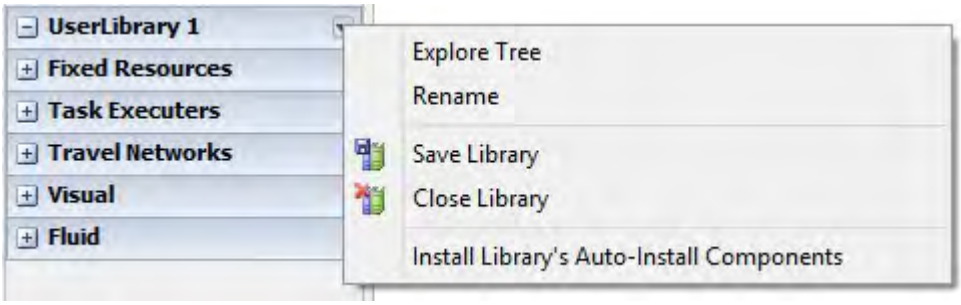


You can also access the properties of your library object through the right click menu on the Library Icon Grid.



Saving / Removing Libraries

To Save or Remove a User Library, click the arrow next to the User Library in the Library Icon Grid.

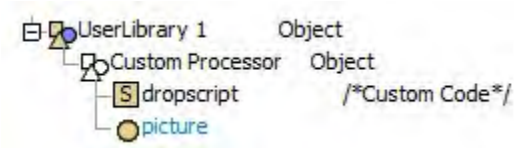


Dropscript and Droppath

The droppath and dropscript 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 "dropscript" or a "droppath" attribute to the object and specifying the data for the dropscript or droppath.

Dropscript

If a dropscript attribute is added, then the dropscript node will be executed as flexscript, and values will be passed into the flexscript function that give it information about the drop.



Note: In order for an object to display in the Library Icon Grid, it needs to have a picture attribute as shown above in our Custom object.

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 a 3D view 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 dropscript 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 command. 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 dropscript function itself. The command returns a reference to the object that was created.

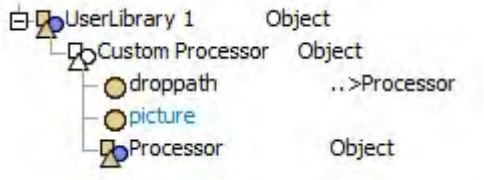
Return Value - Your dropscript function should also return a reference to the object you created. This allows FlexSim to do some extra work like checking to see if there were any c++ nodes created, and if so, set a flag to notify you that you need to compile the next time you run the model.

The dropscript 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 dropscript 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 dropscript 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.

Refer to the Example page for examples on using the dropscript node.

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 of creating a droppath object is the same as for creating a dropscript. 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 flexscript, but should give text data to the droppath attribute and specify in the text a path to another object that should be dropped.



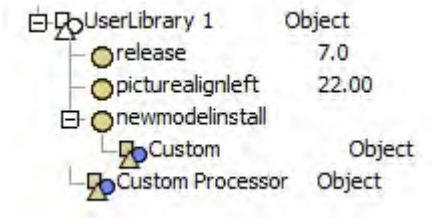
For example, you may give the droppath attribute the text: MAIN:/project/library/NetworkNode. This will effectively cause a network node to be created when you drag your object's icon into the model. You might

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.

The droppath can either be an absolute path *MAIN:/project/library/fixedresources/Processor* or relative *..>Processor*.

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. Automatic install is available through FlexSim's user interface for GUIs, Global Tables, Flowitems, and User Commands.



To create a custom automatic install, you can add special nodes to the user library object. The specific set of valid node names are listed as follows:

newmodelinstall - This node is installed when the user creates a new model. It is also installed when the library is loaded.

startupinstall - This node is installed when the library is loaded if the user has designated this library as a library to load during startup. loadinstall - This node is installed when the library is loaded explicitly by the user.

openmodelinstall - This node is installed when the user opens an existing model. You may use this node to check if the model contains the library's components, and if not, installs them. You could also use it to update components in the model if those components are from an earlier version of the user library.

Customizing the Library Icon Grid

There are a few attributes that can be added to objects in your custom library that will affect they way they are displayed by the Library Icon Grid.

Note: Some visual attributes will not be update in the library icon grid without closing and reopening the view.

Pictures

The picture attribute specifies the path to an image to display in the icon grid. Leaving the path blank will display the object name only. To give your object a Processor image, your picture attribute would contain the path:

bitmaps\processorpicturesmall.png

You can also specify the left alignment by changing the picturealignleft node value.

Icon Grid Width and Height

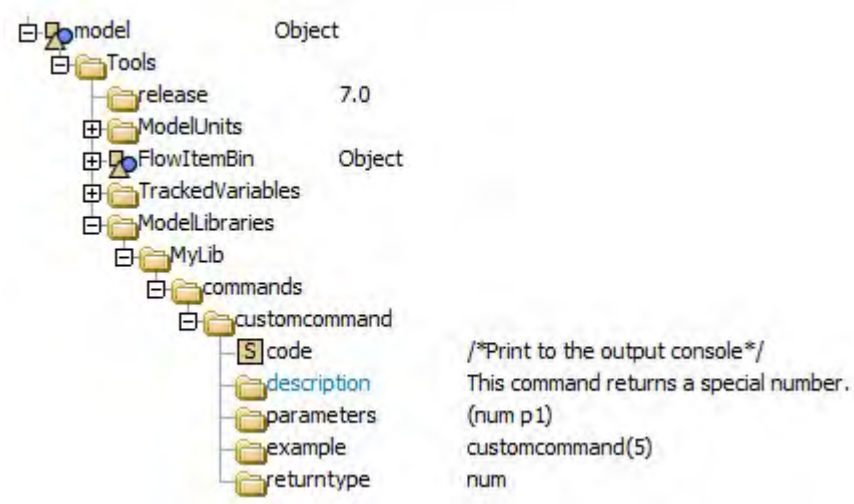
You can designate the size of icons in the library icon grid. Just give the library attributes named "cellwidth" and "cellheight". Each attribute should contain a number that is the width and height of one icon in the icon grid in pixels.

ModelLibraries Node

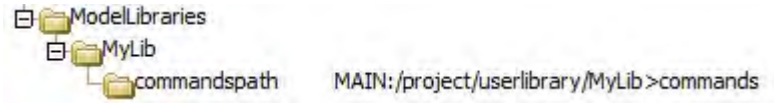
You can add many facets to a model, such as user commands, global variables, global macros, etc. where the definition is "hidden" from the user, so that they don't see it in their User Commands window, Global Variables window, etc. These can also simply refer back to structures in your library, so that you don't have to continually copy stuff into the model when updating. This is done by adding a node named "ModelLibraries" to the model's Tools folder and giving its substructure various data. While you don't need to have a custom library loaded to have this feature be part of a model, it will probably be most used for user libraries, hence we include this information with this topic.

To create this functionality, add a node into model/Tools, and give it the name "ModelLibraries". Inside of that node, add another node, and give it the name of your library, i.e. MyLib. Inside of that node, there are several nodes that can be added. Their name defines their meaning to FlexSim. Names and meanings are listed below:

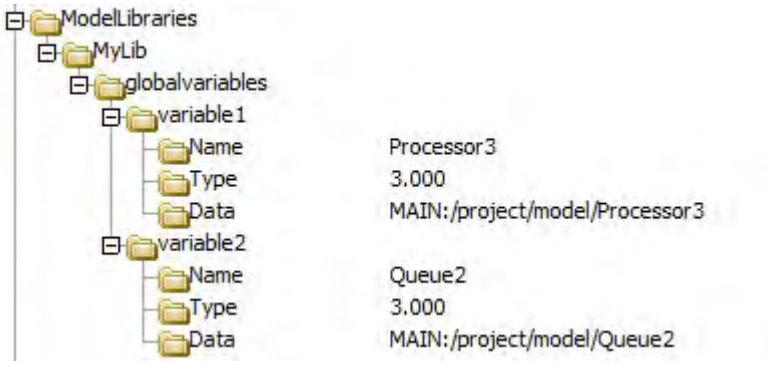
- commands - If you add a node named "commands", you can place commands inside that node. These commands will be accessible to the user to call, and will be documented in the commands documentation, but won't be visible in the User Commands window. The structure of each command should be the exact same as the structure of model/Tools/UserCommands.



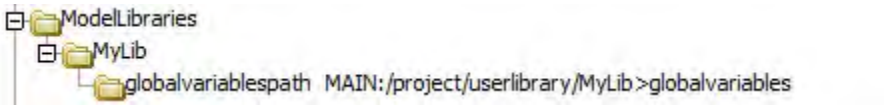
- commandspath - You can add a node named "commandspath" and give it text data that specifies a path to an alternate location that holds the definition of the commands. In this case the structure of the path's destination should be the exact same as the structure of model/Tools/UserCommands. This allows you to leave your command definitions in the user library, and then just have this node refer to it.



- globalvariables - You can add a node named "globalvariables" and give it the same sub-structure as found in Tools/GlobalVariables to define additional global variables that are available to the end user but "hidden" from the "Global Variables" window.

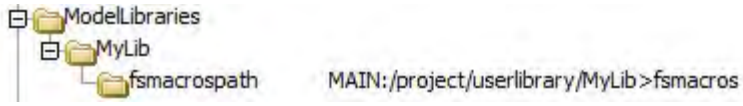


- globalvariablespath - Like commandspath, you can redirect the location of where global variables are defined



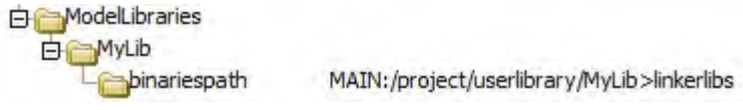
- fsmacros - Here you can add a node named "fsmacros", give it text data with multiple #define macros. These macros will be visible (blue) to the user and will be added as options in auto-completion hints.

- fsmacrospath - Like commandspath, you can redirect the location of where flexscript global macros are defined with the "fsmacrospath" option.

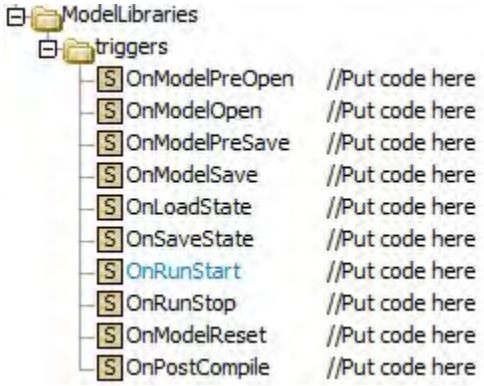


- binaries - You can add a node named "binaries" and give it text data that specifies a list of additional .lib or .obj files that you want to link to when FlexSim is compiled. This is only needed if you have c++ toggled code that must be compiled, or if you expect your end users to define c++ code that can access pre-compiled functionality. This text will be added to the "Additional Library Dependency" field during the C++ linker phase. The linker's library path is specified as \program\system\lib in FlexSim's install directory, so to link with a lib file in FlexSim's libraries directory, you would specify a path like: ..\..\..\libraries\MyLib...

- binariespath - Like commandspath, you can redirect the location of where linker options are specified with the "binariespath" option.



- triggers - You can also add several triggers to fire at certain points. To do this add a node and give it the name "triggers", and add various nodes into that triggers node.



- Valid triggers are:
    - OnModelPreOpen - Fired immediately after a model tree is loaded, and before any model initialization code is fired.
    - OnModelOpen - Fired after a model has been opened and all initialization code has finished. o OnModelPreSave - Fired just before a model is saved to file. o OnModelSave - Fired just after a model has been saved to file.
    - OnLoadState - Fired when a state load is performed. o OnSaveState - Fired when a state save is performed. o OnRunStart - Fired whenever the user presses the "Run" button, as well as whenever the go() command is called.
    - OnRunStop - Fired whenever the user presses the "Stop" button, as well as whenever the stop() command is called. o OnModelReset - Fired whenever the user presses the "Reset" button, as well as whenever resetmodel() is called.
    - OnPostCompile - Fired after the user compiles the model.

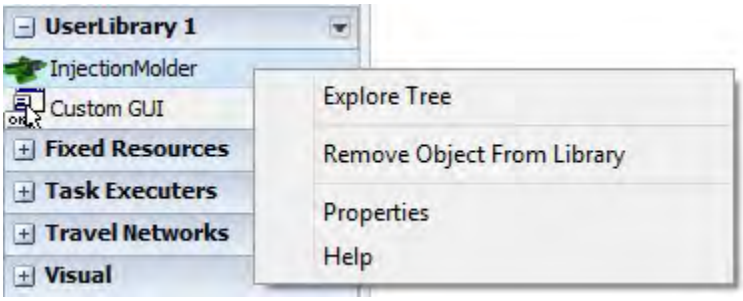
Custom Libraries Example

Topics

- Dropscript Example
- Dropscript Dynamic Parameters Example
- Automatic Install Example

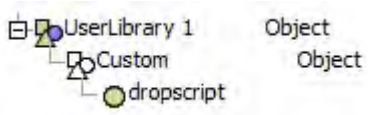
Dropscript Example

Create a new user library by going to File > New User Library. Click on the down arrow next to the newly created User Library in the Librar Icon Grid and select Explore Tree.



You should now see an empty tree.

Insert a new object into the library by right clicking on it and selecting Node > 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 by pushing the plus button. Give it the name "Custom". Now add object data to the node by right-clicking on it and selecting Node > Add Object Data or by left-clicking on the node and hitting the O key. 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 appears as follows.

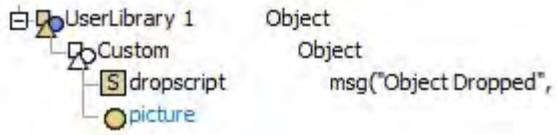


Add text to the dropscript attribute by right-clicking on it and selecting Node > Add Text Data or by leftclicking the node and pressing the T key. Toggle the node as flexscript by right-clicking on it and selecting Build>Toggle as FlexScript. Give the attribute the following flexscript text:

```
msg("Object Dropped",   concat(
   "Onto Object: ",     getname(parnode(1)), "\n",
   "X: ",     numtostring(parval(2),2,2), "\n",
   "Y: ",     numtostring(parval(3),2,2), "\n",
   "Z: ",     numtostring(parval(4),2,2), "\n",       "Onto
View: ", getname(parnode(5))
 )
);
```

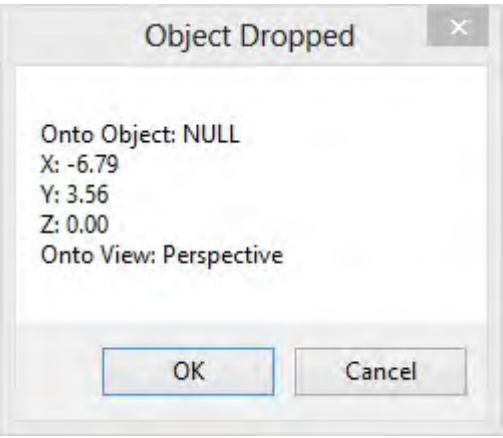
Click off of the attribute node, then right-click it and select Build>Build Node FlexScript.

Finally, add a picture attribute to the attribute tree of the object by right-clicking on the drop-script node and selecting Node > Insert After. 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 and will just leave the picture attribute blank, which causes only the object text to be shown in the grid. The tree should appear as follows.

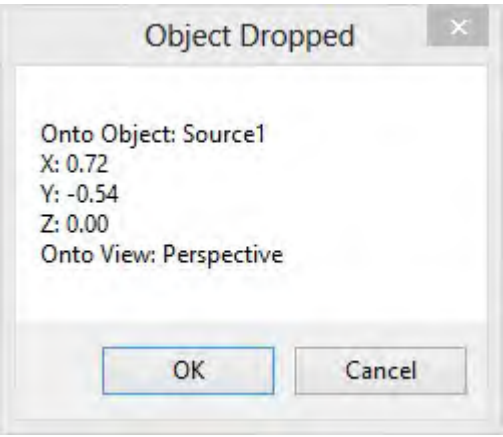


The library icon grid should appear as follows.

Now drag the Custom icon from the Libary 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.



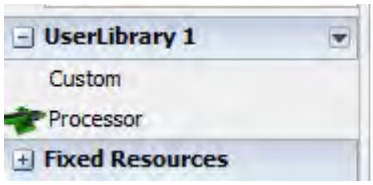
This time the message shows that it was dropped onto Source1. The x, y and z locations are now relative to the source object.

Dropscript Dynamic Parameters Example

In this example, we will have an object that is created through our User Library that will have it's size defined by a Global Table.

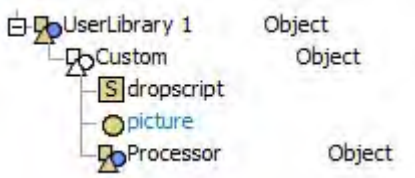
Create a new User Library as described in the above Dropscript Example.

Drag-and-drop a Processor object from the Library Icon Grid to the 3D view. Right click the Processor and select Edit > Add to User Library > UserLibrary 1 to add it to our library.

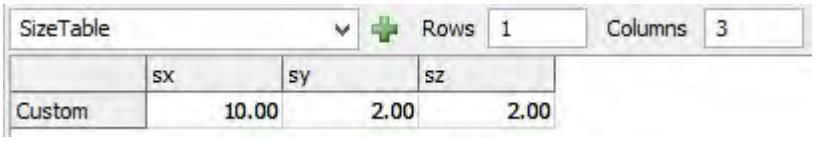


We want to dynamically change the size of our Processor as it is dragged into the 3D view. In order to accomplish this, we will use our Custom object with code in our dropscript node. We don't want our Processor to show up in the Library Icon Grid, so we'll copy it into our Custom object as an attribute node. Copy the Processor object (left click and hit Ctrl+C). Left click on the Custom object and open the object's attribute tree if it is not currently open by pressing the >. Hit the enter key to create a new attribute node in

the Custom object. Left click the new node and hit Ctrl+P to paste the Processor object. You can then delete the original Processor object.



Create a new GlobalTable through Tools > Global Tables > Add. Name the table SizeTable. Make your GlobalTable look as follows:



The values from this table will define the size of our processor.

Back in the Custom object, update the dropscript code to the following:

```
treenode ontoObj = parnode(1); double x
= parval(2); double y = parval(3); double z
= parval(4); treenode ontoView =
parnode(5);
treenode newProcessor = dropuserlibraryobject(node("..>Processor", c), ontoObj, x, y, z, ontoView);
setnodenum(spatialsx(newProcessor), gettablenum("SizeTable", 1, 1)); setnodenum(spatialsy(newProcessor),
gettablenum("SizeTable", 1, 2)); setnodenum(spatialsz(newProcessor), gettablenum("SizeTable", 1, 3)); return
newProcessor;
```

Right click the dropscript node and select Build > Build Node FlexScript.

Now create a Custom object by dragging and dropping it from the Library Icon Grid to a 3D view. Notice the size of the object has been altered by our dropscript.

Note: This is the proper way to dynamically change parameters or variables of a custom library object when it is dropped into a view (though it does not matter where the created object is contained). If you add a dropscript node straight to the Processor, the Processor will not be created. If you try and create the object in the Processor's dropscript node, you will cause FlexSim to get into an infinite loop and crash.

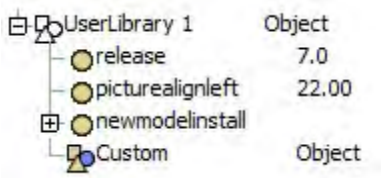
Automatic Install Example

For this example, we will have the Processor object from the Dropscript Dynamic Parameters Example be automatically installed when the user creates a new model.

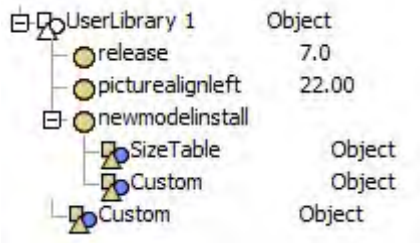
Open the GlobalTable SizeTable. Click on the button and select Add to UserLibrary 1 > As AutoInstall Component.

Open UserLibrary 1's Tree view by click the arrow next to the User Library in the Library Icon Grid and select Explore Tree.

Expand the library's attributes tree by pressing the button next to UserLibrary 1. The tree should appear as follows.



Press the plus button next to the newmodelinstall to expand the node. Notice our SizeTable already appears inside the node. This is important as our Custom object's dropscript refers to this GlobalTable. Create a subnode of newmodelinstall by left-clicking the node and hitting the Enter key. Copy and paste the Custom object onto this new node.



Now you can hit the "New" button on the toolbar of File > New Model. You'll notice that there is a Processor1 object in the 3D view and our SizeTable in the GlobalTables. The newmodelinstall folder acts a set of objects that will be "dropped" into the model when a new model is created. The object's are "dropped" in that the same functionality is executed as when you actually drag an object from the icon grid. In the case of a dropscript object, the script is executed as if the object were dropped at the point (0,0,0) in the model.

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 (FlexScript): 

Function (FlexScript, not built): 

Function (C++): 

DLL Linked Function: 

Global C++ Function: 

Simple Data: 

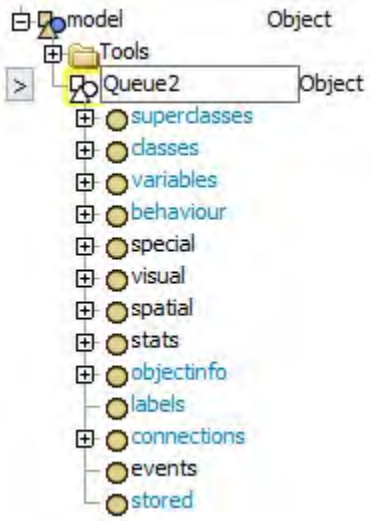
Nodes can be added and deleted from the tree. To delete a node, simply click the node and then hit the delete key. To insert a node, right-click on an existing node and choose Edit > Insert. This will add a new node immediately after the node that was clicked on. The shortcut for this operation is to 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 Main Tree and the View Tree.

The main tree contains the Executive data, the Library, and the Model.

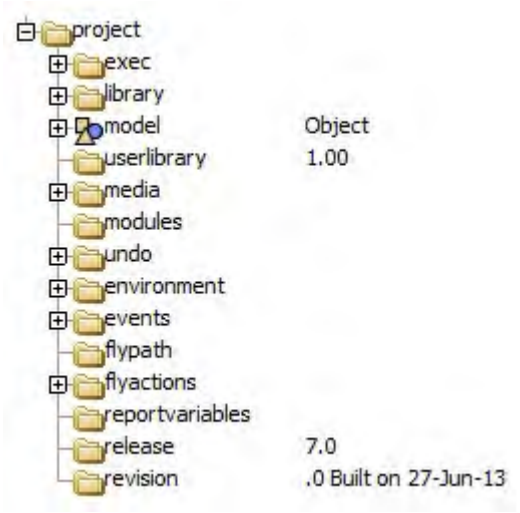
The view tree contains information on windows, editors, and other user interfaces. It also manages active windows.

Main Tree

To view the main tree, click on the 

in the FlexSim toolbar, or select the main menu option: View > Model Tree. A new tree window will appear. In the Quick Properties window you'll see a Tree Navigation section. Click on the Main.

The main tree holds many of the higher level functions in FlexSim. It also includes 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.

userlibrary

All loaded custom user libraries.

Media

Stores Images, 3D models and Sounds.

Undo

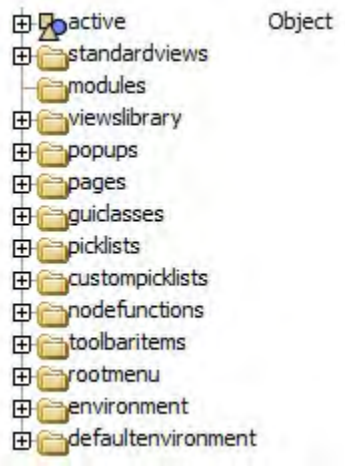
Holds undo history. A numerical value for this node is the limit on the number of undo steps. If there is no numerical data, undo will be disabled. Undo functionality may also be globally disabled.

View Tree

To view the view tree, click on the 

in the FlexSim toolbar, or select the main menu option: View > Model Tree. A new tree window will appear. In the Quick Properties window you'll see a Tree Navigation section. Click on the View.

The view tree contains data for creating, storing, and using graphical user interfaces for objects.



Active

This stores all of the currently open windows for the interface.

Standardviews

All of the non-property windows are stored here.

Popups

Popups are used throughout FlexSim, mostly for trigger and picklist options as well as utility purposes.

Pages

Stores all of the Object Property Windows.

Picklists

Preset code for picklist options.

Keyboard Shortcuts

Spacebar - Insert a new node after.
Enter - Insert a new node into.
N - Add number data to the highlighted node.
T - Add string (text) data to the highlighted node.
O - Add object data to the highlighted node.
P - Add pointer data to the highlighted node.
Shift + Delete - Delete all data from the highlighted node. Backspace and
Delete - Delete the node.

FlexSim XML

FlexSim saves it's models, library and tree files in XML format. There are many advantages to using this capability in your model development, including:

- Since XML is an ascii/text based format, it increases the utility of using content management and versioning software such as Subversion, CVS, Microsoft Visual SourceSafe, git, etc. to manage the development of a model. These systems automatically track a history of changes to a model or project, and for ascii/text based files, they allow you to see line-by-line change logs for the files, as well as revert line-item changes if needed.
- With the added benefit of versioning systems comes the ability to develop a model concurrently by different modellers using a much more stream-lined method. No more saving off individual t files from one modeller's changes to a model and loading them manually into the master model. When saving to XML format, if one modeller is working on one portion of the model, only that portion of the model file changes when saved, so when the modeller checks his changes into the versioning system's repository, another modeller can automatically merge those changes into his copy. If there are conflicts, where two modellers change the same part of the model, then the versioning system has tools that allow modellers to easily see in the XML file where the conflicts occur and quickly perform the manual conflict resolution.
- FlexSim also has the added ability to distribute a single model across multiple XML files. This increases your control over how to manage the model development in your versioning system, and makes conflict resolution even easier.
- The distributed save mechanism also opens the door for much more automated control of FlexSim. By saving small pieces of your model off into separate XML files, you can auto-regenerate one or more of those XML files outside of FlexSim, consequently changing the configuration of the model for the purpose of running different scenarios.

Tutorial

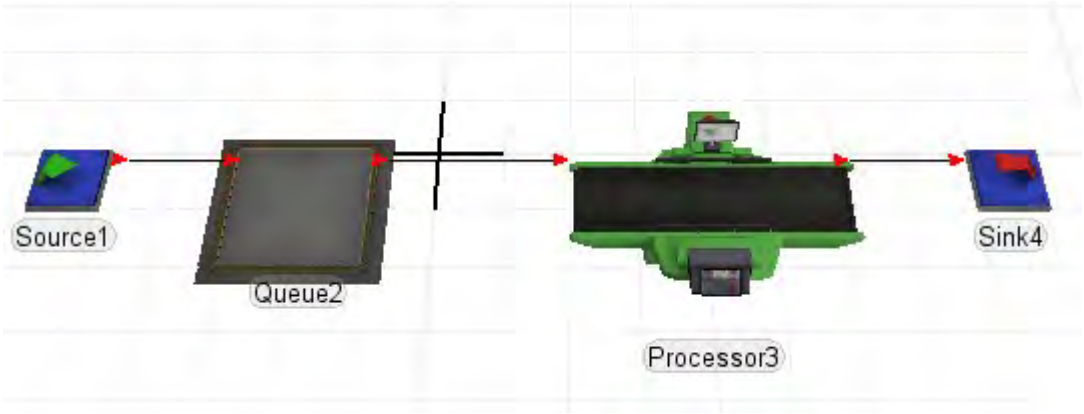
This tutorial will take you through saving a model as XML, and how you can configure the distributed save.

Topics

- Build a Simple Model
- Save in XML Format
- Version Management Utilities • Distributed Save
- Why Distribute?

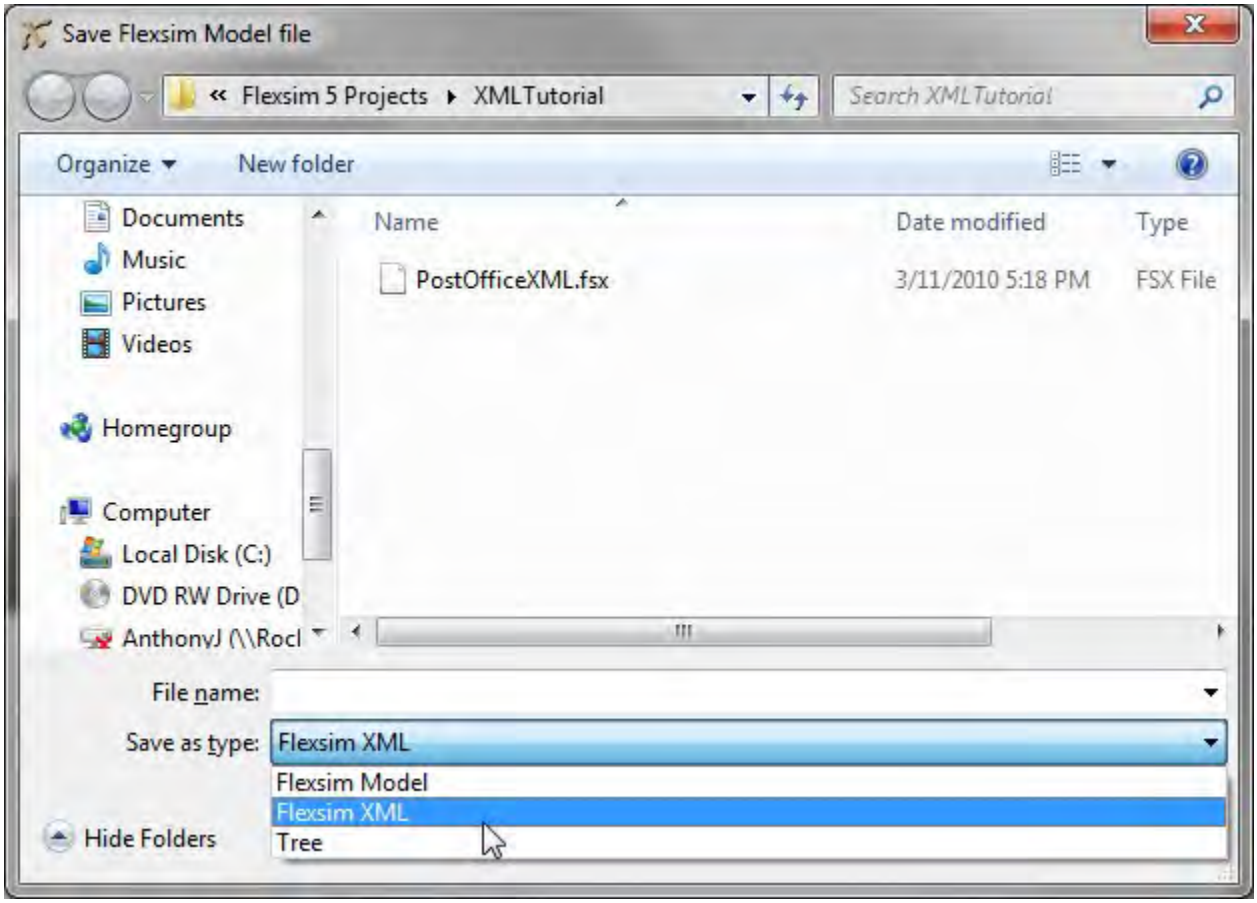
Build a Simple Model

First build a simple example model containing a Source, Queue, Processor and Sink.

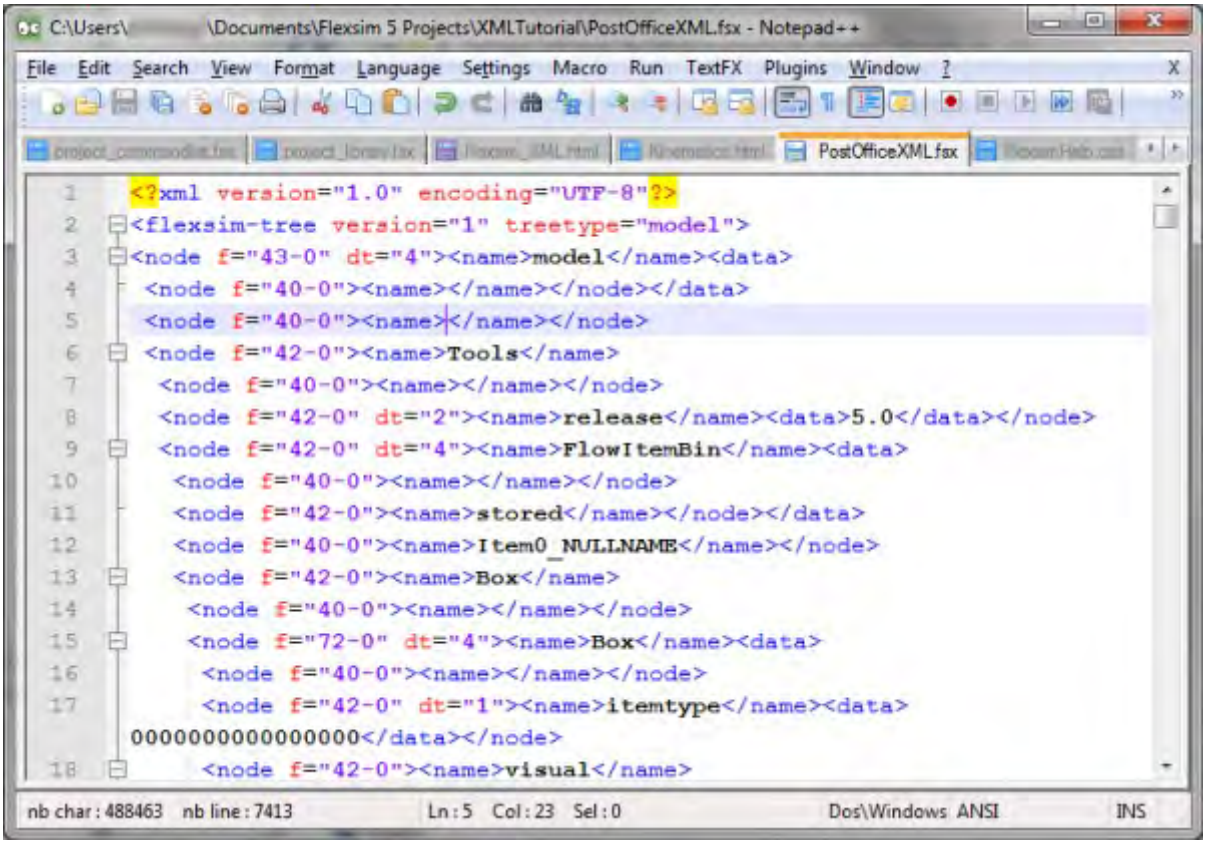


Save in XML Format

Save the model, and in the save dialog, choose "FlexSim XML" from the drop-down at the bottom. Save the model as "PostOfficeXML.fsx".



Now you've saved the file in FlexSim's XML format. You can view the file in a regular text editor like Notepad, Visual Studio, Notepad++, EditPlus, etc.

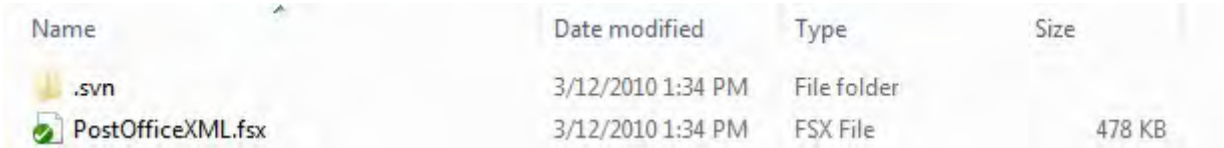


Unless you plan on doing automated changes to the XML file, it's not necessary to know all the details of the file format. In simple terms, the primary tag in FlexSim XML is the <node> tag, representing a node in FlexSim. The node's name is described in the <name> tag, and the node's data, if applicable, is described in the <data> tag. If you do plan on doing automated changes, and need a more detailed definition of the xml format, you can refer to FlexSim's xml schema (see the FlexSim install directory/help/MiscellaneousConcepts/FlexSimXML.xsd for more information).

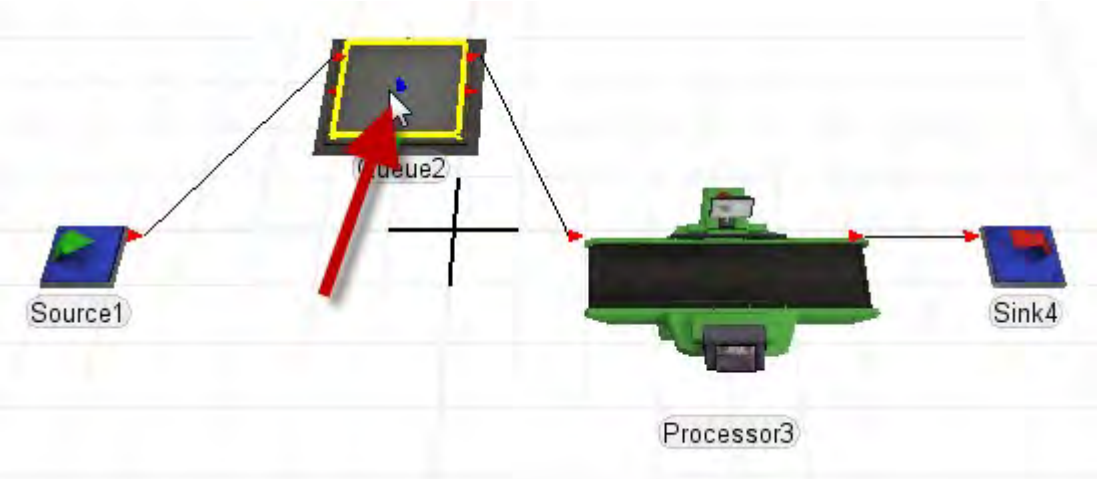
Version Management Utilities

To give you a better idea of the advantages of using an XML format, we'll go through some of the things that you might do with a versioning system. At FlexSim we use Mercurial, with TortoiseHg as our clientside tool, and have found this to meet all of our development needs quite sufficiently. Details of installing and configuring a version management system are outside the scope of this tutorial. You can find many install and setup helps online. As part of this tutorial we will simply assume that, once the model has been saved, a Subversion repository is set up for the model.

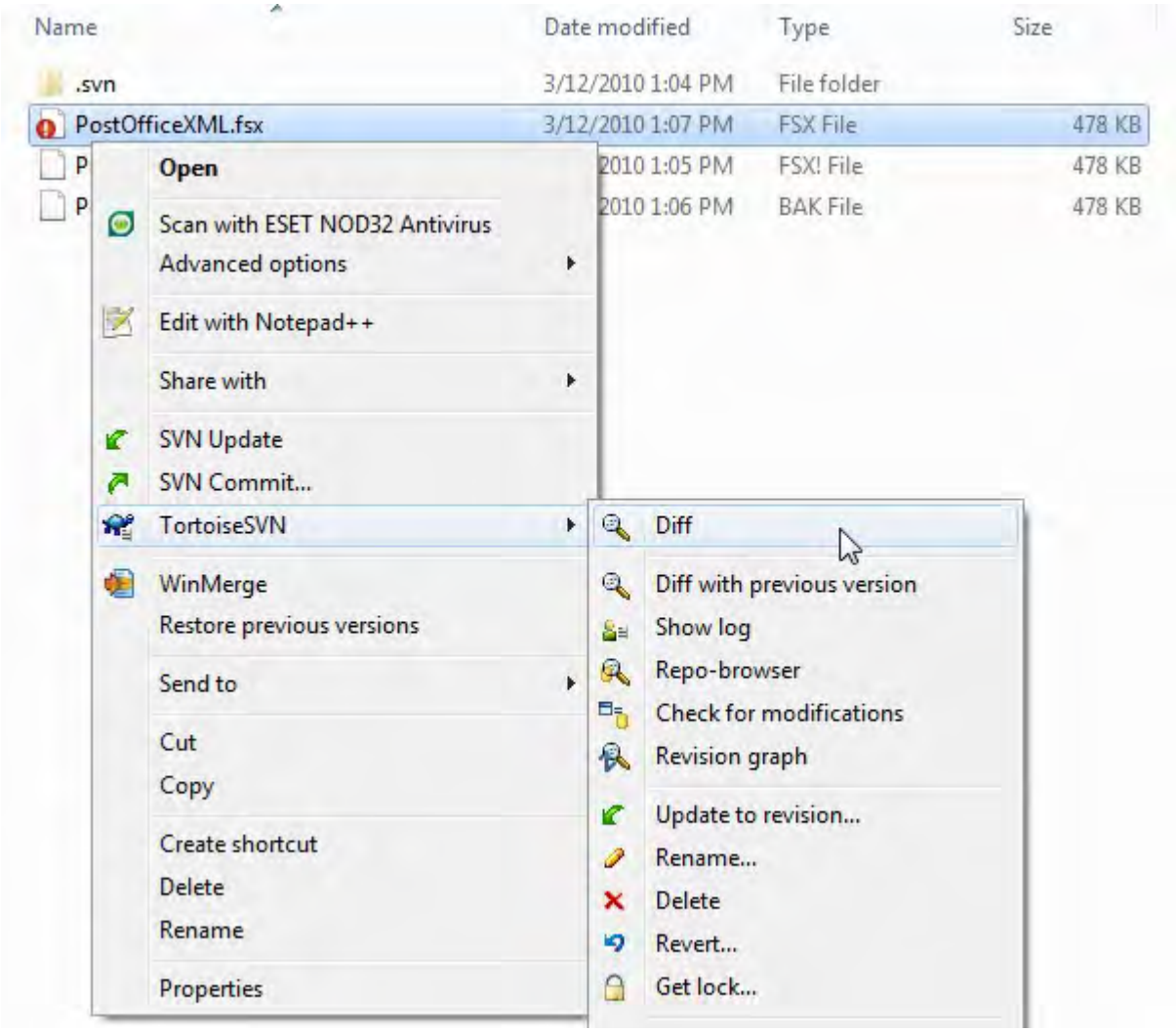
When the Subversion repository is initially created, the file will appear with a green icon indicating that your copy is "clean", or you haven't made any changes since your last check-in.



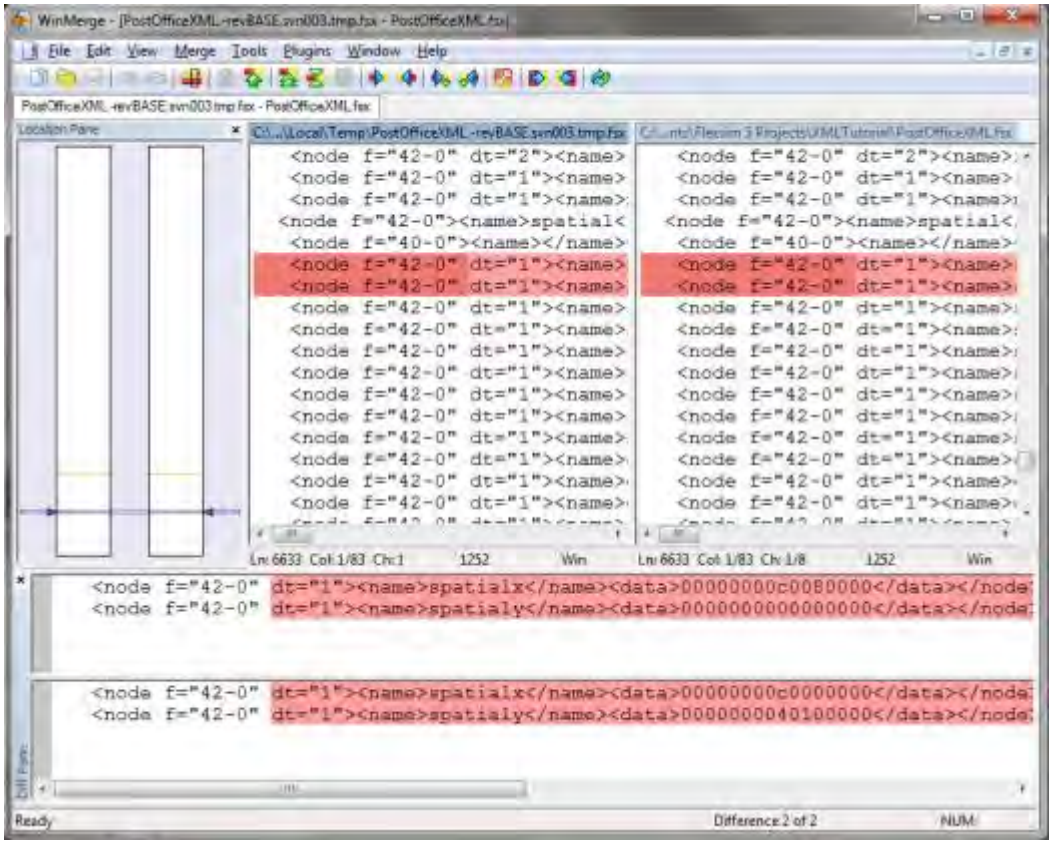
Now let's say we make a small change to the model, such as moving the Queue to a different location and then saving the model again.



Once saved, you'll notice that the file's icon has changed to a red symbol, meaning the file is "dirty", or it has changed since the last check-in. You can also right-click the file, and see exactly what's changed by choosing the "Diff" operation. This will give you a difference comparison of your current copy of the file with the last version that you checked-in.



TortoiseHg provides a simple diff utility, but you can also configure it to use a third-party comparison tool such as WinMerge.



Using the diff tool you can see where the model has been changed, namely in spatialx and spatialy attributes. You can revert line-item changes in the diff tool if you don't want those changes applied.

Distributed Save

Next let's distribute the model across multiple files. To do this you create a "FlexSim File Map" file. This is an xml file with the .ffm extension. It must be placed in the same directory as the model's .fsx file, and, except for the extension, must be given the same name as the .fsx file. So, let's create that file. Let's also create a subdirectory to put the distributed files into.

Name	Date modified	Type	Size
.svn	3/12/2010 1:34 PM	File folder	
distributedfiles	3/12/2010 2:10 PM	File folder	
PostOfficeXML.ffm	3/12/2010 2:10 PM	FFM File	0 KB
PostOfficeXML.fsx	3/12/2010 1:34 PM	FSX File	478 KB

Now edit PostOfficeXML.ffm in a text editor. The first thing we'll do is put the node model/Tools/active into a different file. This is something you'll probably want to do always if you're using version management. The node model/Tools/active stores all of the windows open in the model, so if you're editing a model and change or reposition the windows that are open, that will change the model file. For version management this is often just static that doesn't need to be tracked, so we'll have the node saved off to a different file, and then we can just never (or at least rarely) check that file into the version management system. Specify the following code:

```
<?xml version="1.0" encoding="UTF-8"?>
<flexsim-file-map version="1">
    <map-node path="/Tools/active" file="distributedfiles\windows.fsx" file-mapmethod="single-node"/>
</flexsim-file-map>
```

Save the file map, then go back into FlexSim. Save the post office model again under the same name "PostOfficeXML.fsx". Now look in the distributedfiles directory. You'll see that it contains a new xml file named windows.fsx. This xml holds the definition of the node model/Tools/active. All interaction with the model remains the same, i.e. from FlexSim you just load and save the main xml model file, but now the model is distributed across multiple files.

In the file map, the main document element is the <flexsim-file-map> tag. Inside the main document element should be any number of <map-node> elements. Each <map-node> element should have a path attribute, a file attribute, and a file-map-method attribute. The path attribute should specify the path, from the main saved node, to the node that is going to be saved into the distributed file. The file attribute specifies the path, relative to the saved model file, to the distributed file that you want to save. The filemap-method should either have a value of "single-node" or "split-points". In this case "single-node" means you want to save all of the active node into windows.fsx.

Now let's do an example of the "split-points" method. Let's say we want to save the Tools folder in one file, the Source and Queue into another file, and the Processor and Sink in a third file. To do this, add another <map-node> element to the file map:

```
<?xml version="1.0" encoding="UTF-8"?>
<flexsim-file-map version="1">
    <map-node path="/Tools/active" file="distributedfiles\windows.fsx" file-mapmethod="single-node"/>
 <map-node path="" file="distributedfiles\Tools.fsx" file-map-method="splitpoints">
                        <split-point name="Source1" file="distributedfiles\Source_Queue.fsx"/>
                <split-point name="Processor3"
file="distributedfiles\Processor_Sink.fsx"/>
        </map-node>
</flexsim-file-map>
```

Now save the file, save your FlexSim model, and again you'll see new files created in the distributedfiles directory.

If a <map-node> element uses the "split-points" method, then it can have <split-point> sub-elements. Each split-point should have a name, defining the name of a node inside the map-node's defined node, and a file attribute defining the file to save to. The map-node has a path="" attribute. This means that it applies to the root node, or the model itself. The map-node's file attribute defines the first file to use. This configuration tells FlexSim to save all sub-nodes in the model up to but not including the node named "Source1" to the file "distributedfiles\Tools.fsx", save everything from Source1 up to but not including the node named "Processor3" in "distributedfiles\Source_Queue.fsx", and save everything from Processor3 to the end in "distributedfiles\Processor_Sink.fsx".

Why Distribute?

The main reason you would want to distribute your model across multiple files is for version management. It allows you to easily see what you've changed in your model by seeing what files, and consequently

which parts of the tree, have changed. If you inadvertently changed one piece of the model, you can easily see that and then revert that change if needed. When multiple modellers are developing the model, one modeller can essentially confine himself to one part of the model tree, and by associating that part of the tree with a specific file, it makes it much easier to merge changes from different developers, it reduces the chance of conflicts in the merge, and makes it easier to do a manual merge if needed.

Note on connections: FlexSim's XML save mechanism does have one catch regarding input/output/center port connections, as well as any other mechanism that uses FlexSim's coupling data. If you change connections in one portion of your model, it will actually change the serialized values of all connections/coupling data that occur after those connections in the tree, even if those connections were not changed. This can very easily cause merge issues if multiple modellers are changing connection data. However, if you distribute the model across multiple files, connection changes where both connection endpoints are within the same file will only affect that file, and will not change other files. So if you can localize large "connection sets" into individual files, you can minimize the effect of changes and subsequently minimize merge conflicts.

GUI Events and View Attributes

GUI Events

OnClick
OnClose
OnDrag
OnDraw
OnDrop
OnKeyDown
OnKeyUp
OnKillFocus
OnMenuPopup
OnMouseButtonDown
OnMouseButtonUp
OnMouseWheel
OnMouseWheelDelta
OnOpen
OnPreDraw
OnPreOpen
OnPress
OnSelect
OnSize
OnStick

The following are not recommended, but may still work

OnActivateNotify
OnDropFile
OnDropNode
OnListen
OnMessage
OnPreListen
OnTimerEvent
OnUndo

View Attributes

alignrightposition
alignrightmargin
alignbottomposition
alignbottommargin
aligncenterx
aligncentery apply
beveltype bitmap
cellheight cellwidth
close
coldlink
coldlinkname
coldlinknamex
coldlinkx
connectorsize
connectorstyle
graphannotate
graphaxes
graphbars
graphgrid
graphhistodata
graphlegend
graphlegendhisto
graphlines
graphmaxpoints
graphpie
graphpoints
graphpiedata
graphstep
graphtitle
graphxy grayed
gridfog
gridlinecolor
gridlinewidth
gridx gridy gridz
hidden
hotlink
hotlinkname
initialtext
itemcurrent
items
labelscale title viewlighty menucustom tooltip viewlightz
menumain viewautoconnect viewmagnification menupopup
viewbackgroundcolor viewnear menuview
viewconnectioncolor viewpointradius
noformat viewfar viewpointrx objectfocus viewfield viewpointry
pagelist viewfirstperson viewpointrz palettewindow viewfocus
viewpointx pickcopydataonly viewfog viewpointy pickitem
viewfont viewpointz picklist viewfull viewprojectiontype
picklistnameonly viewhideallbases viewshowgrid pickprimary
viewhideallconnectors viewshowheads picture
viewhidealldrawcontent viewsnaptogrid rangeexp
viewhidealllabels viewsyncupdate rangemin viewhiderouting
viewwindowclean rangemax viewlightaspos viewwindowopen
spatialsx viewlightb viewwindowsource spatialsy viewlightg
viewwindowtype spatialx viewlightr windowtitle spatialy
viewlights wordwrap
statusbar viewlightx

GUI Events

OnClick

This event 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 for this event:

- "c" is a reference to the control node.
- "i" is a click code: 2 means left mouse button down, 3 means left up, 4 means right down, 5 means right up, and 1 means double-click.

Some commands that you might use within the OnClick are: cursorinfo() or selectedobject().

OnClose

This event contains text data with flexscript code that will be executed when the window is closed.

OnDrag

This event allows you to execute code when an object is dragged from the icongrid onto another view. The event 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, perpective, 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 event exists, then a copy of the object will be made in the dropped view's focus or in the dropnodeto() if it exists.

OnDraw

This event is fired when the window is drawn. This occurs frequently during a simulation, such as when the window is resized, the model is running, or you click on the window.

OnDrop

This event 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().

OnKeyDown

This event 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 event 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().

OnKillFocus

This event is fired when the window that the event is applied to loses keyboard focus meaning that you from being able to type in the window to not being able to type in the window because you have clicked on a different window.

OnMenuPopup

This event is executed after the menu is created but before it draws the menu so that you can check or gray out menu items using the commands menucheck() or menugray().

OnMouseButtonDown

This event 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 event 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.

OnMouseWheel

This event 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 event 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 accesses the OnMouseWheel event itself.

OnMouseWheelDelta

This event is used as described in the OnMouseWheel event above.

OnOpen

This event 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.

OnPreDraw

Executed before an OnDraw event. See OnDraw for more information.

OnPreOpen

This event 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 initialized. 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.

OnPress

The OnPress event 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.

OnSelect

The OnSelect event is a trigger that fires when the user selects an option in the combobox, the user drags the locator of a tracker to a given position, or when a tab page is selected for a tabcontrol. The event 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. For the Tracker, within the function, c will access the tracker control while for the Tabcontrol c accesses the tabcontrol view. For a tabcontrol, use get(itemcurrent(c)) to get the currently selected page.

OnSize

This event is executed when the window gets re-sized.

OnStick

This event is executed on joystick or 3D mouse events (moving the joystick or pressing buttons on the joystick or 3D mouse).

OnActivateNotify

Executed when you change the active window. This event may only work on palette windows. An example of this event can be found in the Tree Navigation window. When a 3D view is selected with the Tree Navigation window open, options in the Tree Navigation window are grayed out. However, when a tree window is selected, those options become active (un-grayed) again.

OnDropfile

Executed when a file is dragged on top of the window and then dropped onto it.

OnDropNode

This event 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.

OnListen

Executed on any event happening: after dispatch, when event has been removed. To set up listening, you add a "listeners" attribute to the object you want to listen to (Object A). This listeners attribute should have subnodes that are couplings to the object(s) that you want to listen from (Object B). Before an event fires on Object A, Object B's OnPreListen event function will fire. After an event first on Object A, Object B's OnListen event function will fire. You can also filter events by adding a subnode with number data below Object B's coupling node. The number data should be a bitwise sum of the bitshifted event codes you want to listen for. If the coupling has no subnode, the object will default to listening to every event. To help you with the OnPreListen and Onlisten event functions, you may consider using the listenerinfo(int info) command. The information returned is information that was passed to the event you are listening to. Info 1 will return the engine event code. Engine event codes have macros such as SM_MESSAGE and SM_DRAW. Info 2 and 3 return pointers to associated

OnMessage

Executed when the GUI object receives a message that was sent from another object in the model.

OnPreListen

Executed on any event happening: before dispatch, when event is still in list. Refer to OnListen for a more detailed description.

OnTimerEvent

Executed on timer event. Executed on an event that was created with createevent(). This occurs in numerous objects, one example is the Processor finishing its process.

OnUndo

Executed on content-defined undo action. When you call the createundorecord(.,.,UNDO_CUSTOM); command you create a document that records the most recent changes that were made to the model.

Then when you undo something (Ctrl z) you can use the OnUndo event to reference that information and undo what was done or perform some other function.

View Attributes

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.

aligncenterx

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.

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.

apply

If the apply attribute is added to a button, then FlexSim will call applylinks() on the button's owner view when the button is pressed. The attribute needs no data.

The beveltype attribute specifies what the border of a 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

Using a bitmap attribute, a bitmap image can be applied to a given view/control.

Panel: Simply add the bitmap attribute to the panel and give it the path to the bitmap file (like button\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.

Static: 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 a "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.

Checkbox: The bitmap attribute causes the checkbox to have a bitmap next to it instead of text. It should be created the same as the static view type was created.

Radiobutton: The bitmap attribute causes the radiobutton to have a bitmap next to it instead of text. This one is also created in the same way as the static view type was created.

Button: The bitmap 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.

cellheight

This attribute establishes the height of a given cell. It can be used in a couple of different views.

Table: This attribute specifies the height, in pixels, of each row in a table. Unlike table columns however, all table rows must be the same.

Icongrid: 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.

cellwidth

This attribute establishes the width of a given cell. It can be used in a couple of different views. *Table:* This attribute specifies the default column width, in pixels. The attribute should have number data with the value in pixels. The 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 while the second subnode defines the first column's width, and so forth.

Icongrid: The attribute allows you to define the width, in pixels, of each rectangle in the icongrid. It should have number data containing the desired width.

Tree: The attribute lets you specify how wide, in pixels, each node's name will be shown.

close

For a Button, if the close attribute is added, then FlexSim will close the button's owner view when the button is pressed.

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 (or another value of an attribute or variable node) 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. 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 paramter 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. Below is a list of the different areas of GUI building that use coldlinks and the attributes purpose in each case. For more information on traversing the tree, see Traversing the Tree.

Edit View: This attribute defines what will appear in the text of the control. If the linked code 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. If you were to use a coldlink for a static you would find the attribute allows you to have the text be dynamic based and an attribute of the object.

Static View: This attribute simply defines 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 the setviewtext() and getviewtext() to explicitly get and set the text of the control. This can similarly be done for an Edit control as well.

Checkbox: This attribute links the state of the checkbox to a value in an attribute of the object. The node that it links 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).

Radiobutton: The attribute links the state of the radibutton to a value in an attribute of the object. The node that you 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). Radio buttons present a problem because in order to have them work using just a coldlink, 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 checked, 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.

Button: The coldlink works as stated at the beginning and simply designates what will be shown as the button's text.

Script: The attribute simply links the text of the script control to the text of a node on the object and works like documented earlier.

Tracker: The attribute simply links the tracker or statusbar to a value on a node and it works like documented earlier.

Statusbar: The attribute simply links the tracker or statusbar to a value on a node and it works like documented earlier.

coldlinkname

This attribute is just like the coldlink, except it links with the name of the node specified by the coldlink's path. For more information on traversing the tree, see Traversing the Tree.

coldlinknamex

This attribute is just like the coldlinkname, except that instead of holding text data with a path to the involved node, the coldlinknamex holds flexscript code that returns the node whose name should be linked to. For more information on traversing the tree, see Traversing the Tree.

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. For more information on traversing the tree, see Traversing the Tree.

connectorsize

This attribute determines the size of the arrows drawn at the ends of connections between connected objects in a 3D view.

graphaxes

In a Graph, 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.

graphbars

This attribute has many diverse uses and purposes in GUI building. Below is a list of the different areas of GUI building that use graphbars and the attributes purpose in each case.

Histogram: This attribute signifies that the graph will be shown as bars. The attribute needs no data.

Line Chart: 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.

Bar Chart: 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.

Scatter Plot: 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.

graphgrid

In a Graph, 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.

graphhistodata

In a Histogram, 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.

graphlines

This attribute has many diverse uses and purposes in GUI building. Below is a list of the different areas of GUI building that use graphlines and the attributes purpose in each case.

Line Chart: Add this attribute to have a line drawn between consecutive points in the graph. The attribute needs no data.

Bar Chart: Add this attribute to have a line drawn between consecutive points in the graph. The attribute needs no data.

Scatter Plot: Add this attribute to have a line drawn between consecutive points in the graph. The attribute needs no data.

graphpie

In a Pie Chart, this attribute signifies that the graph will be shown as a pie chart. The attribute needs no data.

graphpoints

This attribute has many diverse uses and purposes in GUI building. Below is a list of the different areas of GUI building that use graphpoints and the attributes purpose in each case.

Line Chart: Add this attribute to have a dot drawn at each point in the graph. The attribute needs no data.

Bar Chart: Add this attribute to have a dot drawn at each point in the graph. The attribute needs no data.

Scatter Plot: Add this attribute to have a dot drawn at each point in the graph. The attribute needs no data.

graphpiedata

In a Pie Chart, this attribute should have text data the contains a path to a node with sub-nodes. Each subnode's name will be show 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 the each sub-node's value represents a chunk of the pie.

graphstep

This attribute has many diverse uses and purposes in GUI building. Below is a list of the different areas of GUI building that use graphstep and the attributes purpose in each case.

Line Chart: 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.

Bar Chart: 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.

Scatter Plot: 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.

graphtitle

In a Graph, 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.

graphxy

This attribute has many diverse uses and purposes in GUI building. Below is a list of the different areas of GUI building that use graphxy and the attributes purpose in each case.

Line Chart: 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.

Bar Chart: 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.

Scatter Plot: 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 xvalue 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.

grayed

This attribute causes the control to be disabled, graying the control and disallowing the user from manipulating the control. It should container 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.

gridfog

This attribute determines how much fog will cover the grid on a scale from 0 to 1. Typically small numbers work better than larger ones.

gridlinecolor

This attribute determines the color of the gridlines. The 3 subnodes of this attribute determine the red, green, and blue color components used to draw the grid in a 3D view.

gridlinewidth

This attribute determines the width of the grid lines drawn in a 3D view.

gridx

If the graphgrid attribute is present, then you can also add a gridx attribute, which specifies the x grid interval.

gridy

If the graphgrid attribute is present, then you can also add a gridy attribute, which specifies the y grid interval.

gridz

If the graphgrid attribute is present, then you can also add a gridz attribute, which specifies the z grid interval.

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.

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. For more information on traversing the tree, see Traversing the Tree.

hotlinkname

This attribute is just like the hotlink, except it links with the name of the node specified by the coldlink's path. It can be used in a static or edit control in the same way you would use a coldlinkname. For more information on traversing the tree, see Traversing the Tree.

hotlinkx

This attribute can be used in a static, edit, or checkbox control in the same way you would use a coldlinkx, except that it will dynamically update like a hotlink. For more information on traversing the tree, see Traversing the Tree.

itemcurrent

This attribute has many diverse uses and purposes in GUI building. Below is a list of the different areas of GUI building that use itemcurrent and the attributes purpose in each case.

Check Box: 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.

Radio Button: 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.

Combobox: The itemcurrent attribute is also a required attribute. It should have number data, and it specifies which attribute is currently selected.

Listbox: The itemcurrent attribute is also required. It also acts just like the itemcurrent attribute of the combobox.

Tracker: The itemcurrent attribute is a required attribute. Its value holds the current value that has been selected for the tracker.

Tabcontrol: 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.

items

This attribute has many diverse uses and purposes in GUI building. Below is a list of the different areas of GUI building that use items and the attributes purpose in each case.

Combobox: 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.

Listbox: 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.

menucustom

In a dialog view, 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.

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.

menuview

In a dialog view, 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.

noformat

This attribute has many diverse uses and purposes in GUI building. Below is a list of the different areas of GUI building that use noformat and the attributes purpose in each case.

Script: 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.

Table View: 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.

Tree View: If this attribute is present, then when text data is edited it will not be formatted for code.

objectfocus

In a dialog view, 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.

pagelist

In a tabcontrol view, 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 Properties windows.

palettewindow

In a dialog view, 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.

pickcopydataonly

In a combobox view, 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 affects 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 "If Item Type 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 themselves just implement the relevant code and not worry about the headers. This also allows for easy maintenance if the header changes. 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.

pickitem

In the combobox view, 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 Properties page uses this attribute to tie the "Arrival Style" combobox to the Source's arrivalmode variable.

picklist

In the combobox view, 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.

pickprimary

In the combobox view, this attribute is used with pickcopydataonly, as explained above.

picture

In the icongrid view, this is used in conjunction with viewfocus, and should contain a path to a picture file that will be displayed in the icongrid.

rangeexp

In the tracker view, 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.

rangemin

In the tracker view, the rangemin attributes specify the minimum values of the tracker. They should have number data specifying the minimum value.

rangemax

In the tracker view, the rangemax attributes specify the maximum values of the tracker. They should have number data specifying the maximum value.

spatialsx

This specifies the X size of the control it is on.

spatialsy

This specifies the Y size of the control it is on.

spatialx

This specifies the X location of the control it is on.

spatialy

This specifies the Y location of the control it is on.

statusbar

If this attribute is added to a 3D view and has a numerical value of 1, then a status bar will be drawn at the base of the view. An example of this can be found at the base of the standard 3D view.

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.

viewbackgroundcolor

This attribute defines the text background of the control. For a 3D or planar view, it defines the background color of the scene. The attribute should have 3 sub-nodes, each with number data between 0 and 1 corresponding to the red, green and blue components respectively. This attribute can be added to almost every view.

This attribute specifies the connection colors. The attribute should have 3 sub-nodes, each with number data between 0 and 1 corresponding to the red, green and blue components respectively.

viewfar

This attribute defines the far clipping plane distance.

viewfield

This attribute defines the field-of-view angle in degrees.

viewfirstperson

This attribute uses the firstperson camera mode in a view.

viewfocus

This attribute has many diverse uses and purposes in GUI building. Below is a list of the different areas of GUI building that use viewfocus and the attributes purpose in each case.

Graph View: 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 subnodes, 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.

Icongrid View: 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.

Ortho/Perspective View: 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.

Planer View: 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.

Table View: The viewfocus attribute is required for the table control. It specifies what the table will "point at". It is similar to 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.

Tree View: 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.

viewfog

This attribute defines the fog density factor. It is normally a number between 0 and 1, where 0 is equivalent to none and 1 is equivalent to the maximum value.

viewfont

This attribute only applies to Windows Common Controls. It defines the font that the control will use. The viewfont attribute is a wrapper around the Windows API CreateFont() command. Add this attribute and give it sub-nodes defining the different parameters that will be passed into CreateFont(), as shown in the figure below. You can refer to Microsoft's Windows API documentation for more information on this.

viewfull

This attribute is used to specify that the window it is placed is does not have a title bar.

viewhideallbases

This attribute hides all objects 2D shapes in a view.

viewhideallconnectors

This attribute hides all port connectors in a view.

viewhidealldrawcontent

This attribute hides all 3D shapes in a view.

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.

viewhiderouting

This attribute hides all object routing lines in a view.

viewlightaspos

This attribute sets the light as directional or a point. To set it the light to directional, the value needs to be 0. To set it the light to a point, the value needs to be 1.

This attribute must be the parent of a list of nodes each having viewlightx, viewlighty, viewlightz, viewlightr, viewlightg, viewlightb, viewlightaspos attributes. The subnodes define the position and color of directional lights or point light sources used in the 3D view.

viewlightr

This attribute corresponds to the red color component of the light.

viewlightg

This attribute corresponds to the green color component of the light.

viewlightb

This attribute corresponds to the blue color component of the light.

viewlightx

This attribute sets the x position of a point light source or the x component of a directional light source.

viewlighty

This attribute sets the y position of a point light source or the y component of a directional light source.

viewlightz

This attribute sets the z position of a point light source or the z component of a directional light source.

viewmagnification

This attribute is only used with ortho views and planer views, and specifies the magnification or "zoom" of the view.

viewnear

This attribute defines the near clipping plane distance.

viewpointradius

This attribute is only used with perspective views, and specifies the camera's distance from the focal point.

viewpointrx

This attribute specifies the RX viewing angle of the view.

viewpointry

This attribute specifies the RY viewing angle of the view.

viewpointrz

This attribute specifies the RZ viewing angle of the view.

viewpointx

This attribute specifies the X focal point of the view.

viewpointy

This attribute specifies the Y focal point of the view.

viewpointz

This attribute specifies the Z focal point of the view.

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.

viewshowgrid

This attribute enables or disable the showing of grid in a view. It should have number data with the value of 1, meaning show the grid and the value of 0, meaning don't show the grid.

viewshowheads

This attribute shows or hides all node ranks and node indices in a view. It should have number data with the value of 1, meaning show the node ranks and the value of 0, meaning hide the node ranks.

viewsnaptogrid

This attribute enables or disable the grid snapping in a view. It should have number data with the value of 1, meaning enable snap to grid and the value of 0, meaning disable snap to grid.

viewsyncupdate

This attribute enables or disable the repainting of all views when object editing is done. It should have number data with the value of 1, meaning enable repainting and the value of 0, meaning disable repainting.

viewwindowclean

This attribute specifies that when the window is closed, the underlying view tree is to be deleted. It is automatically managed by the FlexSim engine.

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.

viewwindowtype

This is the type of the window. Window types have specific values associated with them. When you create a window or drag out a control on the window it is given this attribute with a numeric value representing the window type. Window types are associated with window controls like buttons, radio buttons, check boxes, edits, listboxes, etc. As well as controls like groupbox, panel, table, graph, tree icongrid, etc. All the window types can be found in the online documentation in the View Attributes Reference section.

windowtitle

This attribute defines the title of the window. It should have text data defining the title. Otherwise the viewfocus path is used.

wordwrap

This attribute specify wrapping text in the editor. If the value is 0, then there is no text wrapping.

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.

Topcis

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. 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.

Tthe 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.

Windows Common Controls

- Button
- Checkbox

- Combobox
- DateTimePicker
- Edit
- Listbox
- Radiobutton
- Scrollbar
- Spincontrol
- Static (or label)
- Statusbar
- Tabcontrol
- Trackbar
- Treeview

FlexSim Registered Controls

- 3D View (Ortho/Perspective)
- Dialog (i.e. a Properties window)
- Graph
- Groupbox
- HTML
- IconGrid
- Panel
- Table
- Tree

Other Controls Used by FlexSim

- Script (code editor)
- Histogram
- Pie Chart
- Line Chart, Bar Chart, Scatter Plot

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.

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 container 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 container 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 container 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. For more information on traversing the tree, see Traversing the Tree.

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. For more information on traversing the tree, see Traversing the Tree.

coldlinkname: This attribute is just like the coldlink, except it links with the name of the node specified by the coldlink's path. For more information on traversing the tree, see Traversing the Tree.

hotlinkname: This attribute is just like the hotlink, except it links with the name of the node specified by the coldlink's path. For more information on traversing the tree, see Traversing the Tree.

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. For more information on traversing the tree, see Traversing the Tree.

hotlinkx: This attribute is just like the coldlinkx, except that it is refreshed every time the window is repainted. For more information on traversing the tree, see Traversing the Tree.

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 popup. 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 the 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 popup 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 popup 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 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.

To create a modal window, use viewwindowtype 4 (Dialog) and add the subnode *FS_MODAL*.

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.

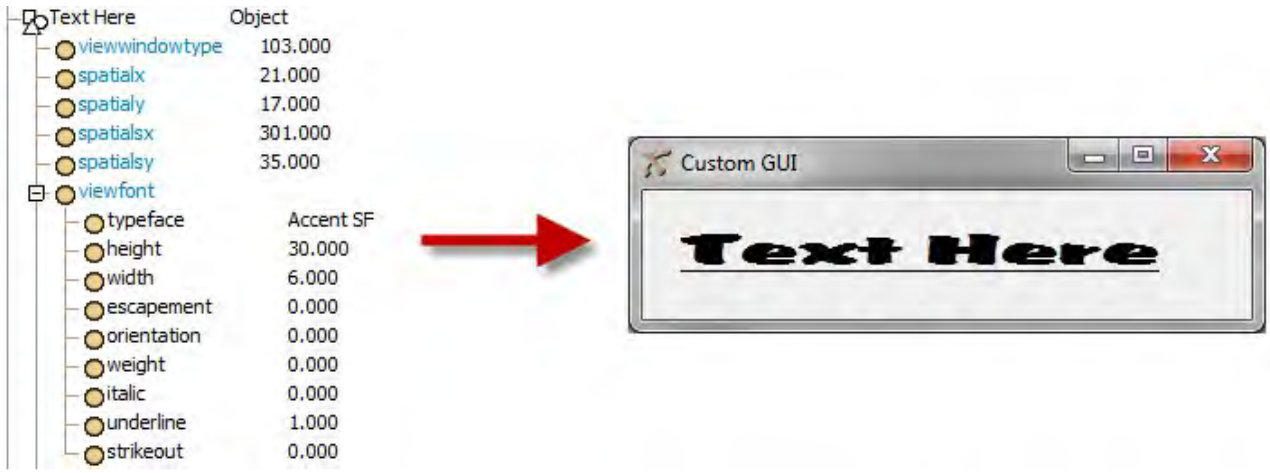
OnMouseWheelDelta: This attribute is used as described in the OnMouseWheel attribute above.

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().

OnFocus: This attribute is fired when the control receives keyboard focus. You can access the control that just lost focus using nodefromwindow(eventdata).

OnKillFocus: This attribute is fired when the control loses keyboard focus.

viewfont: This attribute only applies to Windows Common Controls. It defines the font that the control will use. The viewfont attribute is a wrapper around the Windows API CreateFont() command. Add this attribute and give it sub-nodes defining the different parameters that will be passed into CreateFont(), as shown in the figure below. You can refer to Microsoft's Windows API documentation for more information on this.



viewbackgroundcolor: For windows common controls, the viewbackgroundcolor attribute defines the text background of the control. For a 3D view, it defines the background color of the scene. The attribute should

have 3 sub-nodes, each with number data between 0 and 1 corresponding to the red, green and blue components respectively.

color: For Windows Common Controls, the color attribute defines the text color of the control. The attribute should have 3 sub-nodes, each with number data between 0 and 1 corresponding to the red, green and blue components respectively.

Control Specific Attributes

3D (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.

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. The attribute needs no data.

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. The file can be a bmp, jpg, png, or ico file.

coldlink, hotlink, coldlinkx, hotlinkx: The link 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.

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.

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 the OnPress 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.

OnPress: The OnPress attribute is a trigger that fires when the user clicks the arrow at the right of the view, right before the combo box displays its list of items. This is a good place to execute code to dynamically change the contents of items.

DateTimePicker

format: This attribute specifies how the DateTimePicker will display the given date or time. This attribute only works if the view has the DTS_APPCANPARSE style. For more information about format options, see the Model Settings page.

coldlink/coldlinkx/hotlink/hotlinkx: This attribute should be a link to a node with number data on it, or return a number respectively. The numerical value is the date/time value that will be displayed. The number should be in unix time, which is the number of seconds since Jan 1st 1601. For example, 13024569600 would be 8:00:00 AM Wed 25 Sep 2013 You can get the current unix time with the command applicationcommand("getunixtime").

OnSelect: The OnSelect attribute lets you execute flexscript code when the user changes the date or time.
OnKillFocus: The OnKillFocus attribute lets you execute flexscript code when the user clicks off of the datetimepicker.

style: The following styles are available to this view: DTS_TIMEFORMAT, DTS_LONGDATEFORMAT, DTS_SHORTDATEFORMAT, DTS_SHORTDATECENTURYFORMAT, DTS_UPDOWN, DTS_APPCANPARSE.

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. style: To create a modal window, add the subnode *FS_MODAL* to the style attribute.

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.

Groupbox

The groupbox is a control that groups several other controls together with a heading an a border around it. There are no special attributes for the groupbox.

Edit

The edit control shows text that can be edited by the user. If no special attribtes 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 by 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. ES_NUMBER will cause the field to only accept numbers and gives an alert if the user tries to enter a character.

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.

HTML

The HTML window is used in GUIs such as the Start Page, Online Content Page, and Dashboards. html:

The html node in the view's variables node will be the html that is displayed on the view.

OnPreLoad: The OnPreLoad node in the view's eventfunctions node (must be toggle as Flexscript) will be executed prior to the html being loaded on the page.

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 draggable 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. The file can be a bmp, jpg, png, or ico file. 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 incongrid'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, 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 dropnodeto() if it exists.

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.

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.

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. The file can be a bmp, jpg, png, or ico file.

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, splittery: The splitterx and splittery 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 subcontrols 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 splittery panel to show the template code and actual code simultaneously (VIEW:/standardviews/picklistedit).

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 the contains a path to a node with sub-nodes. Each sub-node's name will be show 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 the each sub-node's value represents a chunk of the pie.

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.

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. 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.

Scrollbar

OnDrag: This attribute allows you to execute code when the scroll bar is dragged.

vertical: If this attribute is added, the scrollbar will be a vertical scrollbar. This attribute needs no data.

Spincontrol

viewfocus: This attribute defines what the field the spinner is tied to.

OnClick: This attribute is a Flexscript toggled node in the eventfunctions node. It is called when the user mouse downs and mouse ups on the control.

OnMouseMove: This attribute is a Flexscript toggled node in the eventfunctions node. It is called after the user mouses down on the control, and before they mouse up.

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 and 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 can be a bmp, jpg, png, or ico file. If the file is a bmp, you can specify within the bitmap file certain areas a "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.

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.

Tabcontrol

The tabcontrol is a control that contains sub-controls that are each a tab page. To 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 Properties 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.

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.

Trackbar

The trackbar control is a control that allows you set numeric values by visually dragging a locator along a trackbar. This control is also known as a slider. Below are attributes that can be used with the trackbar.

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.

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.

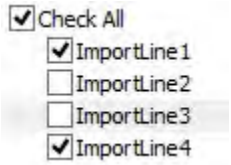
viewfocus: The viewfocus specifies the focus of the tree view. It should have text data with a path to the node that will be the head of the tree. cellwidth: The cellwidth attribute lets you specify how wide, in pixels, each node's name will be shown. noformat: If this attribute is present, then when 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.

Treeview

This control is not to be mistkan for the FlexSim Tree. The treeview control can be seen in such GUIs as the Excel Interface and the Breakpoints window.



coldlinkx,hotlinkx: These attributes can be used to execute code to update the treeview.

items: The items attribute is required for this control. It contains all of the items that will be displayed in the control. IT is important that each item has number data. The check boxes set and get their state based on the number in each item (1 or 0). Items may have subnodes which will create a parent/child display in the treeview.

itemcurrent: The itemcurrent attribute is required for this control. It contains the currently selected item.

OnPress: The OnPress trigger is called when the user clicks on a check box.

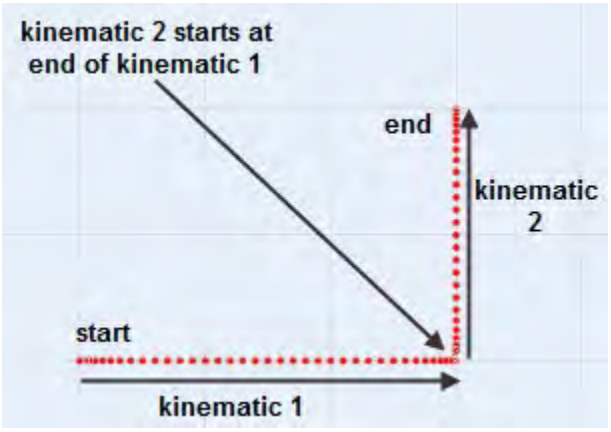
OnSelect: The OnSelect trigger is called when the user selects one of the items in the tree.

style: Styles that are important with this control are TVS_CHECKBOXES, TVS_SHOWSELALWAYS, TVS_FULLROWSELECT, TVS_DISABLEDRAGDROP, TVS_EDITLABELS, TVS_HASBUTTONS, TVS_HASLINES, TVS_LINESATROOT, TVS_SINGLEEXPAND.

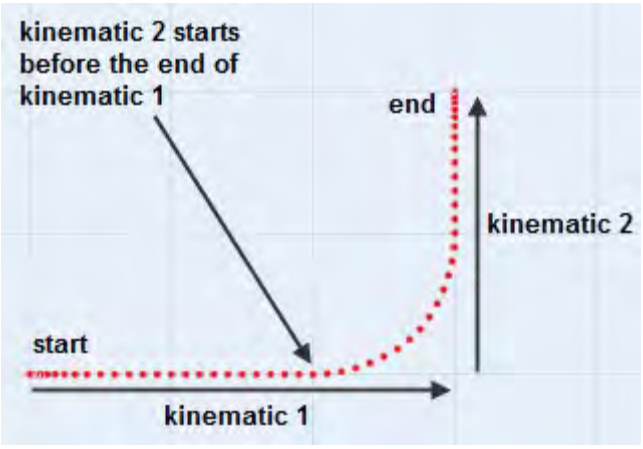
Kinematics Concepts

FlexSim's kinematics functionality allows you to have a single object perform several travel operations through one common interface. Each travel operation can have its own acceleration, deceleration, startspeed, endspeed, and maximum speed properties. Travel operations can overlap with each other, or be performed in sequence.

Below is a time-based plot of two kinematics performed in sequence:



And a time-based plot of two kinematics that overlap each other:



Kinematics can also be used just to make it easier to handle the math for any logic that involves speeds, rates, accelerations, decelerations, etc.

As an 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. Using kinematics, you can define all of these motions through a single kinematics interface, where different motors can work simultaneously, giving the motion of the crane's end effector a very dynamic behavior. 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. While this works very well in some cases, in other instances it can become tedious and unfriendly. Kinematic functionality attempts to fix this problem by allowing one object to do several motions or kinematics simultaneously.

Commands

1. initkinematics - This initializes data for the kinematics, saving things like the start location and rotation of the object you want to apply motion to.
2. addkinematic - This gives travel/rotate operations to the object. For example, you can tell the object, starting in 5 seconds and 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, 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 call to addkinematic will add another operation to the object.
3. updatekinematics - This command should be called when the object is being redrawn. This calculates the current position and rotation of the kinematics based on the current time, and sets the object's location to that position.

Refer to the Commands page for information about kinematics commands.

Kinematics Commands

Commands

- initkinematics
- addkinematic
- updatekinematics
- Other Kinematics Commands

initkinematics

The Blank Node

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.

Overloading

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 flags])

datanode - This is the blank node for kinematic data. object - This is the object that

will do the moving.

flags - This optional parameter defines various flags for the kinematic. It can be a bitwise or on the following values:

KINEMATIC_MANAGE_ROTATIONS - If this flag is set, the object will always point in the direction of travel.
KINEMATIC_RELATIVE_LOCS - If this flag is set, then the object will travel according to its local coordinate system, based on the rotation of the object at the time you initialize the kinematic. If not, kinematic coordinates are based on the object's container's coordinate system.
KINEMATIC_DO_NOT_PRUNE - By default, the kinematics logic will "prune" the kinematics as they go, meaning when you update the kinematics to a time that is after the end time of a given kinematic then it will remove that kinematic from the kinematics list. If you do not want it to do this, i.e. you may be doing queries across various times in the kinematic, then you should set this flag.
KINEMATIC_RESET_BUFFER - By default, the kinematic's memory allocation is optimized for speed, i.e. it will only reallocate memory when it needs it, and it will not resize back down when you initialize the kinematic. However, if this flag is set, it will resize its data buffer to an initial size when initialized.

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 flags])

datanode - This is the blank node for kinematic data.

x, y, z - These are optional initial location variables. Defaults: 0, 0, 0 rx, ry, rz - These are optional initial rotation variables. Defaults: 0, 0, 0 flags - See the first overload above for more information.

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.

void setkinematicsrotoffset(treenode datanode, double rx, double ry, double rz)

datanode - This is the blank node for kinematic data.

rx, ry, rz - These are initial rotation variables.

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).

addkinematic

---

double addkinematic(treenode datanode, double x, double y, double z, double targetspeed [, double acc, double dec, double startspeed, double endspeed, double starttime, int type])

datanode - This is the blank node for kinematic data.

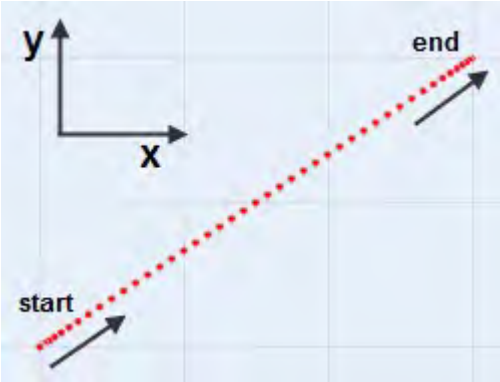
x, y, z - These are offset locations or rotations. targetspeed

This is the target travel speed.

acc - This optional parameter specifies the acceleration. Default: 0 (or infinite acceleration) dec - This optional parameter specifies the deceleration. Default: 0 (or infinite deceleration) startspeed - This optional parameter specifies the initial velocity. Default: 0 endspeed - This optional parameter specifies the ending velocity. Default: 0 starttime - This optional parameter specifies the absolute start time. Default: Current Time type - This optional parameter specifies the type of travel to apply (rotation or travel). Default: KINEMATIC_TRAVEL

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 that 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 will usually 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.

Below is a time-based plot of a kinematic that travels 3 in the x and 2 in the y, with accelerations and decelerations.



Turn Kinematic Types

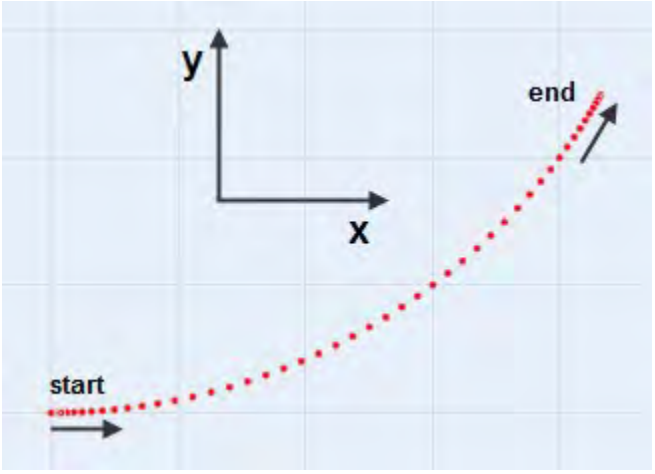
In addition to KINEMATIC_TRAVEL and KINEMATIC_ROTATE, there are three "turn" kinematic types, which cause the moving object to turn a certain angle with a given turn radius. These are KINEMATIC_TURN_XY, KINEMATIC_TURN_YZ, and KINEMATIC_TURN_ZX. For these kinematic types, the x, y, and z parameters that you pass into addkinematic() take on a different meaning, as follows:

x - The start angle in degrees. An angle of 0 means the initial direction of motion when the turn starts will be in the same direction as the first axis of the kinematic type's name, i.e. start angle 0 for KINEMATIC_TURN_XY means the initial motion is positive along the x axis, where for KINEMATIC_TURN_YZ the initial motion would be positive along the y axis. Use the right hand rule around a given axis when figuring out start and turn angles.

y - The turn angle in degrees. The motion of the object will turn this number of degrees, parallel to the plane defined by the two axes in the kinematic type's name, i.e. KINEMATIC_TURN_XY will turn in the x/y plane, i.e. around the z axis. z - The turn radius in distance.

All other parameters are the same for this kinematic type, i.e. max speed, acceleration, deceleration, etc. Speeds are in length units per time unit, NOT in degrees per time unit.

As an example, if you would like to make a motion that turns from travelling positive along the x axis to travelling at 60 degrees off the x axis, with a turn radius of 5 (i.e. turn about the z axis), call: addkinematic(kinematics, 0, 60, 5, 1, 1, 1, 0, 0, time(), KINEMATIC_TURN_XY);



updatekinematics

void updatekinematics(treenode datanode, treenode object [, double updatetime]) datanode - This is the blank node for kinematic data.

object - This is the object that will do the moving.

updatetime - This optional parameter specifies the absolute update time. Default: Current Time

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.

Other Kinematics Commands

double getkinematics(treenode datanode, int type [, int kinematicindex, double updatetime/traveldist]) datanode - This is the blank node for kinematic data. type - This specifies the type of information to retrieve (listed below).

kinematicindex - This optional parameter specifies which kinematic to query. Default: 0 (all kinematics) updatetme/traveldist - This optional parameter represents either the update time or the travel distance, depending on the parameter type. Default: Current Time

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.

Information	Value of type Parameter	Single Kinematic	All Kinematics
Location	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	These return the x, y, or z location of the object at the given update time.

		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.	
End Distance	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.
Total Distance	KINEMATIC_TOTALDIST	This returns the total distance for the kinematic operation.	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.
Total Component Distances	KINEMATIC_TOTALX KINEMATIC_TOTALY 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.	These return the sum of x, y, or z component of all added kinematics. Here the updatetime parameter is not used.

	This returns the cumulative travel distance of all added kinematics. Unlike 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. Note: for cumulative distance, if you are using turn kinematics, the turn kinematics may not overlap other travel or turn kinematics. If they overlap with others, then the cumulative distance calculation will be incorrect.
se return the x, y, or z omponent of the rent velocity for the cified kinematic at given update time.	These return the x, y, or z velocity of the object for the given time.
is returns the start ed for the kinematic. s is the startspeed ou specify in the inematic command. ere the updatetime ameter is not used.	

s returns the peak peed or "reached d" for the kinematic. is usually the same s the target speed pecified in the kinematic command, t may not be if the ematic cannot get to target speed given e distance it has to travel. Here the

		updatetime parameter is not used.	
End Speed	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.	
Total Velocity	KINEMATIC_VELOCITY	This returns the total velocity of the kinematic for the given time.	This returns a scalar value of the total velocity for the given time.

Acceleration (Starting)	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.	
Acceleration (Stopping)	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.	
Rotation	KINEMATIC_RX KINEMATIC_RY KINEMATIC_RZ	These return the x, y, or z rotation of a rotational kinematic at the given update time.	These return the x, y, or z rotation of the object at the given update time. This will only work if rotations are managed manually.
End Rotational Distance	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 rotations are managed manually. Here the updatetime parameter is not used.

Total Rotational Distance	KINEMATIC_TOTALRDIST	This returns the total rotational distance for the kinematic operation if it is a rotational kinematic.	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.
Total Component Rotations	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.	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.
Cumulatvie Rotational Distance	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.

Rotational Velocity	KINEMATIC_VRX KINEMATIC_VRY KINEMATIC_VRZ	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.	These return the rotational x, y, or z velocity of the object for the given time.
Total Rotational Velocity	KINEMATIC_RVELOCITY	This returns the total rotational velocity of the kinematic for the given time if it is a rotational kinematic.	This returns a scalar value of the total rotation velocity for the given time. This only works if rotations are managed manually.

Start Times	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.	This returns the lowest start time of all the kinematics. Here the updatetime parameter is not used.
Acceleration Time (Starting)	KINEMATIC_ACC1TIME	This returns the total time the kinematic will spend accelerating/decelerating from the start speed to the target speed. Here the updatetime parameter is not used.	
Arrival Time	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.	
Peak Time	KINEMATIC_PEAKTIME	This returns the total time the kinematic will spend traveling at the peak speed. Here the updatetime parameter is not used.	

Acceleration Time (Stopping)	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.	
End Times	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.	This returns the highest end time of all the kinematics. Here the updatetime parameter is not used.
Number of Kinematics	KINEMATIC_NR		This returns the number of kinematics that have been added. Here the updatetime parameter is not used.
Turn Kinematic Data	KINEMATIC_STARTANGLE KINEMATIC_TURNANGLE KINEMATIC_TURNRADIUS	These return the start angle, turn angle, and turn radius of a turn kinematic.	
Kinematic Type	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]) datanode - This is the blank node for kinematic data.

index - This optional parameter specifies the index of a specific index to print information for.
Default: 0 (all kinematics)

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) datanode - This is the blank node for kinematic data.

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.

Sampler Concepts

Overview

The Sampler is a new feature in Version 7. The Sampler is a referencing tool and is convenient for referencing objects, their variables, labels or other properties. It eliminates some need for writing code and makes model building faster and easier.

The Sampler can be used to sample or reference various things throughout FlexSim, including:

- Nodes/Objects
- Numbers
- Colors
- 3D Shapes
- Images
- Paths
- Strings

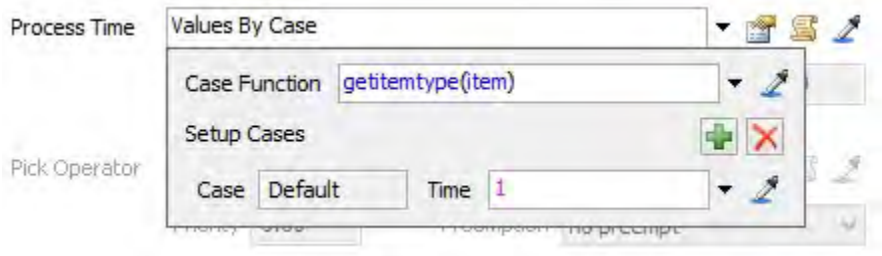
When referencing through the Code Edit window or through picklists, the Sampler will insert correct code for you.

The Sampler is context sensitive, so it will only allow you to sample things in FlexSim that are valid. For instance, if you're trying to specify the Process Time of a Processor, the Sampler will only give you options that return numerical values, like the current.numberLabel, gettablenum, or getvarnum commands. If you are trying to reference an object for the Use Transport option of a FixedResource object, the Sampler will only sample objects and return values like centerobject(current, 1), node("/Operator1", model()).

You can even set values like a label from a Flowitem if it fits the context, simply by sampling the object and choosing the appropriate label from the list.

The Sampler is capable of sampling values from the 3D view as well as in Object Properties Windows, Quick Properties, Tree Windows and other windows throughout FlexSim.

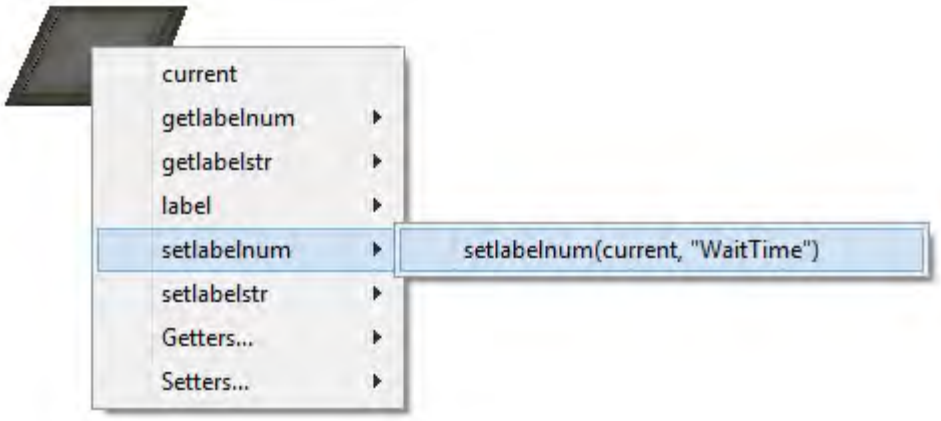
You'll find the Sampler in many places throughout the interface. Below is a picklist option for a Processor's Process Time:



Using the Sampler

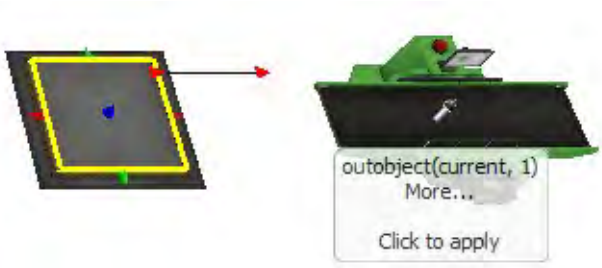
To use the Sampler, you simply left-click once on the to enter "Sample Mode". As you move the mouse over objects and fields, the mouse will display available options based on the current context. If an object has no valid data for the current context, no options will be displayed.

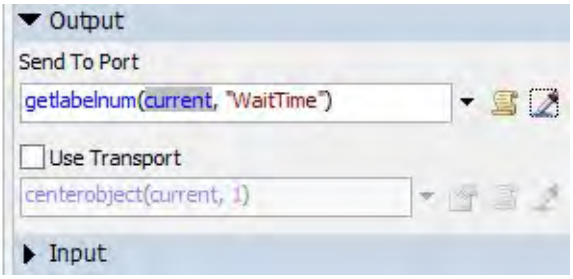
If more than one option is available, a menu will appear allowing you to select the code you wish to use.



Highlighting Code

You can also highlight portions of code you want to change and, instead of figuring out exactly how you're going to reference it, just sample what you need.





Sampling Colors

All of the Sampling functions are restricted to within FlexSim, except for color samplers. If you want to sample a color from another application you can click on a color sampler and click on the desired color outside of FlexSim. Because the mouse is outside of FlexSim, the RGB values will not be displayed under the mouse, however, the Sampler will pull the correct color value under the mouse.

Examples

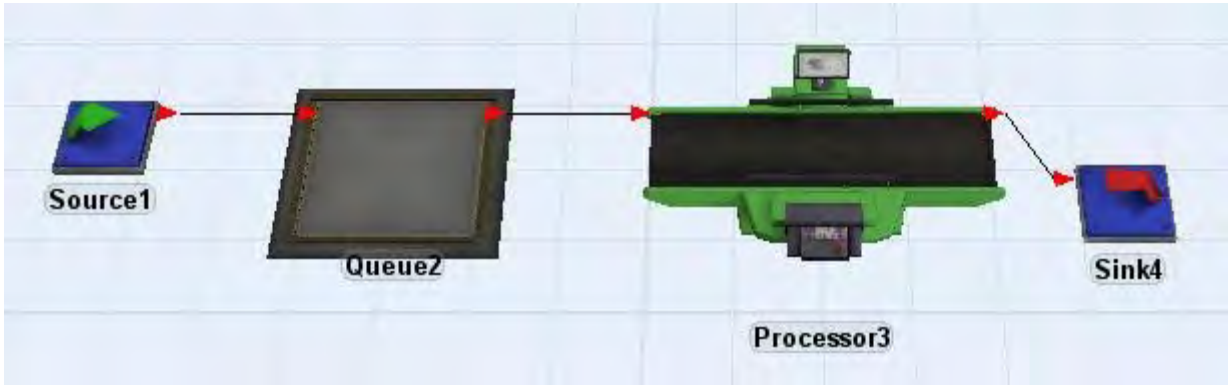
Here are a few examples of where the Sampler is used:

- Referencing a Transporter in an object's flow.
- Defining Setup/Process and Load/Unload Times.
- Choosing object colors.
- Setting object and shape frame 3D shapes.

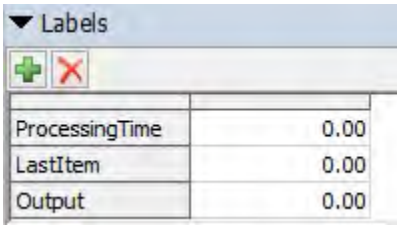
- Writing code in the Code Editor.
- Referencing objects and numbers in triggers and picklists.
- Referencing Global Table cells.

In this example, we'll show you how to set a processor's Process Time dynamically based on a label.

Create a Simple Model



- Create the simple model as shown above.
- Click on the Processor to display it's properties in the Quick Properties window • Press the in the Labels section of the Quick Properties and add some number labels.

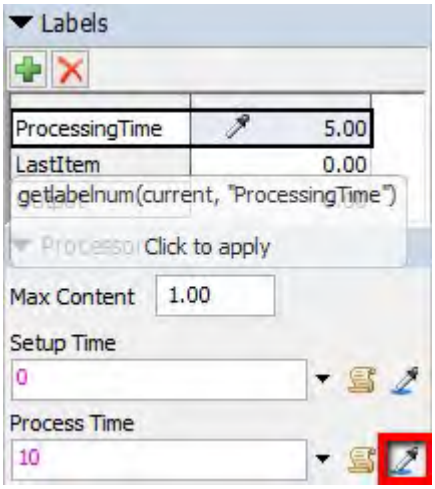


- Set the ProcessingTime label to 5.

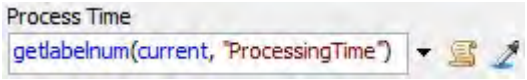
Sample a Label

In the Quick Properties window, below labels, you'll see a Processor section.

- Click the next to Process Time to enter "Sample Mode".
- Click on the *ProcessingTime* label you just created to reference that label.



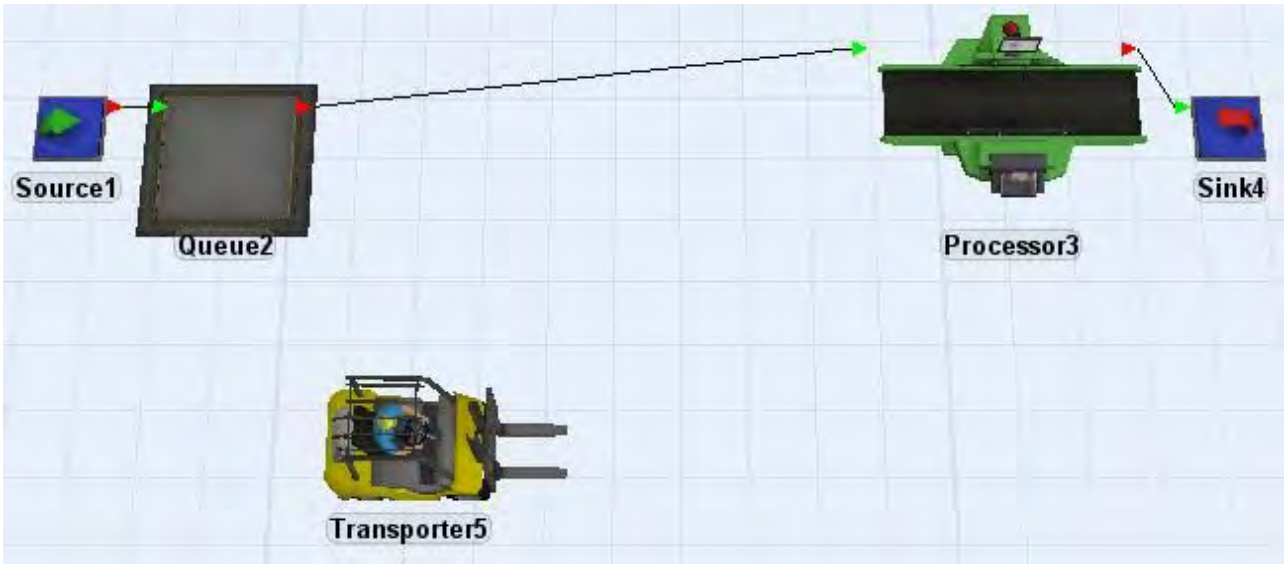
The Sampler will automatically insert the correct line of code for referencing the *ProcessingTime* label.



Reset and Run the model to see how it works.

In this example, we'll show you how reference a transporter object to move flowitems from a Queue to a Processor.

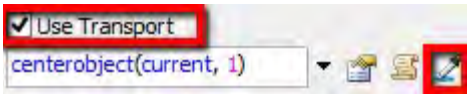
Create a Simple Model



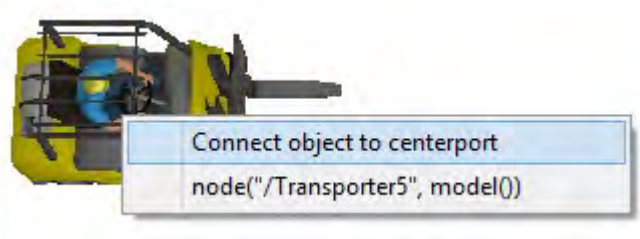
- Create the simple model as shown above.

Assigning a Transporter

- Click on the Queue to display its properties in the Quick Properties window.
- Under the Output section of the Quick Properties, check the Use Transport button.
- Next to the transport reference picklist, click on the to enter "Sample Mode".



- Click on the Transporter in the 3D view. A menu will appear with the following options:

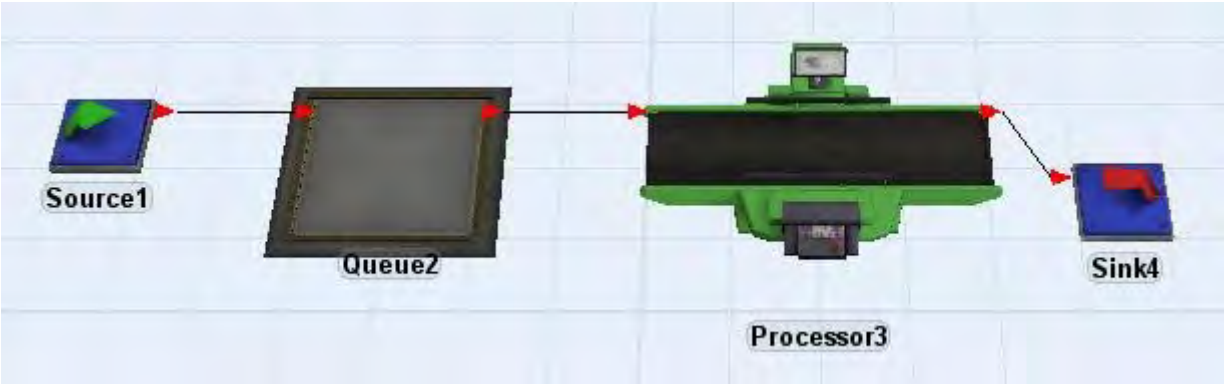


- Select the Connect object to centerport option.

This is will automatically connect the Queue to the Transporter and create the correct reference to the object. Reset and Run the model to see how it works.

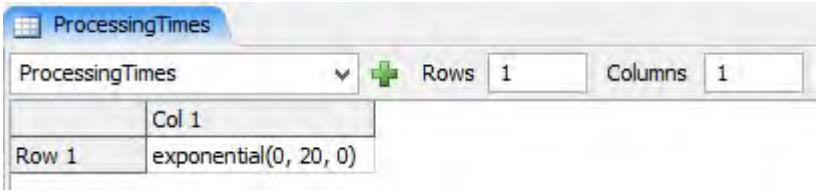
In this example, we'll show you how to set a processor's Process Time dynamically based on a cell in a Global Table.

Create a Simple Model



- Create the simple model as shown above.
- Add a Global Table through the Toolbox Add > Global Table.
- Change the name of the table to *ProcessingTimes*.
- Right click the first cell and column (0.00) of the Global Table and select Assign String Data.
- Enter the following text:

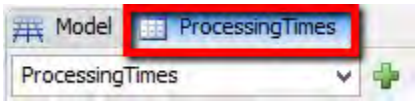
exponential(0, 20, 0)



Arranging Windows

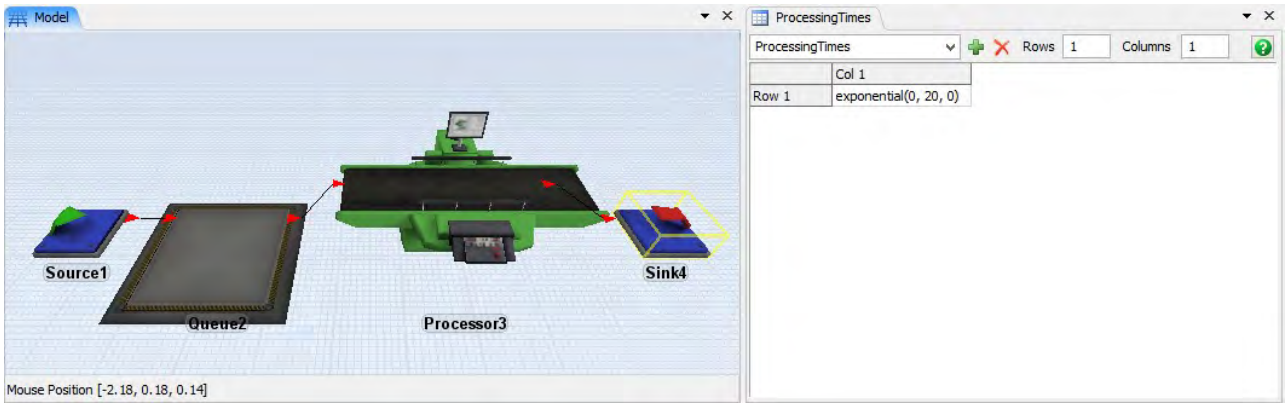
In FlexSim, windows can be arranged in any configuration. Though there are multiple ways to sample a cell in a Global Table, we will look at just one way.

- Click and drag the ProcessingTimes table tab to "pull" the Global Table window out of the docked view.



- While still holding the mouse down, drag the mouse over the Dock Windows icon that has appeared in the middle of your main window.

Your main document window will now have your 3D view on the left and your Global Table on the right.

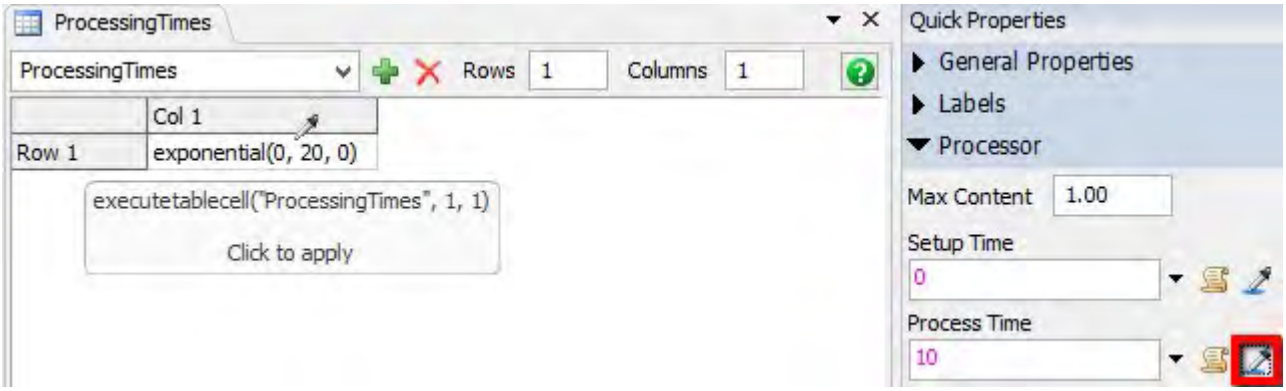


Sample a Global Table Cell

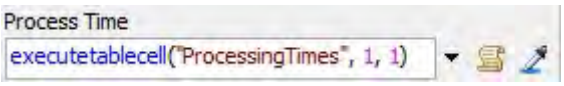
- In the 3D view, click on the Processor object to display its properties in the Quick Properties window.

In the Quick Properties window you'll see a Processor section.

- Click the next to Process Time to enter "Sample Mode".
- Click on the first row and column of the *ProcessingTimes* Global Table to reference that cell.



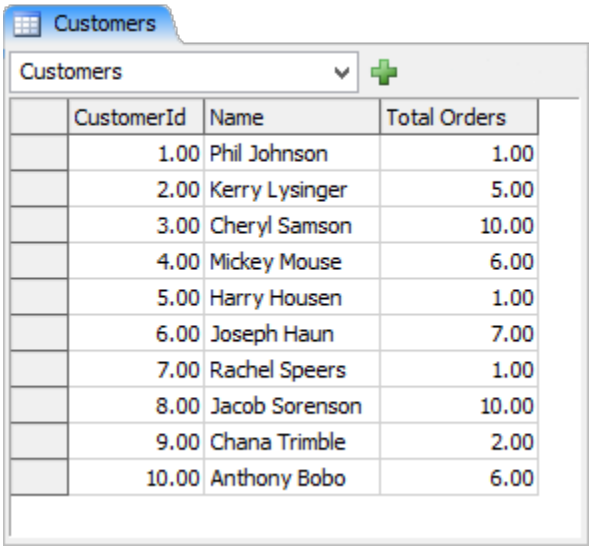
The Sampler will automatically insert the correct line of code for referencing the Global Table cell. Notice that the value the Sampler was looking for was a number (Process Time) and the Global Table cell contained string data. The Sampler automatically added in an executetablecell() command in order to return a numeric value from the cell, in this case exponential(0, 20, 0).



Reset and Run the model to see how it works.

FlexSim provides a flexible method for querying, filtering and prioritizing data in a model using the SQL language. Using SQL queries, users can do standard SQL-like tasks, such as searching one or more tables in FlexSim to find data that matches criteria as well as prioritizing that data. Instead of using FlexScript code to manually loop through and sort table data, you can write a single SQL query using the Table.query() command to do all the work for you.

For example, let's say you have a global table in your model that looks like the following:



	CustomerId	Name	Total Orders
	1.00	Phil Johnson	1.00
	2.00	Kerry Lysinger	5.00
	3.00	Cheryl Samson	10.00
	4.00	Mickey Mouse	6.00
	5.00	Harry Housen	1.00
	6.00	Joseph Haun	7.00
	7.00	Rachel Speers	1.00
	8.00	Jacob Sorenson	10.00
	9.00	Chana Trimble	2.00
	10.00	Anthony Bobo	6.00

And let's say you want to search that table to find the name and id of the customer with the most total orders. The manual method for doing this in FlexScript would look like the following:

```
Table customers = Table("Customers"); int bestRow = 0; int
highestTotalOrders = 0; for (int i = 1; i <= customers.numRows;
i++) {     int totalOrders = customers[i][3];     if (totalOrders >
highestTotalOrders) {         bestRow = i;
highestTotalOrders = totalOrders;
   } } int bestCustomerId = result[bestRow]["CustomerId"]; string
bestCustomerName = result[bestRow]["Name"];
```

Alternatively, using SQL and FlexSim's Table.query() command, you can do it much more simply, as follows:

```
Table result = Table.query("SELECT CustomerId, Name FROM Customers \              ORDER
BY [Total Orders] DESC LIMIT 1"); int bestCustomerId = result[1]["CustomerId"]; string
bestCustomerName = result[1]["Name"];
```

This is a pretty simple example. More complex tasks such as sorting multiple results or searching multiple tables would be much more complex to perform manually in FlexScript, whereas doing those complex searches using SQL is relatively easy once you understand the rules of SQL and how to write SQL queries.

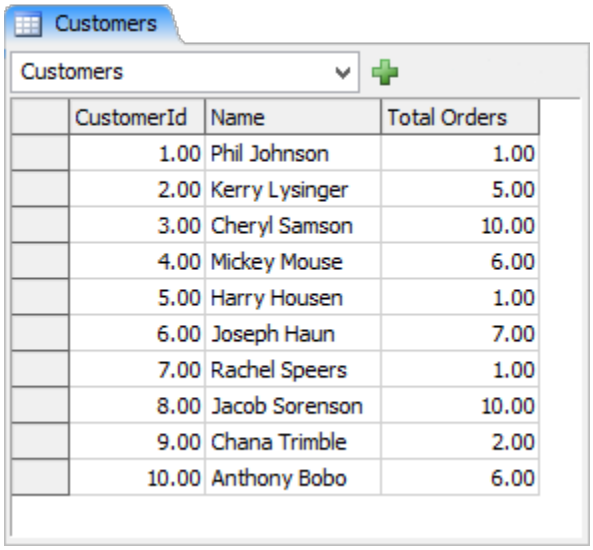
FlexSim's SQL functionality also includes advanced querying techniques. These allow you to do searches on data in the model that are not structured like standard tables. You can search flow items, task sequences, resources, really any data in a model that can be conceptualized into lists.

Your First SQL Query

In FlexSim, SQL queries are done using the Table.query() method:

static Table Table.query(str queryStr[, ...])

Often you will use the Table.query() command to search a single table. Take the following example global table.



	CustomerId	Name	Total Orders
	1.00	Phil Johnson	1.00
	2.00	Kerry Lysinger	5.00
	3.00	Cheryl Samson	10.00
	4.00	Mickey Mouse	6.00
	5.00	Harry Housen	1.00
	6.00	Joseph Haun	7.00
	7.00	Rachel Speers	1.00
	8.00	Jacob Sorenson	10.00
	9.00	Chana Trimble	2.00
	10.00	Anthony Bobo	6.00

Let's say you want to find all customers who have made more than 5 orders, sorted alphabetically by their name. To do that, you'd use the following command:

```
Table result = Table.query("SELECT * FROM Customers \
    WHERE [Total Orders] > 5 \     ORDER BY
Name ASC");
```

Some things to note:

- The SELECT statement tells the query what columns you want to pull out into your query result. By using SELECT * we are telling the query to put all the columns from the source table into the query result. Alternatively, if we were to use SELECT CustomerId, Name then that would put just the CustomerId and Name columns into the query result.
- The FROM statement defines the table that is to be queried. Often this will be just one table, but it may be multiple comma-separated tables, which effects a join operation, discussed later. Here we define FROM Customers. This causes the SQL parser to search the global table in the model named Customers.
- The WHERE statement defines a filter by which entries in the table are to be matched. The expression WHERE [Total Orders] > 5 means that I only want rows whose value in the Total Orders column is greater than 5. In a WHERE statement you can use math operators to define expressions, like +, -, *, / and logical operators like ==, != (also <>), <, <=, >, >=, AND, and OR.

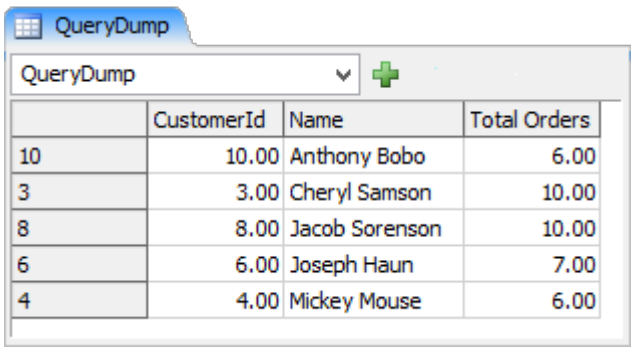
- The [ ] syntax is an SQL delimiter that allows you to define column names with spaces in them. If I had named the column TotalOrders instead of Total Orders, I could have just put TotalOrders for the column name in the query instead of [Total Orders].
- The ORDER BY statement defines how the result should be sorted. Here we sort by the Name column, which is a text column and thus will be sorted alphabetically. You can optionally put ASC or DESC to define if you want it sorted in ascending order or descending order. Default is ascending. You can also have multiple comma-separated expressions defining the sort. The additional expressions will define how to order things when there is a tie. For example, the expression ORDER BY [Total Orders] DESC, Name ASC would first order descending by the number of total orders, and then for any ties (Cheryl Samson and Jacob Sorenson both have 10 total orders) it would sort alphabetically by name.
- The \ escape character lets you extend a quoted string across multiple lines by using the \ at the end of a line within a string. Often with SQL queries, the query is too long to reasonably fit within a singleline quoted string. Also, using multiple lines with indentation can make the query more readable.
- A LIMIT statement, although not used in this example, can be added at the end of the query. This will limit the number of matches. If you only want the best 3 matches, add LIMIT 3 to the end of the query.

Getting Data Out of the Query

Now that you've done your query, the result is stored in a read-only in-memory instance of a Table. To copy the entire result to something like a global table, or a table on a node in the tree, you can use Table.cloneTo(). This will copy a query result to a table or bundle node. For this example we can create a global table named QueryDump, and then copy the result to that table.

```
Table result = Table.query("SELECT * FROM Customers \
      WHERE [Total Orders] > 5 \       ORDER BY
Name ASC"); result.cloneTo(Table("QueryDump"));
```

Dumping the above query to the global QueryDump table yields the following results:



	CustomerId	Name	Total Orders
10	10.00	Anthony Bobo	6.00
3	3.00	Cheryl Samson	10.00
8	8.00	Jacob Sorenson	10.00
6	6.00	Joseph Haun	7.00
4	4.00	Mickey Mouse	6.00

Table Selection

In the example above, we used "SELECT * FROM Customers". The FROM tells the parser to look in global tables for a table named Customers and search that table. Alternatively, you can explicitly define the tables for search. This is done by using a special '$' identifier in the query and by passing additional parameters into the Table.query() command. For example, I could have defined the exact same query as above, but instead defined the customers table explicitly as follows:

```
Table result = Table.query("SELECT * FROM $1 \
      WHERE [Total Orders] > 5 \
      ORDER BY Name ASC",
   Table("Customers"));
```

Notice that instead of using the table "Customers" I now define the table as $1. What this means is that the table I want searched is the table I pass in as the first additional parameter of the Table.query() command. $2 would correspond to the table passed into the second additional parameter of the command, and so on. Tables passed as additional parameters can have either regular tree table data or bundle data. However, they should not be an in-memory result Table, like the result of another call to Table.query(). In

other words, they must be stored on a node in FlexSim's tree in order for FlexSim to process them properly.

By using this table specification method you are no longer bound to using global tables. For example, if the customers table happened to be on a label instead of a global table, the query is still pretty simple:

```
query("SELECT * FROM $1 \
      WHERE [Total Orders] > 5 \       ORDER BY
Name ASC",     current.labels["Customers"]));
```

FlexSim's SQL parser also allows you to simpify the SQL a bit for this single-table search scenario. The SQL standard requires a SELECT and FROM statement, but FlexSim's SQL parser isn't that picky. If you only pass one parameter as the table, it will automatically assume that you want to search the table you passed in. Thus you can leave out the SELECT and FROM statements. Leaving them out is essentially the same as using the statement SELECT * FROM $1.

```
query("WHERE [Total Orders] > 5 ORDER BY Name ASC", current.labels["Customers"]);
```

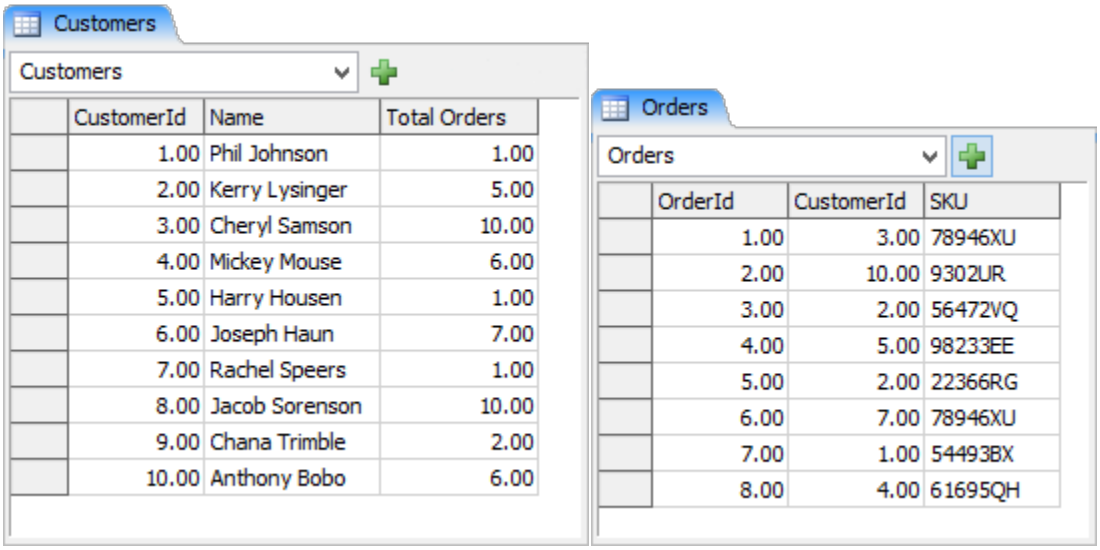
Additional Result Analysis Methods

While using Table.cloneTo() is often useful when you're setting up a model or when you're testing various queries, cloning an entire table/bundle with data is non-trivial and requires memory space and CPU time for creating the table. Alternatively, you can simply use the methods and properties provided by the Table class to process the result directly.

```
Table result = Table.query("SELECT * FROM Customers \
      WHERE [Total Orders] > 5 \       ORDER BY Name
ASC"); result.cloneTo(Table("QueryDump")); for (int i = 1;
i <= result.numRows; i++) {    string name =
result[i]["Name"];     ...
}
```

Joins

You can also use FlexSim's SQL parser to query relationships between multiple tables. To demonstrate this, we'll do another example using the Customers table and an additional Orders Table.



Customers

CustomerId	Name	Total Orders
1.00	Phil Johnson	1.00
2.00	Kerry Lysinger	5.00
3.00	Cheryl Samson	10.00
4.00	Mickey Mouse	6.00
5.00	Harry Housen	1.00
6.00	Joseph Haun	7.00
7.00	Rachel Speers	1.00
8.00	Jacob Sorenson	10.00
9.00	Chana Trimble	2.00
10.00	Anthony Bobo	6.00

Orders

OrderId	CustomerId	SKU
1.00	3.00	78946XU
2.00	10.00	9302UR
3.00	2.00	56472VQ
4.00	5.00	98233EE
5.00	2.00	22366RG
6.00	7.00	78946XU
7.00	1.00	54493BX
8.00	4.00	61695QH

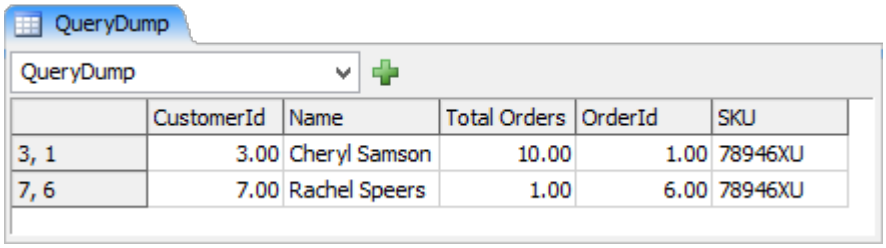
In this example we want to find information associated with customers who ordered SKU 78946XU. For each order of SKU 78946XU we want to know the customer's name, the CustomerId, and the OrderId. Below is the query:

```
Table result = Table.query("SELECT * FROM Customers, Orders \
    WHERE \
      SKU = '78946XU' \
      AND Customers.CustomerId = Orders.CustomerId");
```

Things to note:

- Here we use the statement FROM Customers, Orders. In SQL this is called an inner join. The SQL evaluator compares every row in the Customers table with every row in the Orders table to see which row-to-row pairings match the WHERE filter.
- The filter we define is WHERE SKU = '78946XU' AND Customers.CustomerId = Orders.CustomerId. We only want to match the rows in the Orders table that have the SKU value '78946XU'. Secondly, for those rows that do match the SKU, we only want to match them with rows in the Customers table that correspond with the same CustomerId in the matched row of the Orders table.
- Notice that for the SKU rule we just say SKU = '78946XU'. For SKU, since the Orders table is the only table with an SKU column, the SQL evaluator will automatically recognize that the SKU column is associated with the Orders table. We could explicitly define the table and column with Orders.SKU, and sometimes that is preferable in order to make the query more readable/comprehensible. However, if you leave it out the evaluator will happily figure out the association on its own.
- The CustomerId rule, on the other hand, uses the . (dot) syntax to explicitly define table and column. This is because both the Orders table and the Customers table have columns named CustomerId, and we want to explicitly compare the CustomerId column in Customers with the CustomerId column in Orders. So we use the dot syntax to define table and column.

The result of the result.cloneTo(Table("QueryDump")) for this query:



	CustomerId	Name	Total Orders	OrderId	SKU
3, 1	3.00	Cheryl Samson	10.00	1.00	78946XU
7, 6	7.00	Rachel Speers	1.00	6.00	78946XU

For explicitly defined tables (labels for example) you'd use a query like:

```
Table result = Table.query("SELECT * FROM $1 AS Customers, $2 AS Orders \
      WHERE \
         SKU = '78946XU' \
         AND Customers.CustomerId = Orders.CustomerId",
current.labels["Customers"],      current.labels["Orders"]);
```

Aliases

In the previous example we use the AS construct to create an alias for our table. You can create aliases for both tables and column references. This can increase readability of the query especially if you are using the $ syntax. For table aliases you do not technically need the AS qualifier. Both of the following are valid:

```
SELECT * FROM $1 AS Customers, $2 AS Orders
```

```
SELECT * FROM $1 Customers, $2 Orders
```

Once an alias is defined you should use that alias instead of the table name in other references in the query.

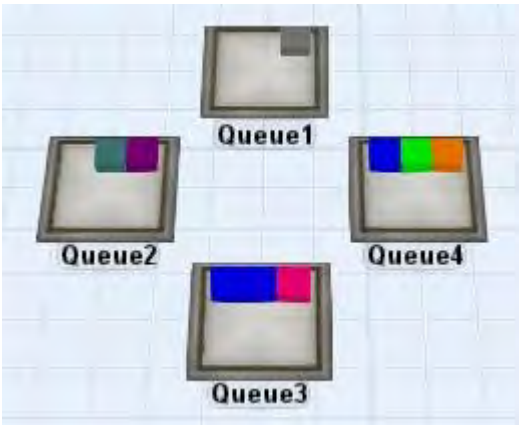
Below are several examples of defining column aliases.

```
SELECT Customers.CustomerId AS ID, Customers.Name AS Name  WHERE ID > 5
AND ID < 10
```

Advanced Query Techniques

Often there are decision-making problems in a simulation that lend themselves well to using SQL-like constructs, such as searching, filtering and prioritizing. However, since a simulation is an inherently dynamic system, data in the simulation is often not structured in the standard way that a database is structured. FlexSim's advanced SQL querying functionality aims to bridge this gap, allowing modelers to use SQL's flexibility and expressiveness in dynamically querying the state of a model.

Let's take an example where you have flow items that queue at various locations in the model. It is quite often the case that modeling logic will try to search for a "best" flow item, and determining which flow item is the best may involve complex rules. It might assign certain eligibility criteria for each flow item. It could also weigh various things like the distance from some source to the location of the flowitem, the time that the flow item has been waiting in queue, an assigned priority for the flowitem, etc. For these types of problems, the SQL language is quite expressive and makes the problem relatively easy. So what if you could represent all of the candidate flowitems as a kind of quasi-database table, and then use SQL to search, filter, and prioritize entries in that table?



Here there are four queues, each with a set of flow items in it. Imagine each of those flow items has various labels defining data on that flowitem. A table of flow items representing all of that data might look something like the following:

Item	Location	DistFromMe	Priority	Step	ItemType	Time Waiting
GrayBox	Queue1	9.85	3	5	6	5.4
PurpleBox	Queue2	8.5	2	2	8	8.1
TealBox	Queue2	8.5	8	12	5	7.2
OrangeBox	Queue4	12.5	4	1	4	1.2
GreenBox	Queue4	12.5	3	5	2	4
BlueBox	Queue4	12.5	6	6	3	22.5
PinkBox	Queue3	7.5	3	9	7	12.8
BlueBox	Queue3	7.5	6	10	3	3.4
BlueBox	Queue3	7.5	4	7	3	7.1

If we could represent the model structure in these quasi-table terms, we could use SQL to do complex filtering and prioritizing based on those tables.

First, let's start simple. We'll just search one queue of flow items, Queue4, and we'll only look at the Step value, which, we'll say is stored on the items' "Step" label. We want to find the flow items with a step value greater than 3. The more simplified table would look like:

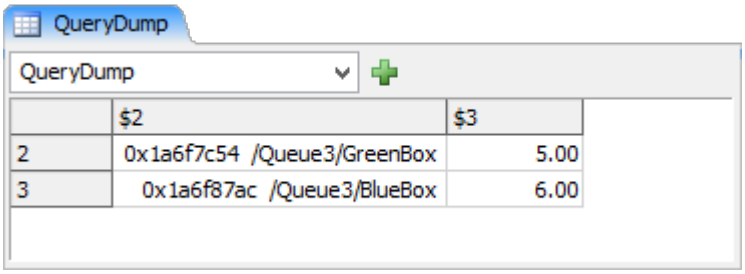
Item	Step

OrangeBox	1
GreenBox	5
BlueBox	6

Here's the command.

```
Table result = Table.query("SELECT $2 AS Item, $3 AS Step \
        FROM $1 Queue \
      WHERE Step > 3",
   /*$1*/node("Queue4", model()),
    /*$2*/$iter(1),
    /*$3*/getlabel($iter(1), "Step"));
```

The results of result.cloneTo() on this would be:



Here we introduce two new concepts: 1. object references as tables, and 2. individual column values being defined by the $ syntax, instead of just tables.

Object References as Tables

The query table is defined with:

```
FROM $1 Queue
```

And we bind the $1 reference with:

```
node("Queue4", model())
```

Here instead of referencing an actual table, we reference an object in the model. In doing this, we're associating each row of our "virtual table" with a sub-node of the object we reference, or in other words, a flow item in the queue. So the first row in the table is associated with the first flow item inside the queue, the second row with the second flow item, and so on.

$'s as Table Values and $iter()

The select statement looks like this:

```
SELECT $2 AS Item, $3 AS Step
```

And we bind $2 with:

```
$iter(1)
```

And $3 with:

getlabel($iter(1), "Step")

Again here's the "virtual table" we are searching, based on each flow item in Queue4 and its Step label.

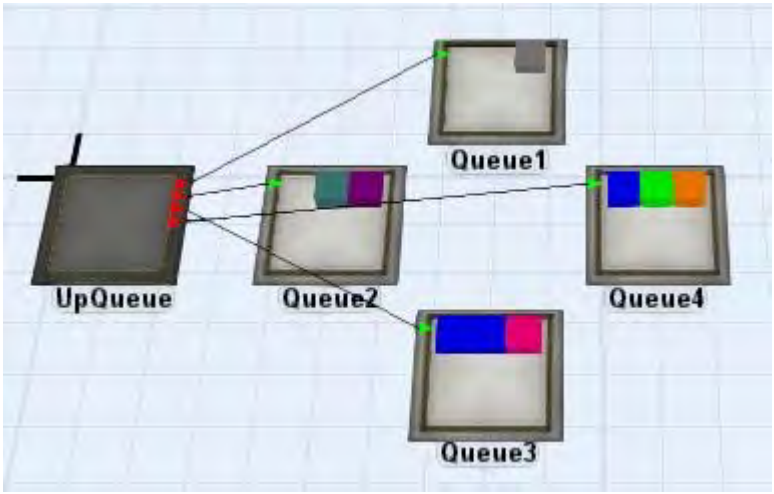
Item	Step
OrangeBox	1
GreenBox	5
BlueBox	6

We use the $iter() command to determine the values of the table cells by traversing the content of Queue4. $iter() returns the iteration on a given table. $iter(1) is the iteration on $1. Since $1 is Queue4, $iter(1) is going to be the iteration on Queue4 associated with a given row in the table, or in other words, one of the flow item sub-nodes of Queue4.

When the evaluator needs to get the value of a certain row in the table, it sets $iter(1) to the flow item sub-node of Queue4 associated with that table row, and then re-evaluates the expression. So when it's on the GreenBox row in the table (row 2) and needs to get the Step value for that row, it sets $iter(1) to rank(Queue4, 2), and evaluates $3. $3 then essentially returns getlabel(rank(Queue4, 2), "Step"). That value is then used as the value of the table cell.

Creating a Table for All Flow Items

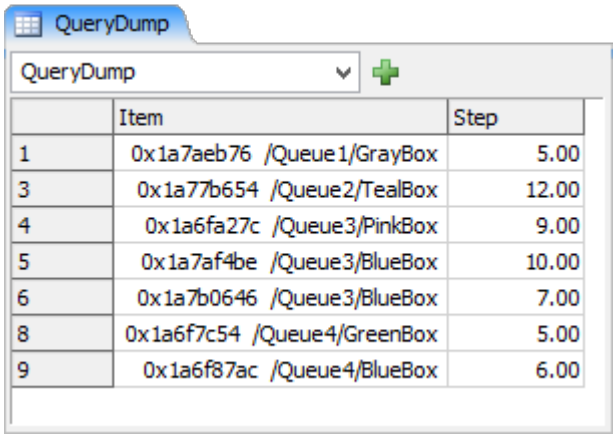
Now that we've done a query on a single queue, let's extend the query to search all queues. Let's assume the queues are all connected to the output ports of an upstream queue.



Now let's write the query.

```
Table result = Table.query("SELECT $3 AS Item, $4 AS Step \
            FROM $1x$2 AS Items \
        WHERE Step > 3",
    /*$1*/nrop(node("UpQueue", model())),
    /*$2*/outobject(node("UpQueue", model()), $iter(1)),
    /*$3*/$iter(2),
    /*$4*/getlabel($iter(2), "Step"));
```

The results of result.cloneTo() on this query are as follows:


QueryDump

QueryDump

	Item	Step
1	0x1a7aeb76 /Queue1/GrayBox	5.00
3	0x1a77b654 /Queue2/TealBox	12.00
4	0x1a6fa27c /Queue3/PinkBox	9.00
5	0x1a7af4be /Queue3/BlueBox	10.00
6	0x1a7b0646 /Queue3/BlueBox	7.00
8	0x1a6f7c54 /Queue4/GreenBox	5.00
9	0x1a6f87ac /Queue4/BlueBox	6.00

Note that we've shifted our column references down. The Item reference is now $3 and the Step label reference is $4. Also notice that we've "flattened" a two-dimensional model structure into a single virtual table by using the special table identifier $1x$2. The first dimension is the output ports of UpQueue, and the second dimension is the sub-node tree of each of the Queues connected to the output ports of UpQueue.

Numbers as Tables

For $1 we return nrop(node("UpQueue", model()). Unlike previously where we assigned $1 to a Queue object itself, now we define $1 as a straight number, namely the number of output ports of UpQueue. Subsequently, the iteration on $1 ($iter(1)) will also be a number.

"Flattening" Model Structures

When the SQL parser sees the special $1x$2 table reference, it will "build" the table by evaluating $1 and then iterating on $1, evaluating $2 for each iteration. In this example, it evaluates $1 which returns 3, the number of output ports of UpQueue. Then it iterates from 1 to 3, evaluating $2 each time. On the first iteration ($iter(1) == 1), $2 returns Queue1: outobject(UpQueue, 1). The evaluator then determines it's an object reference and adds 1 row (the number of flow items in Queue1) to the table. Then it continues to the next iteration on $1 ($iter(1) == 2). $2 returns Queue2: outobject(UpQueue, 2), and the evaluator adds 2 rows (the number of flow items in Queue2) to the table, and so on until it's built the entire table. Note that behind the scenes it's not really "building" a table. It's only storing off the total table size and what rows in the table are associated with what $ iterations. Once it's built the table, the evaluator then goes through the table and evaluates the query for each row in the table, calling $3 and $4 to figure out the associated values of each iteration.

In defining a table you can use any number of model structure "dimensions" to define a table, meaning you could have a table that is $1x$2x$3x$4, i.e. 4 model-structure dimensions, or more. You can do inner joins on these tables just like you would do with standard tables. Thus, by using these advanced querying techniques you can quickly reduce complex filtering and prioritizing problems into relatively simple SQL queries.

SQL Language Support

Enumerating all of the rules and nuances of SQL is outside the scope of this document. You can get very helpful tutorials on SQL from www.w3schools.com/sql/. FlexSim's SQL parser supports a relatively small subset of SQL language constructs, but hopefully it is enough for most of our users' needs. We list here the full set of SQL constructs supported by FlexSim's SQL parser:

- SELECT
- FROM o INNER JOIN
    - Inner joins via comma-separated tables o ON
    - In the Table.query() command you can use $ syntax to define tables, and then pass the table references as additional parameters
    - If you define the table name directly, then Table.query() will first look for a global table of that name, and second look for a global list of that name. If the target list is a partitioned list, then you should encode the sql table name as ListName.$1, and then pass the partition ID as the first additional parameter to Table.query(). Or, if the partition ID is a string, you can encode the string directly into the table name. For example, the table named ListName.Partition1 means it will use the global list named "ListName", and the partition with ID "Partition1".
    - Nested queries
- WHERE
    - IN
        - ✦ IN clause with comma-separated values
        - ✦ IN clause with the value being a FlexSim Array
    - BETWEEN o AND/OR o LIKE
- ORDER BY
    - with multiple comma-delimited criteria o ASC, DESC options
- LIMIT
- SQL Aliases using AS
- SQL Aggregation Functions o SUM() o AVG() o COUNT() o MIN() o MAX() o STD() o VAR() o ARRAY_AGG()
- GROUP BY
- HAVING
- Other SQL Functions o RAND()
- FlexScript Commands: you can use any FlexScript command within an SQL expression.
- ROW_NUMBER: FlexSim's sql parser includes a hard-coded column named ROW_NUMBER, which will give the row number associated with a given table being queried.

SQL Query Result Retrieval

See Table.query() for more information on the Table's SQL interface.

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.

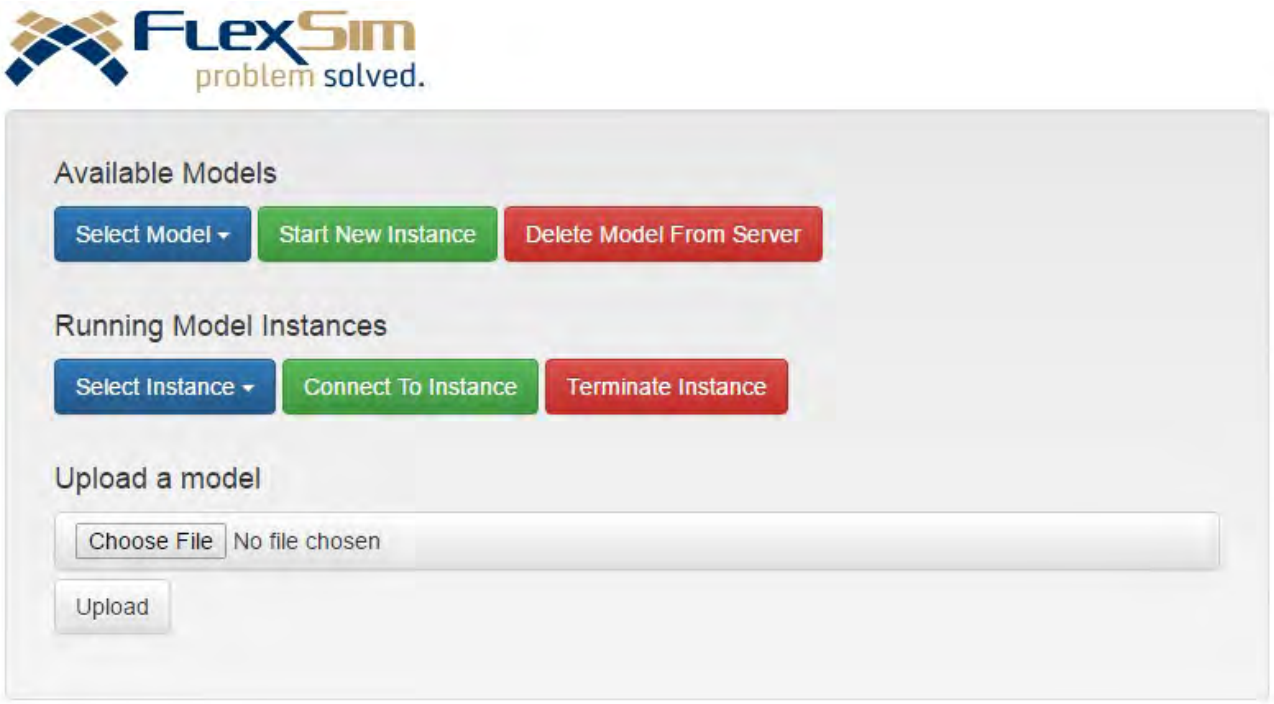
1 - STATE_IDLE
2 - STATE_PROCESSING
3 - STATE_BUSY
4 - STATE_BLOCKED
5 - STATE_GENERATING
6 - STATE_EMPTY
7 - STATE_COLLECTING
8 - STATE_RELEASING
9 - STATE_WAITING_FOR_OPERATOR
10 - STATE_WAITING_FOR_TRANSPORTER
11 - STATE_BREAKDOWN
12 - STATE_SCHEDULED_DOWN
13 - STATE_CONVEYING
14 - STATE_TRAVEL_EMPTY
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

- STATE_SUSPEND
- STATE_AVAILABLE
- STATE_PREPROCESSING
- STATE_POSTPROCESSING
- STATE_INSPECTING
- STATE_OPERATING
- STATE_STANDBY 43 - STATE_PURGING
- STATE_CLEANING
- STATE_ACCELERATING 46 - STATE_MAXSPEED
- STATE_DECELERATING
- STATE_STOPPED
- STATE_WAITING
- STATE_ACCUMULATING

FlexSim's Webserver is a query-driven manager and communication interface for FlexSim. It allows you to run FlexSim models through a web browser like Google Chrome, FireFox, Internet Explorer, etc.

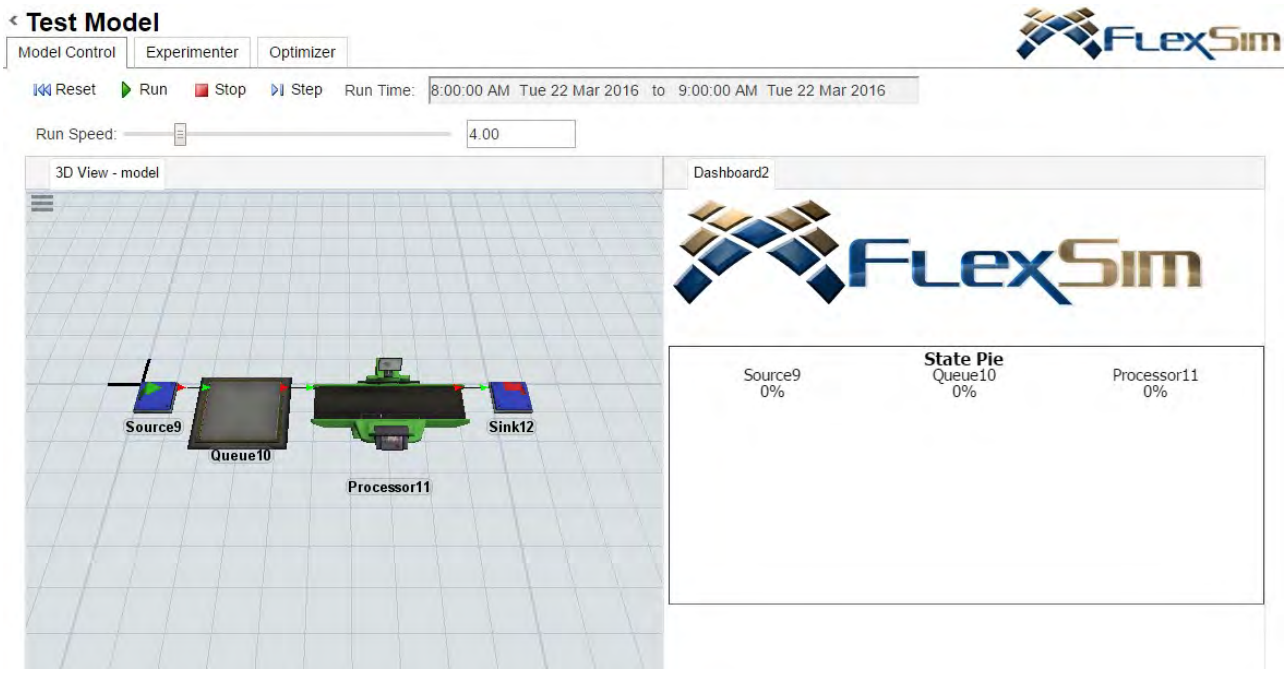
The WebServer now has its own installer which can be found on the FlexSim downloads page.

When you start the FlexSim Server using the desktop shortcut, your computer will begin to host a website that looks like this:



This website can be accessed by typing the address http://127.0.0.1/ into a browser. The Webserver looks for models in FlexSim's projects directory ('Documents/FlexSim 2017 Projects' for example).

To start a new instance of FlexSim, select a model to run and click Start New Instance. The Start New Instance button will gray out until FlexSim has started and the model is open. Once that occurs, the drop down under Running Model Instances will populate with the newly created instance. Select the instance and click Connect To Instance to view the automatically generated web interface for your model. Our Test Model had a 3D view and a Dashboard open side by side:



The interface that is displayed will match the interface of the model. You can rearrange the interface the same way you interact with FlexSim's windows. Click and drag on the window's tab to undock it, or right click and select Float. You can also close views by right clicking the window tab and selecting Close. Once undocked, you will see options for docking that window in various locations in the interface.

Valid Windows The Webserver is not capable of displaying all of FlexSim's different windows. If a window is not supported, it will still be displayed in the windowing interface, but no data will be available.

3D View

You can control your model in the browser using the controls at the top. You can also interact with open 3D views in your model. Left click to pan, right click to rotate, and either right-and-left click or use the mouse wheel to zoom. Additional buttons for panning, rotating, and zooming are generated for tablet devices and smartphones.

Dashboards

Dashboards are also displayed in real time. You can even reposition and resize dashboard statistic widgets. If the model's dashboards contain input widgets like text fields, tables, buttons, etc. you will be able to interact with these elements.

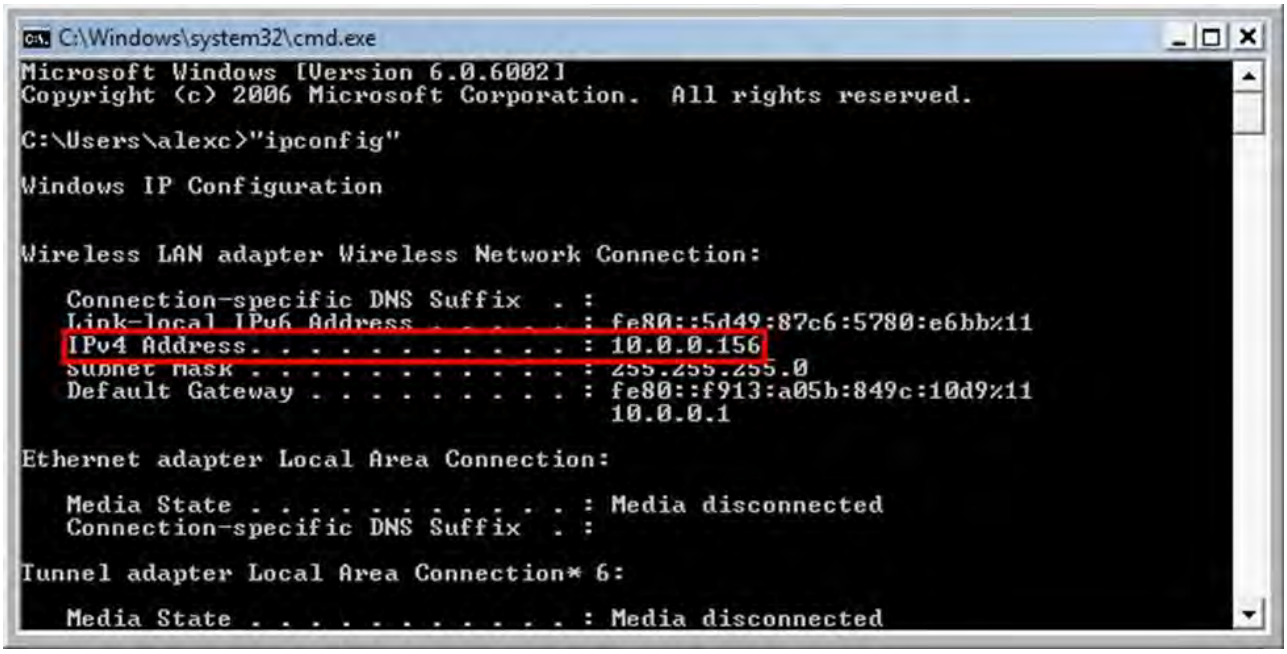
Experimenter / Optimizer

If your model has an Experimenter, a standard interface is generated for running experiments. Use the tabs at the top of the window to move between the Model Control, Experimenter and Optimizer. You can define experiment variables, run times, optimization values and other parameters.

Tablets and Smartphones

To connect other devices such as tablets and smartphones to this server, the address 127.0.0.1 will not work. The other device will need to be connected to the same local area network or wireless local area network as your computer. You will need to find the ip address of your computer. To do this, click the start

button and type *cmd.exe*, which will open a black command prompt. Enter *ipconfig* and press enter. Your IPv4 Address should be entered as the URL in the browser in the device.



Internet Accessibility

To use the server on the internet, your computer needs to be assigned a global IP address. Contact your network administrator.

Developing Custom Web Interfaces

The FlexSim Webserver allows you to start a simple test webserver without installing any webserver software like Apache or IIS. If you want to deploy a webserver using IIS, see Using FlexSim Webserver with IIS. You can create custom webserver applications using queries provided by FlexSim or user defined queries contained in the model. To get exact details on what a query returns, enter the query in the URL bar of the browser to view the reply (for example: 127.0.0.1/webserver.dll?availablemodels).

The following queries are available. You can also enter 127.0.0.1/webserver.dll into the URL bar to display this list.

General

- webserver.dll?availablemodels returns an xml list of the models available to run on the server using the createinstance query.
- webserver.dll?allfiles returns an xml list of all files in the model directory on the server.
- webserver.dll?configuration returns an xml list of all parameters specified in the 'flexsim webserver configuration.txt' file.
- webserver.dll?uploadmodel uploads the model to the server, if this feature has been enabled in the 'flexsim webserver configuration.txt' file. A form such as the form in index.html uploads a model (or another file, such as an excel table) to the model directory.
- webserver.dll?deletemodel=my%20model deletes the model (or another file) from the model directory on the server, if this feature has been enabled in the 'flexsim webserver configuration.txt' file.
- webserver.dll?downloadfile=modeldata%2Ecsv downloads a file from the model directory on the server, if this feature has been enabled in the 'flexsim webserver configuration.txt' file.

Instances

- webserver.dll?instancelist returns an xml list of the models running on the server and their instance numbers. These are the valid possibilities for queryinstance and terminateinstance queries.
- webserver.dll?numinstances returns an xml number representing the number of instances running on the server including from other users. This can be used for load balancing multiple servers.
- webserver.dll?createinstance=my%20model starts an instance of FlexSim running the model specified (here it's 'my model.fsm') and returns a short xml reply containing the instance number.
- webserver.dll?queryinstance=my%model&instancenum=1&... queries an instance of FlexSim. FlexSim decides how to reply to the request. This is the gateway to the main functionality of communicating with a model. See Instance Queries below for more information.
- webserver.dll?terminateinstance=my%model&instancenum=1 terminates the first instance of FlexSim running 'my model.fsm' on the server and returns a short xml reply.

Libraries

- webserver.dll?getlibraries returns a list of the files in the libraries directory on the server.

Modules

- webserver.dll?getmodules returns a list of the files in the modules directory on the server.

Jobs

- webserver.dll?submitjob should be the action of an HTML form like this one containing a job description in JSON format. The reply will be an id for use with getjobresults queries.

```
<form action="http://127.0.0.1/webserver.dll?submitjob" method="POST">
    <input type="hidden" name="job"
value='{"modelname":"my%20model","timeout":3600,"priority":5,"setupcommands":
[{"command":"settable=modelparameters","data":"values=[[\"firstParam\",0],[\"secondParam\",2]
,[\"thirdParam\",\"hello\"]]"},
{"command":"setrunspeed=100000000"},{"command":"setstoptime=86400"},{"command":"tools
nodefunction=setupscript"},{"command":"run"}],
"resultcommands":[{"command":"getnodedata=/Tools/TrackedVariables/WorkInProgress"},{"co
mmand":"getnodedata=/Tools/modeloutput"}]}'>
    <input type="submit">
</form>
```

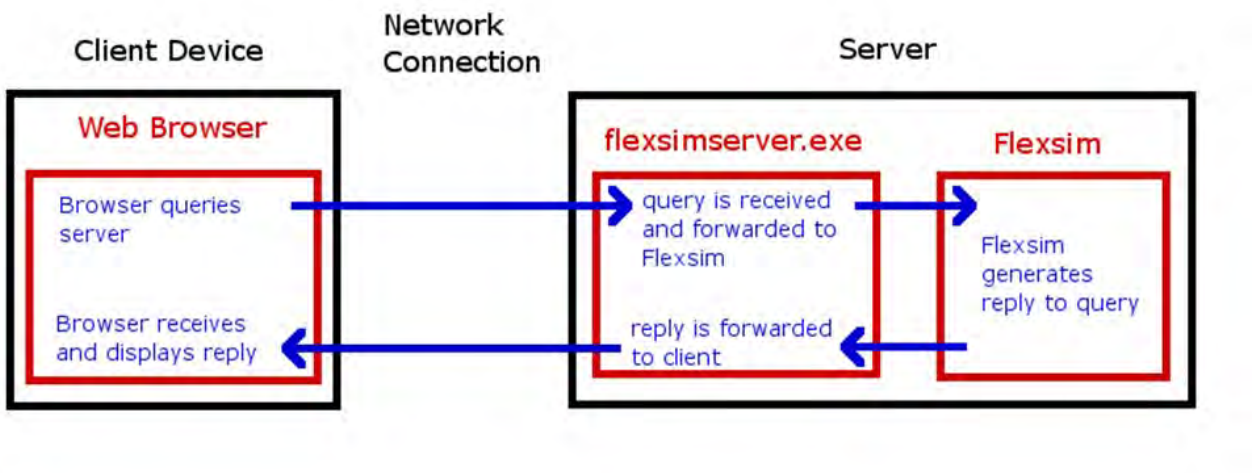
The JSON object must have a "modelname" field, which is the name of the model on the server. It can have a "timeout" field, which is the maximum number of seconds to wait for the model run to complete. It must have a "setupcommands" field, which is an array of command objects. It must have a "resultcommands" field, which is an array of command objects. Command objects must have a "command" field, which is the part of a queryinstance query after webserver.dll?queryinstance=my%20model&instancenum=1&id=1&... Command objects can also have a "data" field, which is the body of the HTTP request, typically containing data from POST requests. Command objects can also have a "verb" field, which is usually "GET" or "POST". The job will be done as soon as the server is able to make more instances of FlexSim. A createinstance query will be created, followed by a queryinstance=my%20model&instancenum=1&defaultpage query, whose replies are ignored. The replies to each of the setup commands are then stored, followed by polling the model with

queries like queryinstance=my%20model&instancenum=1&id=1&getrunstate. When the runstate is 0 (when the model is finished running), the replies to the result commands will be stored. A complete summary of the run in JSON format can be sent as a reply to a getjobresults query.

- webserver.dll?getjobquery=1 gets a JSON string such as "queryinstance=my%20model&instancenum=1" that can be used to interact with models run by the job manager. If the job has no running instance (such as if it is waiting or complete), an empty string will be returned because the instance cannot communicate right now. When communicating with an instance of FlexSim started by the job manager, the instance may be shut down at any time. This is why instances started by the job manager are not listed in an instancelist query.
- webserver.dll?getjobresults=1 returns a JSON summary of a job submitted with a submitjob query.
- webserver.dll?getjobstatus=1 returns a JSON summary of the job status of a job submitted with a submitjob query. This is a shorter version of a getjobresults query that does not include the full description of the results.
- webserver.dll?getjobqueuelength returns a JSON number representing the number of jobs in the job queue.
- webserver.dll?canceljob=1 cancels a job submitted with a submitjob query if it is incomplete or waiting.

Instance Queries

Once an instance has been created, the webserver can communicate with FlexSim through query handlers. These allow you to both send and receive data from FlexSim. Query requests to the server are handled as seen in this picture:



A list of default queries is available at MAIN:/project/exec/globals/serverinterface/queryhandlers. Using these as templates you can create your own custom query handlers by adding subnodes to the MODEL:/Tools/serverinterface/queryhandlers node. These queryhandler nodes should be toggled flexscript. The name of the node corresponds to a value passed in by the query. For example, *webserver.dll?queryinstance=Test%20Model&instancenum=1&reset* executes the queryhandler located at MAIN:/project/exec/globals/serverinterface/queryhandlers/reset. Parameters may be added to the query to pass additional information. For example, *webserver.dll?queryinstance=Test%20Model&instancenum=1&customquery=1&count=5*.

Custom functionality and custom query types for querying the model can also be written for a model by making a flexscript node at MODEL:/Tools/serverinterface/sendreply. This node must call webcommand("httpsendreply", replynode); once to send a reply to the query. This will send a reply in different

forms, depending on the type of data in the node given as a parameter. If the node has string data containing a syntactically correct HTTP reply with HTTP headers as defined by http://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html, then it will be sent as-is, giving developers full access to send any type of HTTP reply desired. Refer to the code in MAIN:/project/exec/globals/serverinterface/sendreply for an example, which can be copied and edited as needed. Remember that the defaultPage and getRunState queries are used by the job interface.

Default Page

The default webpage for a model is generated by using a template found in the tree at MAIN:/project/exec/globals/serverinterface/pagetemplates/default. An html page can be made for a specific model by putting the html into a node created at MODEL:/Tools/serverinterface/defaultpagetemplate. This page can include the php-like <?flexscript ?> tags to generate dynamic material. This will override the default page of the model.

Using FlexSim Webserver with IIS

If IIS is configured to use the iisnode module, it can also manage instances of FlexSim and interact with FlexSim's webserver interface. This requires IIS 7.x with IIS Management Tools, ASP.NET, and the URL Rewrite module for IIS.

All interaction should be the same except for three major differences: visibility, threading and authentication.

1. IIS is run in the background, so FlexSim instances will not be visible on the server. This makes it harder to debug any problems.
2. IIS use multiple threads to respond to requests in parallel, so the order of replying to requests could be different.
3. Access to models in the Model Directory can be restricted when Windows Authentication is enabled (both in IIS and in 'flexsim webserver configuration.txt'). Directories named after usergroups defined by the domain controller are restricted. Users are only given access to restricted directories named after usergroups which they belong to.

WebSockets
The FlexSim Server uses websockets for faster video streaming. WebSocket functionality requires IIS 8.x

Allocating Windows Memory

There is a Windows Registry setting that limits the amount of memory (desktop heap size) that Windows will allocate for non-interactive processes. The registry path is:

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session Manager\SubSystems\Windows

This key has a value that is a long string with more key/value pairs:

%SystemRoot%\system32\csrss.exe

ObjectDirectory=\Windows

SharedSection=1024,20480,512

Windows=On

SubSystemType=Windows

ServerDll=basesrv,1

ServerDll=winsrv:UserServerDllInitialization,3

ServerDll=sxssrv,4

ProfileControl=Off

MaxRequestThreads=16

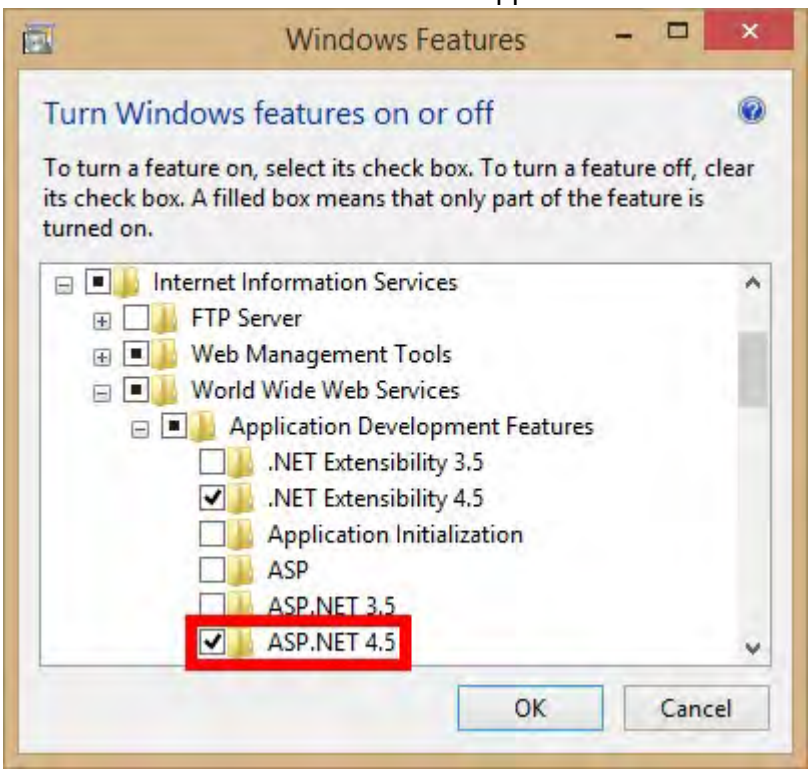
The SharedSection key defines, in its 3rd parameter, the amount of memory that can be allocated by IIS. By default, this is 512. If you need to open multiple instances of FlexSim through the webserver, you may need to increase this number to beyond the total amount of memory that all your instances will use simultaneously. This number will vary based upon the memory requirements of the models that are being run.

Be sure to reboot your machine after making any changes.

Configuring IIS

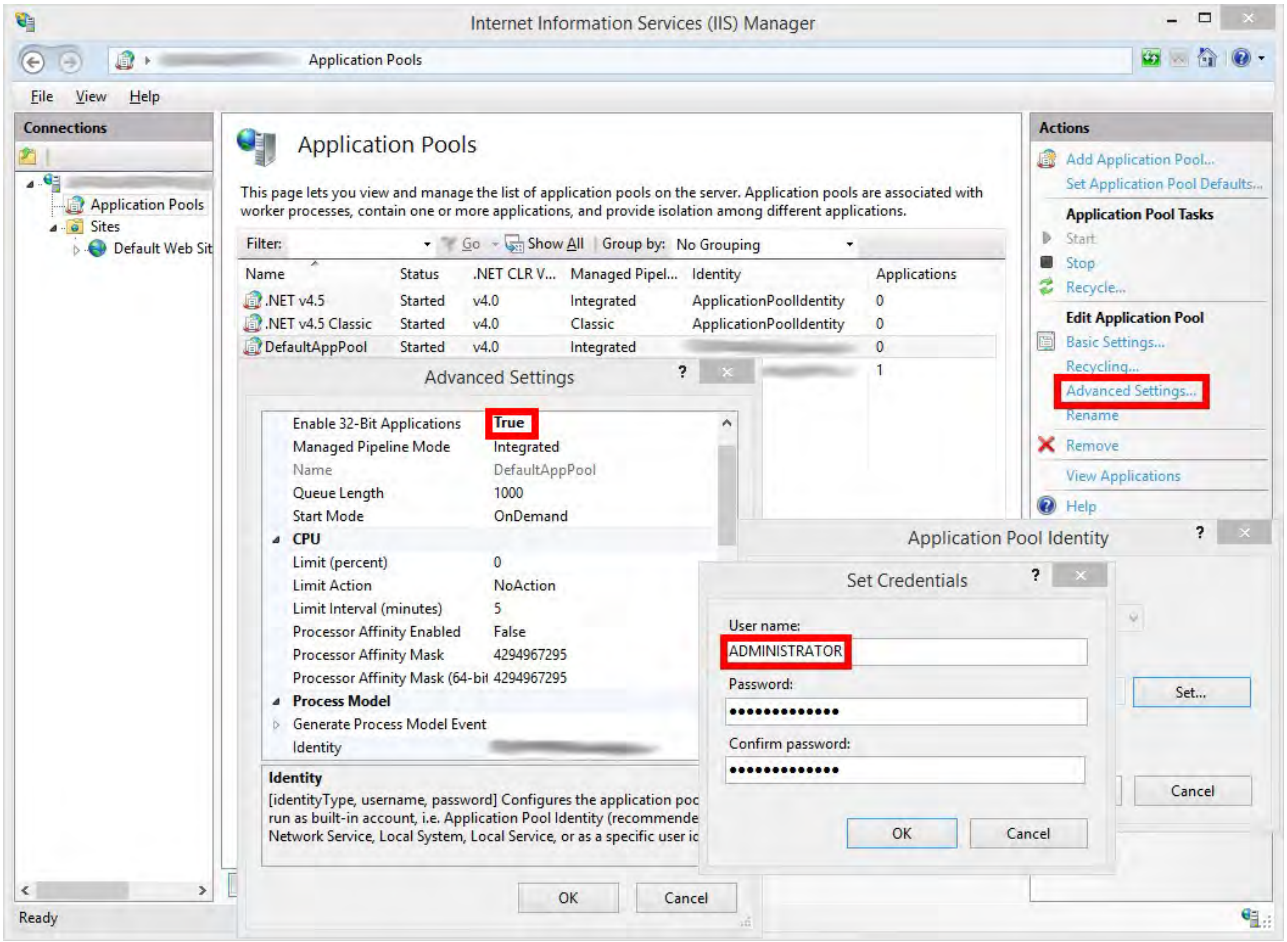
IIS requires some additional configuration to work with FlexSim. The steps are outlined as follows:
Windows Server operating systems and 64 bit operating systems may have some additional complications.

- Here is how to configure IIS from a fresh install of Windows Server 2012:
  - Open the Server Manager. o In Server Manager, select Dashboard, and click Add roles and features.
  - In the Add Roles and Features Wizard, on the Before you begin page, click Next.
  - On the Select installation type page, select Role-based or feature-based installation, and click Next.
  - On the Select destination server page, select Select a server from the server pool, select your server, and click Next.
  - On the Select server roles page, select Web Server (IIS), and click Next. o On the Select features page, click Next.
  - On the Web Server Role (IIS) page, click Next.
  - On the Select role services page, note the preselected role services that are installed by default, expand the Application Development node (under World Wide Web Services), and then select ASP.NET 4.5.
  - On the Summary of Features to Install page, confirm your selections, and then click Install. o In the Add features that are required for ASP.NET 4.5? box, click Add Features. Then click Next.
  - On the Confirm installation selections page, click Install.
- Here is how to configure IIS from a fresh install of Windows 8:
  - In the Turn Windows features on or off dialog box, click Internet Information Services to install the default features.
  - Expand the Application Development Features node (under World Wide Web Services) and click ASP.NET 4.5 to add the features that support ASP.NET.

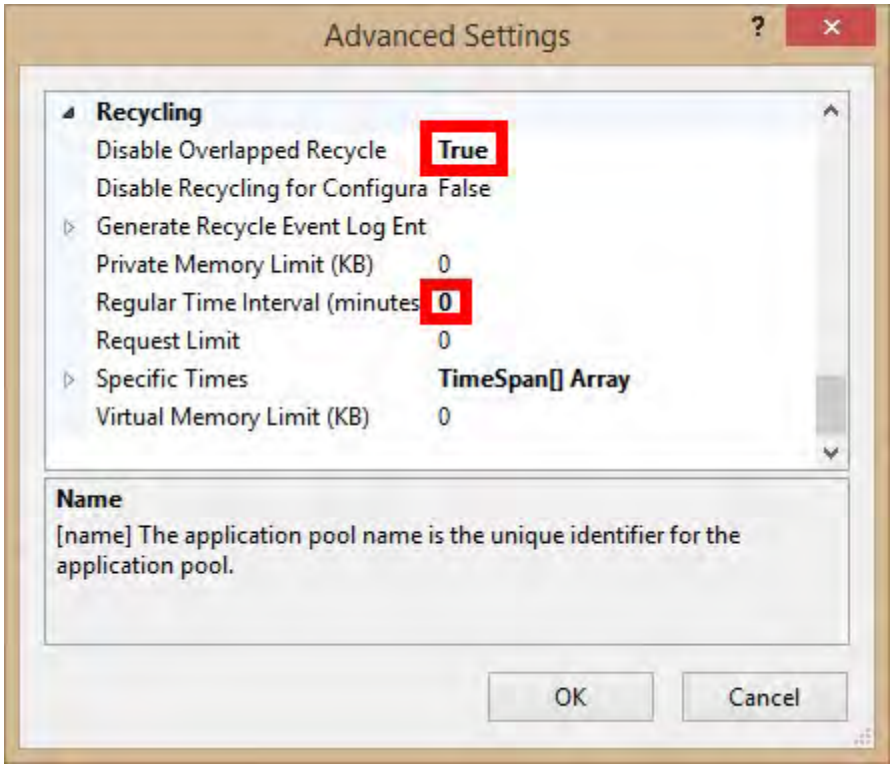


- Download and install URL Rewrite.
- Download and install iisnode.

- Make sure that FlexSim is installed in a directory that is accessible by IIS and your models are in a directory that is accessible by IIS (for example, not in C:\Users\...).
- Make sure that the FlexSim Server is installed. Download it here: FlexSim downloads page
- Replace any %DOCUMENTS% or %PROGRAMFILES% macros in the directories in 'flexsim webserver configuration.txt' with the full path of the directory.
- Copy the contents of the FlexSim Server directory into C:\inetpub\wwwroot.
- Copy the entire flexsimweb folder (not just its contents) from the FlexSim program directory to C:\inetpub\wwwroot.
- Go to the Advanced Settings of the DefaultAppPool, in the IIS Manager. o If you are running a 32-bit version of FlexSim, enable 32-bit Applications (otherwise leave this option set to FALSE).
    - Change the Application Pool Identity to ADMINISTRATOR. There will be many permissions issues avoided by doing this.



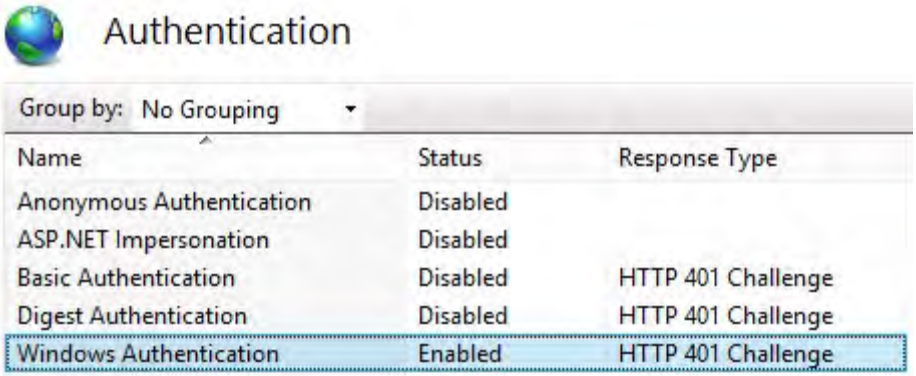
    - To prevent more than one instance of the server running at once, enable Disable Overlapped Recycle. o Set Regular Time Interval (minutes) to 0. This prevents periodic recycling of the server process.



- Go to URL Rewrite under the Default Web Site, still in the IIS Manager.
  - Click Add Rule(s)... and select Blank rule. o In the Pattern field enter "webserver.dll". o In the Rewrite URL field enter "webserver/index.js".

Windows Authentication in IIS

Under Sites/Default Web Site, double click on Authentication. Enable Windows Authentication.



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++.

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 3D 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 pick lists.
- Copying variables from a highlighted object to any set of objects.
- Editing C++ code directly.

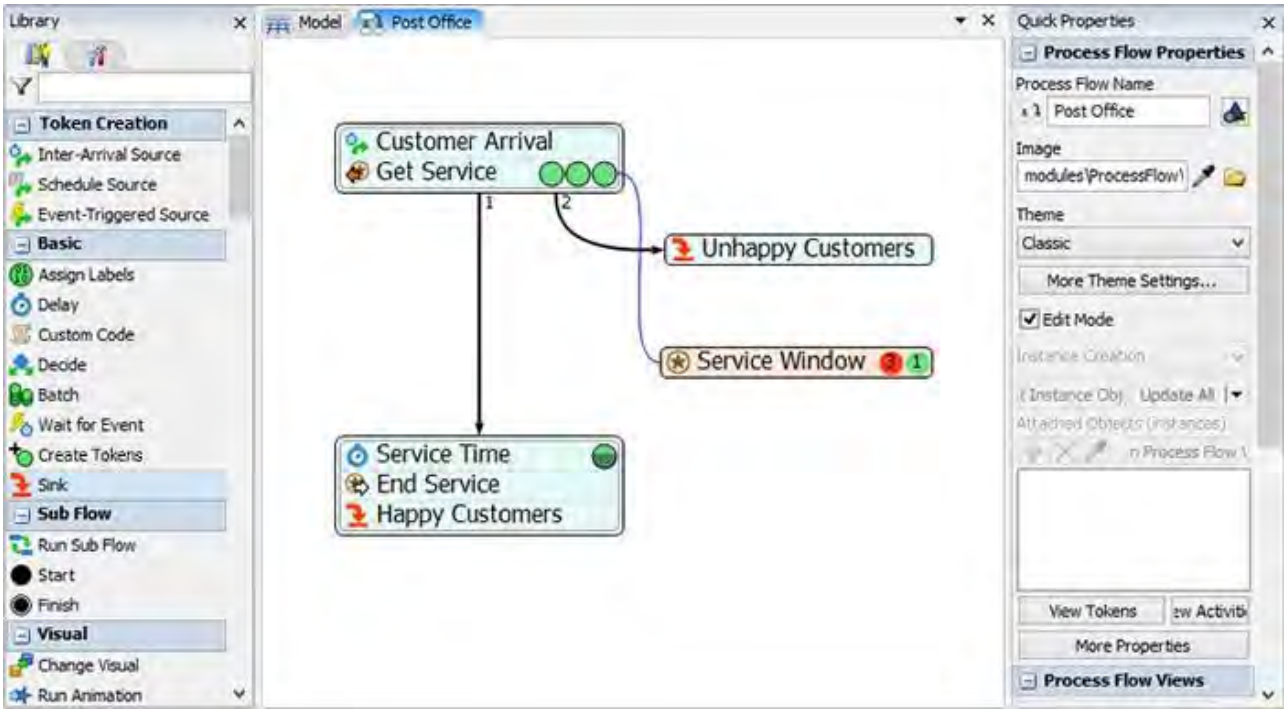
Whenever you perform one of the following operations, you may need to reset the model, but you should NOT need to compile the model.

- 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.

Why Use Process Flow?

One of the major selling points of FlexSim is the ability to create custom objects that are capable of simulating the underlying logic of complex business processes. Up until now, most of this customization required you to write code in FlexScript (FlexSim's internal scripting language). While many advanced users appreciate being able to tinker directly with the code, some users might prefer a simple, user-friendly interface that writes the code behind the scenes for them. If you're one of those users, the new Process Flow module is designed specifically with you in mind.

The following image shows an example of a process flow:



NOTE: The process flow shown in this image is the same one that you will build in the Process Flow Basics Tutorial.

Scalable, User-Friendly Flowchart Tools

The Process Flow module has many tools and features that can easily scale for use in projects of any size and complexity. A simulation project generally involves large, complex problems that can be broken down into sets of smaller problems. The Process Flow module has the ability to break up large simulation models into small, manageable pieces that can be understood on your terms.

Many FlexSim users typically begin the simulation process by creating a simple flowchart of their business systems. With the Process Flow module, you can design this kind of flowchart in an early model-building stage and then possibly add more complex logic to that same flowchart to control the logic for your simulation model in the later model-building stages. You can easily integrate flowcharts with existing FlexSim 3D models or even let the flowcharts be stand-alone, abstract simulations that can run independently of your 3D model.

Cleaner, Deeper Simulations with No Code

Using the Process Flow module, you can build the entire logic of your simulation system using a visuallyoriented, user-friendly interface. Instead of writing code, you'll select an appropriate pre-programmed activity from the process flow library, then drag and drop it directly into your flowchart. Each activity can perform simple or complex logical operations, including some new operations that were not previously available in FlexSim. These activities can be combined together in a variety of ways to simulate practically any kind of system.

With the Process Flow module, you can begin building the logic of your simulation earlier in a project life cycle from a perspective that makes sense to you. This means you will be able to build your ideas into the simulation more quickly. It also means that you can adapt your simulation more easily and efficiently later

in the simulation life cycle as your knowledge changes or you get new information about your business system.

Greater Flexibility

The Process Flow module can flexibly adapt to many different kinds of simulation projects and different approaches to the task of simulation. Rather than asking you to learn one particular approach toward simulation and then adapt your simulation model to this style, the Process Flow module will adapt to you and your approach to simulation. Because there are many possible ways to solve the same problem in the Process Flow module, you can build the logic of your simulation in a way that makes sense to you. This flexibility makes it possible for you to address the challenges that are unique to your simulation project.

Engaging, Intuitive Visuals

The Process Flow module is ideal for presenting simulation models to decision makers. Stakeholders are more likely to catch your vision when you can present the deeper logic of your business system using easy-to-understand, visually-engaging flowcharts.

You can easily change the visual settings and overall organization of your process flowcharts. The Process Flow module has many options that make it possible for you to customize fonts, colors, and shapes. You can also move and connect activities so that they are spatially organized and grouped in whatever way best communicates their meaning.

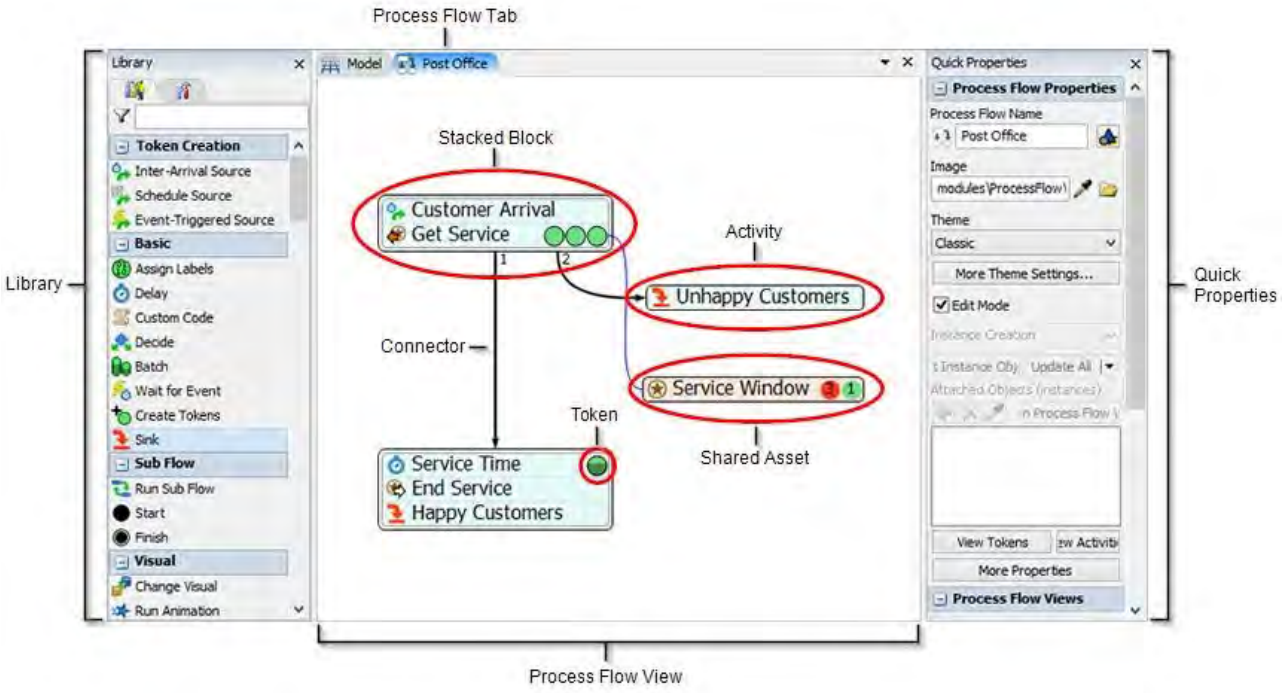
Besides simply making compelling presentations, these engaging visuals also makes it easy to debug your simulation since process flow helps you to see the deeper logic of your business system right before your eyes.

In short, the new Process Flow module will make model-building much easier and more user-friendly from the beginning of the process to the end.

Overview of Process Flow

This topic will provide a high-level overview of the most important elements of the Process Flow module and will explain some of its key terms and concepts. It will also link to other related topics for more in-depth information.

The most important elements of the Process Flow module are labeled in the following image:



NOTE: The process flow shown in this image is the same one that you will build in the Process Flow Basics Tutorial.

Each element will be explained in more details in the following sections:

- Process Flow View
- Process Flow Pane
- Token
- Library
- Activity
- Shared Asset
- Connector
- Block (Stacked Block)
- Quick Properties
- Display Objects

Process Flow View

The Process Flow view is the main workspace where you will build your process flow. See Navigating in Process Flow for more information about creating a new process flow and navigating in the process flow view.

Process Flow Pane

When you first create a new process flow, the Process Flow pane will appear next to the Model pane. Every time you create a new process flow, it will show up as its own tabbed pane. See Navigating in Process Flow for more information about rearranging the display of process flow tabs.

Token

The most basic component of a process flow is the *token*, as shown in the following image:

Tokens are objects that flow through the activities in a process flow during a simulation run. They function very similarly to flowitems in a standard 3D model. Like flowitems, tokens move from one activity to the next, much like a flowitem moving from a source to a queue to a processor and so forth. But unlike flowitems, tokens do not necessarily have to represent a physical object moving through a business system. Tokens can be more abstract---meaning they can represent anything you want them to represent. For example, they can represent a customer's order being placed, a call to a call center, a logical grouping of pallets that are eventually loaded onto a truck for delivery, etc. Tokens will often be logically linked to physical objects in a standard system (and to flowitems in a standard 3D model), but they do not have to be. It is essentially up to you to define the nature of the links between tokens and the physical elements in your simulation model.

At its most basic level, a token is just a bundle of data that is moving through a process flow. Each token can contain the following basic information:

- ID - When a token is created using one of the token creation activities (one of the Source activities), it is automatically assigned a unique ID number that can be used as a reference point. The ID number will be unique among all process flow objects and instances within a FlexSim model.
- Name - Naming your tokens can help you to better identify their purpose, function, or location in the process flow. Tokens are not required to have a name but can be given one when they are created by one of the process flow Source activities. You could also possibly change the name at a later stage in the process flow using a Custom Code activity with the Setname picklist option or the `setname()` command.
- Labels - Labels are essential to building a complex and dynamic process flow. Labels store custom information on a token that can be used to affect what happens to that token as it advances through a process flow. Various activities might assign data to a token's labels as it moves through the activities in a flowchart. See Process Flow Labels for more information about using labels generally.

Tokens will only show up in a process flow when a simulation model is running. See Running a Process Flow Simulation for more information about how tokens will function and appear visually during a simulation run.

Library

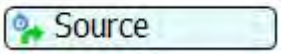
When a process flow view is active, the Library will change to display all the activities and shared assets that can be added to a process flow. You can drag activities directly into the process flow to add them.

See Adding and Connecting Activities for more information about additional ways to add activities to a process flow.

See Overview of Process Flow Objects for a high-level summary of each of the process flow activities available in the Library.

Activity

An *activity* is a logical operation or step in a process flow. As such, activities are the basic building blocks of any process flow. The following image shows an example of a typical activity (a Source):



Activities are dragged from the Library into a process flow and linked together with connectors. As a token enters an activity, it performs the logic associated with that type of activity. This can include assigning labels, moving flowitems, delaying a token, etc.

In some activities, like the Assign Labels activity, the token will start and end the activity with no time passing in the simulation clock. While running a model the token will not be visible in that activity. However, using the Step button on the simulation control bar, you can see the token move through each activity one step at a time. In other activities like the Acquire Resource, the token may be delayed and wait in that activity until certain criteria are met.

See Overview of Process Flow Objects for a high-level summary of each of the process flow activities.

Shared Asset

A shared asset is a finite resource that tokens may claim or release at certain points in the process flow. Although they are very similar to activities, there are a few subtle differences. By default, they have a different color than other activities, as shown in the following image:

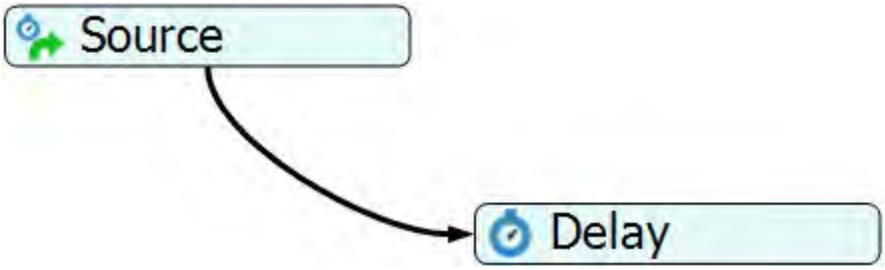
Shared assets can impose constraints on the tokens by making the token wait if the requested asset is unavailable. A real-life example of this would be a certain tool that is shared by three different work stations. If one station needs the tool while it is already claimed by another station, that station must wait until the tool becomes available. In that same vein, if a token needs an asset in order to move on to the next activity, the token will wait at its current activity until the asset is available. Currently, there are three types of shared assets:

- Resource - Represents a limited supply of some resource that can be acquired and released. It can be used to simulate a supply of goods, services, time, materials, employees, etc.
- List - Represents a list of tokens, flowitems, task executers, task sequences, numbers, strings, etc. Process flows can use a list that is local to the process flow itself or could be tied to a Global List in the simulation model.

- Zone - Can collect statistical information not available for standard activities. It can also restrict access to a section of the process flow based on certain statistics or other criteria.

Connector

A *connector* is a connection between two activities, as shown in the following image:



During a simulation run, tokens will use connectors to go from one activity to the next downstream activity. Some activities allow more than one outgoing connector, but many do not.

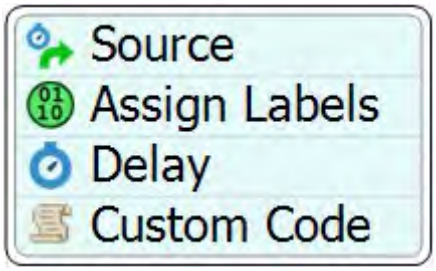
See Adding and Connecting Activities for more information.

Connectors are not quite the same as port connections in the 3D model

Although connectors function very similarly to port connections in the 3D model, they do not function exactly the same. For instance, there is no First Available option in process flow. Tokens can always be sent through any connector at any time as activities cannot be blocked.

Block (Stacked Block)

Occasionally this manual will use the term *stacked block* or *block* to describe a set of activities that have been snapped together into one movable block, as shown in the following image:



Stacked blocks let you easily link a set of activities into a single sequence of steps, making it easy to move and edit them as if they were one block. When activities are linked together in a stacked block, they are automatically connected as though each activity were connected with connectors.

When a token enters a stacked block, it will always start at the topmost activity and then work its way down to the bottom activity. Any connections into a stacked block will be connected to the topmost activity, regardless of where the connection points to in the stacked block. Any outgoing connections from the stacked block will be connected from the bottom activity.

Quick Properties

You can use the settings that appear in the Quick Properties pane to change the basic functionality of the process flows and activities. You can also use the Quick Properties pane to view important information about a token while it is selected. See the following related topics:

- Editing Activity Properties
- Common Process Flow Object Properties
- General Process Flow Properties
- Changing Process Flow Visuals
- Using Process Flow Themes
- Troubleshooting Process Flows

Display Objects

Although they are not pictured in the image above, display objects are used as presentation tools to convey information about the process flow without actually defining any simulation logic. Display objects may group blocks together and convey flow charting concepts, as in a Flow Chart object, or they may show text, images or arrows. See Flow Chart Objects for more information.

Navigating in Process Flow

This topic will discuss how to navigate in this Process Flow view. NOTE: The directions in this topic assume that the process flow view is the active view. To make the process flow view active, simply click on the view.

Creating a New Process Flow

There are two ways to create a new process flow:

- Using the main toolbar
- Using the Toolbox

Both methods will be explained in this section.

Using the Main Toolbar

To create a new process flow using the main toolbar:

1. Create a new model, then click the ProcessFlow button on the main toolbar to open a menu.
2. Select the type of process flow you would like to add. There are four available process flow types:

    o General o Fixed
    Resource o Task
    Executer o
    Subflow

See Types of Process Flows for more information about these four options.

Using the Toolbox

Like other FlexSim Tools, a new Process Flow can be added from the Toolbox (which shares the left pane with the Library). To add a new process flow:

1. Open the Toolbox by clicking the Tools button on the main toolbar.
2. Click the Add button  button at the top of the Toolbox to open the menu of available tools.
3. Point to Process Flow, then select the type of process flow you would like to add. There are four available process flow types:

    o General o Fixed
    Resource o Task
    Executer o
    Subflow

See Types of Process Flows for more information about these four options.

Navigating in a Process Flow

Navigating in a process flow is very similar to navigating inside model view. You can zoom in and out and also pan left and right or up and down as needed. However, since process flows only display in 2D, you won't be able to rotate the view.

There are two ways to pan left, right, up, or down:

- Using the mouse, click and drag on any white space in the process flow view.
- Use the scroll wheel to pan up and down. Hold the Shift key while scrolling to pan left and right.

There are two ways to zoom in and out:

- Using the mouse, click and drag using the left and right mouse buttons.
- Hold the Ctrl key and use the scroll wheel.

Rearranging the Process Flow Tabs

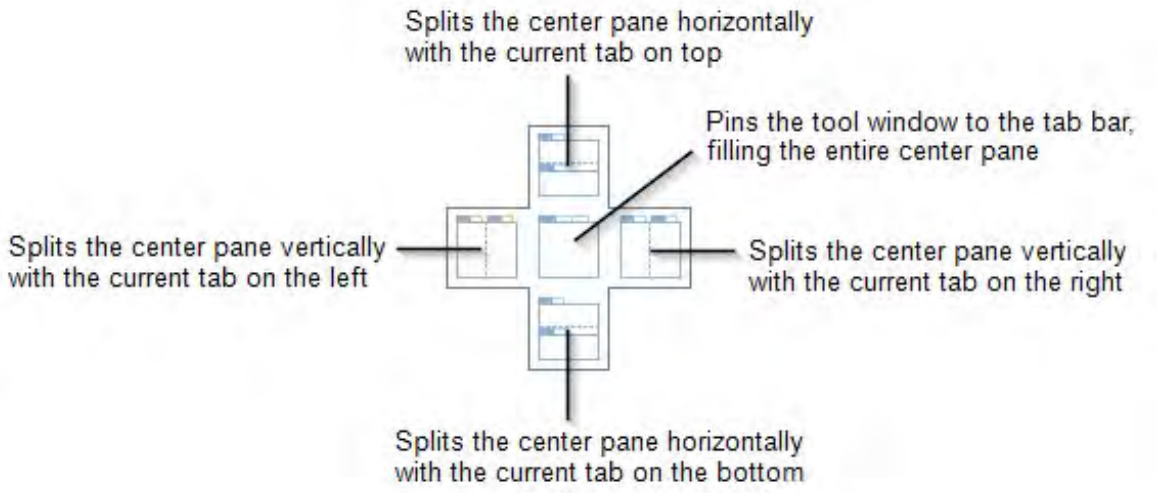
If needed, you can change the way that the process flow tab displays in the center pane. For example, you could possibly split the center pane with the model view and the process flow view, or tab them on top of each other so that each takes up the entire pane. To re-arrange the way windows or tabs are displayed or docked:

1. Click on the tab for the process flow you want to move and begin dragging it to another part of the screen.

2. Before you release the mouse button, you'll notice that the window becomes undocked and floats as an independent window outside of the main FlexSim window. If you release your mouse button at this point, the window will continue to be undocked as a separate window.

3. Also as you click on the tab or the window's title bar and drag it around the screen, notice that a Guide Diamond appears near the center of the main FlexSim window, as shown in the following image:

4. While still pressing the mouse button, drag the mouse to the Guide Diamond. Notice that the Guide Diamond has five different areas. Each area corresponds to a different way of displaying or docking the window, as explained in the following image:



5. As your mouse moves over different areas in the Guide Diamond, notice that the area turns gray to indicate it is selected. Release your mouse when your mouse is positioned over the desired screen position.

Process flows can be linked to simulation model or they can run independently as a self-contained, abstract simulation model. You run a process flow the same way you would run any simulation model in FlexSim, which means using the simulation control bar:

As a reminder, you should always press the Reset button before starting a new simulation run.

What Happens During a Process Flow Simulation?

During a simulation run, process flows follow the same basic procedure:

1. Most process flows begin with a Source activity that creates one or more tokens. Tokens typically look like green dots on an activity:  Depending on which type of Source activity you use in your process flow, Sources can create new tokens at regular intervals or on a schedule as soon as a simulation run begins. They can also listen for events in a simulation model and create tokens whenever a specific event occurs. NOTE: If the process flow is a sub flow, it will usually begin with a Start activity instead of a Source. See Sub Process Flows for more information.

2. The Source will then release the token to the next downstream activity. When that activity finishes performing its action on the token, it will release it to the next downstream activity and so forth.

3. If a Delay activity needs to hold a token for a certain amount of time, the token will change color as time elapses, which is similar visually to a fill line increasing over time:  When the time has elapsed, the Delay activity will release the token to the next downstream activity.

4. Some process flow activities will hold tokens while waiting for a certain amount of time to elapse or while waiting for another event in the process flow or simulation model. If an activity is holding many tokens at a time, the tokens might appear to stack visually. If a large quantity of tokens builds up on a process flow activity, they will be replaced by a single token that merely indicates the number of tokens inside of it, as shown in the following animated image:

5. Process Flows typically end with a Sink activity (or a Finish activity if it is a sub flow). However, it is possible for activities to loop indefinitely without crashing the software. You can create a loop by

connecting the last activity in a process flow to a previous activity in the process flow if needed. Tutorial 2 - Linking Process Flows to 3D Models has an example of looping in a process flow.

Debugging and Troubleshooting

See Troubleshooting Process Flows for more information about how you can debug a process flow during a simulation run.

When you build a simulation model using the process flow module, you can build four different types of process flows. This topic will provide a high-level overview of these four types so that you can decide which type of process flow to create when you're making a new process flow.

The main purpose of a process flow is to control the basic logic of your simulation model. In this respect, you could think of process flows as the "brains" of a simulation model. Therefore, when you decide which type of process flow type to create, it is like answering the question: whose brain is it that you are implementing?

- General - Choose this type of process flow if you want to create a centralized brain for the entire model. General types of process flows are ideal for creating process flow logic that will need to be globally accessible to many objects in the simulation model. A general process flow is a good place to put any labels, variables, or logic that needs to be used by multiple objects in the model.

- Fixed Resource - Choose this type of process flow if you want to create the brain of one or more fixed resource objects. You'd use this type of process flow if you want to create logic for custom FR objects or if you simply want a fixed resource to have additional functionality beyond the standard FlexSim logic for that object.

- Task Executer - Choose this type of process flow if you want to create the brain of one or more task executer objects. Task Executer process flows are identical to Fixed Resource process flows, except that they are meant for task executer objects.

- Sub-Flow - Choose this type of process flow if you want to create a sub flow, which is like a small part of a brain that will get used by many different objects. A sub flow is a separate process flow that begins running when it is triggered by another activity or event in a different process flow. Sub flows are chunks of self-contained logic that will get executed when they are triggered by certain events in the simulation model or general process flow. If you are familiar with programming terms, you could think of a sub flow as a *function* or a *subroutine*.) See Sub Process Flows for more information.

If you are unsure what type of process flow to start with, choose a General Process Flow. Be aware that the type of process flow you choose could affect the picklist options that are available to link that process flow to objects in the 3D simulation model. See Linking Process Flows to 3D Models for more information.

Changing a Process Flow Type

After you create a process flow, there is no method for changing a process flow's type (such as in its general properties). But if you begin working on a process flow and realize you should have chosen a different process flow type, simply create a new process flow and copy the activities and stacked blocks from the old process flow into the new one. The activities will retain all the same settings and properties when they are copied. See Copying and Pasting Activities or Stacked Blocks for more information.

Instances

In using different types of process flows, you will come across the idea of instances. Instances are specific copies of a process flow's 'brain'. For example, as mentioned before, a fixed resource process flow would implement the brain of a fixed resource. Once you've implemented this brain, you can attach that process

flow to one or more fixed resources in the model. Each fixed resource will then become a separate instance of the process flow.

If you are using a general process flow, there will only be one instance, namely the process flow itself. If you use a fixed resource or task executer process flow, there will be one instance for each object that is attached to the process flow. If you use a sub flow, instances are created dynamically during the model run, based on rules that you define on the process flow.

See Process Flow Instances for more information on instances.

This topic will provide a high-level overview of each task involved in building a process flow. It will explain which tools are involved in each task and will provide links to the relevant sections of the User Manual for each phase and tool.

Be aware that you do not necessarily need to build your model in the exact order listed here.

1. Plan Your Process Flow

Before you begin designing your process flow, you might want to understand the basics of how process flow works. You should start by reading these topics:

- Overview of Process Flow - Provides an overview of the process flow user interface and important terms and concepts.
- Navigating in the Process Flow Window - Explains how to create a new process flow and how to arrange process flow tabs.
- Types of Process Flows - Discusses the four different types of process flows and their purposes.
- Overview of Process Flow Objects - Provides a high-level overview of all the different activities that are available in process flow

2. Add, Connect, and Edit Activities

Activities are the basic building blocks of every process flow. To learn more about how to use activities, consider reading these topics:

- Adding and Connecting Activities - Explains key concepts and several different methods for adding and connecting activities in process flow.
- Moving and Deleting Activities - Discusses how to move and delete activities.
- Editing Activity Properties - Describes two different methods for editing an activity's properties. It also explains the Universal Edit feature, a new type of property that is available for some properties in the Process Flow module.
- Common Process Flow Object Properties - This reference page explains some of the common properties that are found in many process flow activities.

3. Link the Process Flow to a 3D Model

Process flows can run independently from a simulation model if needed. But process flows are probably most useful when they are linked to the 3D simulation model. Process flows can help you build complex logic into your model, replacing the need for complicated code. Process flows will also make it easier for you to troubleshoot your model logic if needed. With that in mind, these topics cover how to integrate process flows with a standard simulation model:

- Linking Process Flows to 3D Models - Gives a high-level overview of two different ways to link process flows to 3D simulation models.
- Key Concepts About Event-Listening - Events are the most common way to link to a process flow from a simulation model. This topic explains the key concepts that are related to event-listening and gives different examples of ways to use event-listening in a simulation model.
- Event Types - This topic discusses the two different types of events and their related properties.

4. Edit General Process Flow Properties

You can also change some of the process flow's general properties and visual settings:

- General Process Flow Properties - Explains the general properties that are available for each process flow.
- Changing Process Flow Visuals - Describes how to change the way process flows appear visually.
- Using Process Flow Themes - Discusses how to change the overall color scheme in process flow.

5. Running and Troubleshooting a Process Flow

Lastly, you'll want to run your process flow:

- Running a Process Flow Simulation - Running a process flow is slightly different from running a normal simulation model. This topic will discuss what you should see when a process flow is running.
- Troubleshooting Process Flows - One of the main advantages of process flow is how easy it is to troubleshoot it. This topic explains the best strategies for troubleshooting a process flow.

The Process Flow module has a unique set of objects in the Library. When you have the process flow view open and active in FlexSim, the Library will change to display these objects.

The following tables will provide a high-level overview of these objects. Each object has a link to its object reference page for more information. The categories and objects are presented in the same order as they appear in the Library.

Token Creation

Similar to sources in the 3D simulation model, these activities create the tokens that will flow through the process flow. Most process flows will begin with some kind of Token Creation activity.

Name and Icon	Description
Inter-Arrival Source 	Creates new tokens according to a specific interval of time. You can use a fixed number to set an exact interval of time between token creations or you can use a statistical distribution to randomly calculate the time between arrivals. See the Inter-Arrival Source activity for more information.
Schedule Source 	Creates new tokens at a rate specified in its Arrivals table. This table defines the time (in model units) that tokens should be created, the name that will be assigned to the new tokens, and the number of tokens to create. See the Schedule Source activity for more information.
Event-Triggered Source 	Creates tokens in response to an event during a simulation run. This source will listen for that event to occur in the simulation model. When that event occurs, it will create a token. See the Event-Triggered Source activity for more information.

Basic

The Basic activities have many general uses in a wide variety of process flows.

Name and Icon	Description

Assign Labels 	Creates or modifies labels on various objects. Labels can be used to store important data about various objects. See the Assign Labels activity for more information.
Delay 	Holds the token for a certain length of time. You can use a fixed time or you can create the delay time dynamically using a label value on a token, a statistical distribution, etc. See the Delay activity for more information.
Custom Code 	Can create custom behavior in the process flow module. You can create custom codes by selecting pre-defined picklist options or by writing your own code in FlexScript. When a token enters the Custom Code activity, it will evaluate the user-defined code and then immediately be released to the next activity without any time passing. See the Custom Code activity for more information.
Decide 	Send a token to one of two or more possible activities based on conditions that you define. In other words, the Decide activity can determine the next activity that a token should be diverted to. See the Decide activity for more information.
Batch 	Collects incoming tokens and sorts them into groups of tokens (batches). When a batch is ready, the Batch activity will release it to a downstream token. Using the Batch activity for simple grouping and releasing is relatively straightforward. However, the Batch activity has some additional features that allow you to collect, sort, and release batches in fairly complex ways. See the Batch activity for more information.

Wait for Event 	Holds the token until a certain event is triggered. This activity will listen for that event to occur in the simulation model. When that event occurs, it will release the token. See the Wait for Event activity for more information.

Create Tokens 	Creates one or more new tokens and automatically sends them to a different activity or sub flow. See the Create Tokens activity for more information.
Sink 	Destroys tokens, removing all data stored on those tokens. Typically put at the end of a process flow. See the Sink activity for more information.

Sub Flow

Sub flow activities are needed in order to create and run sub flows. See Sub Process Flows for additional information.

Name and Icon	Description
Run Sub Flow 	Initiates a sub process flow. When a token enters this activity, it will create one or more child tokens and send them to the Start activity in a sub flow. See the Run Sub Flow activity for more information.
Start 	The Start activity is usually the first activity in a sub flow. However, the Start activity only marks the beginning of a sub flow. It does not perform any other logic. After a token enters the Start activity, it will immediately move to the next connected activity in the sub flow. See the Start activity for more information.

Finish 	The Finish activity marks the end of a sub flow. The Finish activity esentially acts the same as a Sink in that it destroys tokens that enter it. See the Finish activity for more information.

Visual

These activities can be used to change the visuals of an object or run an animation in the 3D model.

Name and Icon	Description
Change Visual 	Changes the appearance of an object in the 3D model. Generally this is used for changing a flowitem's visuals, but it may also be used to adjust any 3D object's visual properties. See the Change Visual activity for more information.
Run Animation 	Triggers an animation on a 3D object in the simulation model. You can use any of the animations that come pre-programmed with the standard FlexSim objects or you can create your own custom animations. See the Run Animation activity for more information.

Objects

These activities can create, move, or destroy objects such as flowitems, fixed resources, task executers, etc. in the 3D simulation model.

Name and Icon	Description

Create Object 	Creates one or more objects in the 3D simulation model. This activity can also create TaskExecuters and TaskExecuter Flowitems and connect them to a travel network. See the Create Object activity for more information.
Move Object	Moves an object or multiple objects to another place in a simulation model. See the Move Object activity for more information.
Destroy Object 	Removes one or more objects in the 3D simulation model. See the Destroy Object activity for more information.

Task Sequences

These activities can be used to build task sequences that can be assigned to task executers (such as operators) in the simulation model.

Name and Icon	Description
Travel 	Makes a task executer travel to a specific object (such as a fixed resource) in the 3D simulation model. See the Travel activity for more information.
Load 	Makes a task executer load an object in the 3D simulation model. For example, you can make a task executer pick up an object, possibly to transport it to another destination. See the Load activity for more information.

Unload 	Makes a task executer unload an object in the 3D simulation model. For example, you can make a task executer drop off an object at a particular destination. See the Unload activity for more information.
Delay 	Makes a task executer delay for a specific period of time in the 3D simulation model. For example, a delay could represent a task that takes a specific amount of time to complete such as cleaning an object, assembling a product, etc. See the Delay (Task Sequence) activity for more information.
Travel to Loc 	Makes a task executer travel to specific X, Y, and Z coordinates in the 3D simulation model. See the Travel to Location activity for more information.
Custom Task 	If you need to give a task executer a task that isn't currently available in the process flow library, you can use the Custom Task activity to build your own custom task. See the Custom Task activity for more information.
Create Task Sequence 	Creates an empty task sequence. Tasks can be assigned to this task sequence later in the process flow. See the Create Task Sequences activity for more information.
Dispatch Task Sequence 	You could use a Dispatch Task Sequence if you would prefer instead to build a series of task sequences first and dispatch them later. See the Dispatch Task Sequence activity for more information.

Lists

These activities can be used to create and manage lists in the simulation model.

Name and Icon	Description
List 	Represents a list of tokens, flowitems, task executers, numbers, strings etc. Process flows can use a list that is local to the process flow itself or could be tied to a Global List in the simulation model. See the List shared asset for more information.
Push to List 	Can be used to add tokens, flowitems, task executers, numbers, strings, etc. to a list. See the Push to List activity for more information.
Pull from List 	Can be used to retrieve tokens, flowitems, task executers, numbers, strings, etc. from a list. See the Pull from List activity for more information.

Resources

These activities can be used to create and manage resources in the simulation model.

Name and Icon	Description
Resource 	Represents a limited supply of some resource that can be acquired and released. It can be used to simulate a supply of goods, services, time, materials, employees, etc. See the Resource shared asset for more information.

Acquire Resource 	Used to acquire a resource at some point during a process flow. When a resource has been acquired, it reduces the supply of that resource by a specified amount. See the Acquire Resource activity for more information.
Release Resource 	Used to release or return a resource at some point during a process flow. When a resource has been released, it increased the supply of that resource by a specified amount. See the Release Resource activity for more information.

Zones

These activities can be used to create and manage zones in the process flow.

Name and Icon	Description
Zone 	Can collect statistical information not available for standard activities. It can also restrict access to a section of the process flow based on certain statistics or other criteria. See the Zone shared asset for more information.
Enter Zone 	Used to define the point in the process flow where the token will either enter or request access to enter the zone. See the Enter Zone activity for more information.
Exit Zone 	Used to define the point in the process flow where the token will exit the zone. See the Exit Zone activity for more information.

Coordination

These activities can be used to coordinate multiple tokens in process flow.

Name and Icon	Description
Split 	Splits the token into multiple tokens and sends one out each outgoing connector. See the Split activity for more information.
Join 	Tokens wait at the Join activity until one token arrives from each incoming connector. The token from the first connector is then released out the outgoing connector and the other tokens destroyed. See the Join activity for more information.
Synchronize 	Tokens wait at the Synchronize activity until one token arrives from each incoming connector. The tokens then leave through the outgoing connector corresponding to the incoming connecter they came in through. See the Synchronize activity for more information.

Preemption

These activities can be used to manage preemption systems in process flow.

Name and Icon	Description
Save Token Context 	Before preempting, this activity saves the token's current context in the process flow so that it can possibly return to this context after it finishes the preempting activities. See the Save Token Context activity for more information.

Release Token 	Aborts the token's current activity and releases it to a new activity, either to do something else, or to just wait. See the Release Token activity for more information.
Restore Token Context 	After completing the operation that required the preemption, this activity restores the token back to its saved context. See the Restore Token Context activity for more information.

Display

These objects have no actual function in the process flow and are available for visual display only.

Name and Icon	Description
Text 	The Text object can act like a custom text box that can be placed anywhere inside a process flow. See the Text object for more information.
Arrow 	The Arrow object is an arrow you can put anywhere in your process flow. See the Arrow object for more information.
Image 	Use the Image object for a number of different purposes such as adding a custom image somewhere in your process flow, adding a background image to a process flow, or adding a background image to Flow Chart object. See the Image object for more information.

Flow Chart

There are many different Flow Chart objects available in the Process Flow Library. You can use these shapes to build a classic flowchart in your process flow. You could also possibly use them as visual containers that help organize and visualize the activities in your process flow. See the Flow Chart objects for more information.

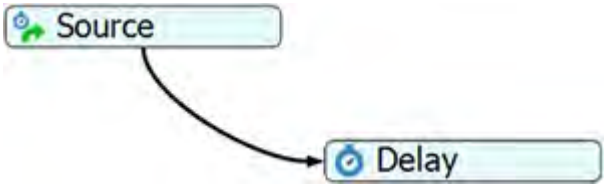
Adding and Connecting Activities

Activities are the basic building blocks of the Process Flow module. This topic will explain some important concepts about adding and connecting activities to your process flows. This topic will also explain different methods for adding and connecting activities. Be aware that Tutorial 1 - Process Flow Basics will provide a hands-on, step-by-step approach to all the different methods for adding and connecting activities. This topic will cover the following topics:

- About Connectors and Stacked Blocks
- Number of Outgoing Connectors
- Adding and Connecting Activities from the Process Flow Library
- Adding and Connecting Activities Using the Quick Library
- Inserting an Activity in the Middle of a Stacked Block
- Copying and Pasting Activities or Stacked Blocks
- Adding Outgoing Connectors
- Removing Connectors and Separating Stacked Blocks

About Connectors and Stacked Blocks

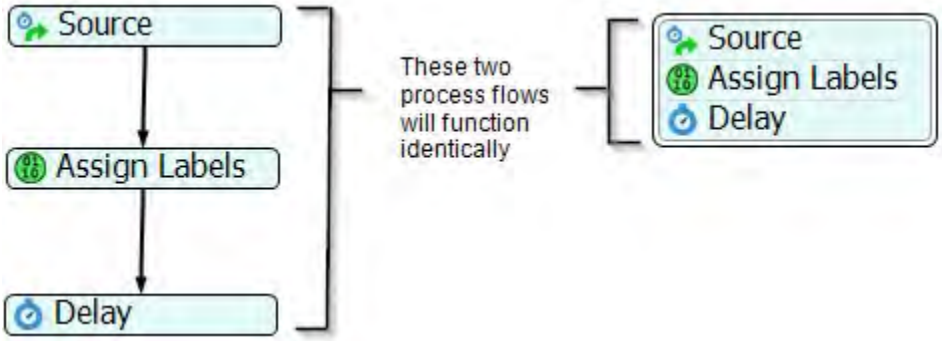
One of the important things to understand are the similarities and differences between connectors and stacked blocks. A *connector* is a connection between two activities, as shown in the following image:



During a simulation run, tokens will use connectors to go from one activity to the next downstream activity. Some activities allow more than one outgoing connector, but many do not.

A *stacked block* is a set of activities that have been snapped together into one movable block, as shown in the following image:

Stacked blocks let you easily link a set of activities into a single sequence of steps, making it easy to move and edit them as if they were one block. When activities are linked together in a stacked block, they are automatically connected as though each activity were connected with connectors. In other words, these two process flows will function identically:



You can create stacked blocks through the object library as explained in Adding and Connecting Activities from the Process Flow Library or you can drag an activity above or below another activity to snap them together.

Number of Outgoing Connectors

Most activities can only have one outgoing connection because they usually implement a very simple piece of logic on a token. In that case, the activity can simply send the token to the next downstream activity when that logic is complete. However, a few activities can have more than one outgoing connection. For example, a Decide activity can have more than one outgoing connection because tokens might need to be sent to a different downstream activity based on certain changing conditions. To check whether an activity allows more than one connection, read the reference page for the specific activity you're interested in. You will need to understand how connectors work for activities that can have more than one outgoing connector. The following list explains a few important concepts to keep in mind:

- Each outgoing connection has a number (also sometimes called a *rank*) based on the order in which you created them. When you create an outgoing connection from one activity to another activity, this connection will automatically be assigned a number (rank). For example, the first outgoing connection you create will be assigned a rank of 1, the second will have a rank of 2, and so forth. You may have noticed that this is similar to how port connections work between objects in the 3D simulation model.

- The token will be sent to the next activity based on the number assigned to the outgoing connector. The underlying logic of the activity depends on the numbers (ranks) of the outgoing connectors. For example, if you wanted to send 75% of tokens to the activity connected to the first outgoing connector, you would send them to connector 1. If you wanted to send 25% to the activity connected to the second outgoing connector, you would send them to connector 2.

- You can re-rank outgoing connectors or assign names to connectors if needed. You can edit the activity's properties to change the ranks of the outgoing connectors or assign a name to a connector. You can also assign a name to a connector for easier reference if needed. Activities that allow for more than one outgoing connector will usually have a group of properties labeled *Connectors Out* that can be used to manage the outgoing connectors. NOTE: If you need to refer to a connector's name in a command of some sort, make sure you put it in quotation marks (" ") so that the FlexSim system can recognize it as a string.

Connectors and Coordination Activities

Coordination activities have unique connector behavior and rules that don't apply to most activities. See Connectors and Coordination Activities for more information.

Adding and Connecting Activities from the Process Flow Library

There are several ways to add activities to a process flow. One way is to add an activity from the Process Flow Library:

1. Make sure that the left pane is open to the Library tab (not the Toolbox tab).
2. Click somewhere inside a process flow to make the process flow view active. The Library will change to show the process flow activities (instead of the standard FlexSim objects).
3. Drag an activity from the Library into the process flow.

If you drop an activity from the library onto another activity within the process flow view, the activities will be joined together by adding the new activity to the end of the block. Dropping an activity inbetween two activities in a block will insert the dropped activity in between the two activities.

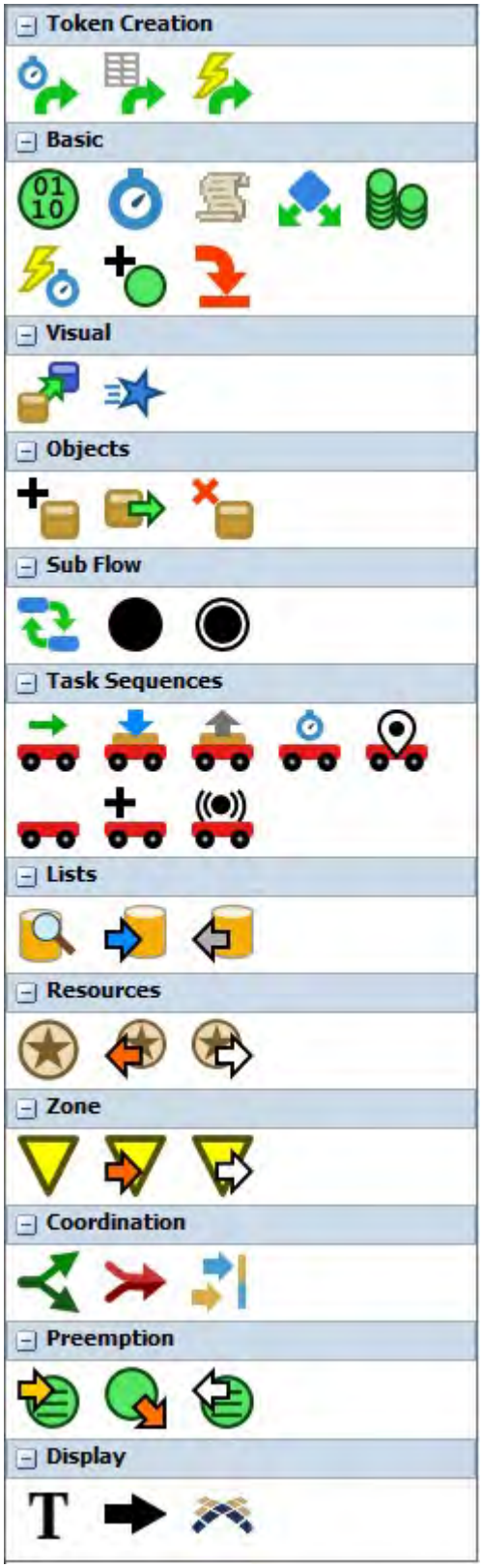
You can also add activities from the library and immediately connect it to a stacked block:

1. Drag an activity from the Library into the process flow without releasing the mouse button.
2. While holding down the mouse button, drag the activity on top of another activity or stacked block and release the mouse. The activity will be appended to the end of the stacked block.

See Removing Connectors and Separating Stacked Blocks for information about separating stacked blocks if needed.

Adding and Connecting Activities Using the Quick Library

You can also add and connect activities using the Quick Library, which is a condensed menu of the process flow activities, as shown in the following image:



If you want to add an activity to a blank area using the Quick Library inside a process flow:

1. Double-click in the process flow to open the Quick Library.
2. Click the icon for the activity you want to add.

If you want to add an activity to the end of a stacked block using the Quick Library:

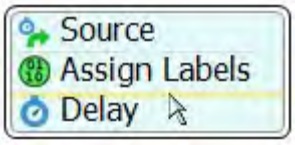
1. Point your mouse on the bottom edge of an activity or stacked block. The mouse icon will change to a chain link. 
2. Double-click the edge to open the Quick Library again.
3. Click the icon for the activity you want to add. The activity will be appended to the end of the stacked block.

See Removing Connectors and Separating Stacked Blocks for information about separating stacked blocks if needed.

Inserting an Activity in the Middle of a Stacked Block

If you want to add an activity inside a stacked block:

1. Point your mouse to the line between two activities until the line turns yellow, as shown in the following image:



2. Double-click the line to open the Quick Library again.
3. Click the icon for the activity you want to add. The activity will be inserted in the middle of the stacked block.

Copying and Pasting Activities or Stacked Blocks

You can easily copy an activity or stacked block. When you copy and paste an activity or stacked block, it will automatically retain all the same settings from the original activity or stacked block. You can also easily copy activities or stacked blocks from one process flow to another in the same simulation model. To copy an activity or stacked block:

1. Click the activity or stacked block to select it.
2. Use Ctrl+C to copy the activity.
3. Use Ctrl+V to paste the copied activity.

Adding Outgoing Connectors

If you want to create an outgoing connector from one activity to another:

1. Point mouse on the bottom edge of an activity or block. The mouse icon will change to a chain link.
2. Click the edge of the activity or block and, while holding down the mouse, drag the mouse toward the activity you want to connect. You'll notice a connector coming from the edge of the block to the mouse pointer.
3. Release the mouse when it is connected to the activity. You can change the curvature or position of the connector by clicking on it and manipulating its curve handles if needed.

You can also add a new activity to the end of a connector using the Quick Library:

1. Point your mouse on the bottom edge of an activity or block. The mouse icon will change to a chain link.
2. Click the edge of the block and, while holding down the mouse, drag it a little bit toward the bottom of the screen. You'll notice a connector coming from the edge of the block to the mouse pointer.
3. When you release the mouse, the Quick Library will appear.
4. Click the icon for the activity you want to add. The activity will be appended to the end of the stacked block, as shown in the following animated image:

Removing Connectors and Separating Stacked Blocks

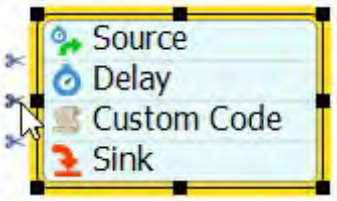
To remove a connector:

1. Click the connector. It will turn yellow to indicate it is highlighted.

2. Press the Delete key. The connector will disappear.

To separate activities in a stacked block:

1. Click anywhere inside the stacked block to select it. Its borders will turn yellow to indicate it is selected. You will also notice that some scissor icons will appear to the left in between the activities, as shown in the following image:



2. Click the scissors icon between the two activities you want to separate. The stacked block will separate into two different blocks.

Moving, Resizing, and Deleting Activities

In order to move, resize, and delete activities or stacked blocks, you must first select the activities or stacked blocks. Selecting activities is similar to selecting files on a Windows folder or desktop:

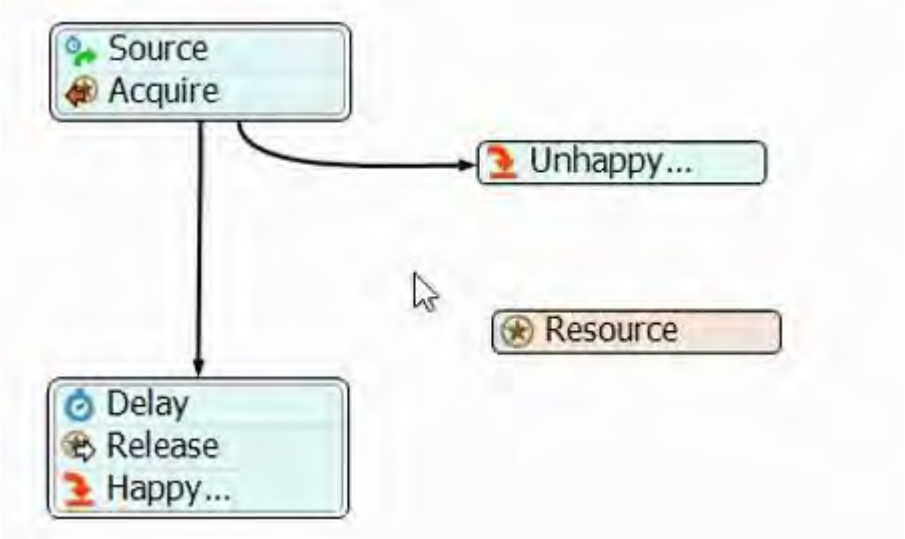
- Simply click on an activity or stacked block to select a single activity or stacked block.
- Use Ctrl + click to select more than one activity.
- Shift + click then drag the mouse to create a selection box. Any activities or stacked blocks inside the box will be selected.

Moving Activities

To move an activity, click on any activity and drag it to a new position in the process flow. Be aware that when you select a stacked block, it will move the entire stacked block. If you drag one activity or one stacked block, the activity/block will automatically align to other activities in the visible view. To disable aligning, hold down the Alt key while dragging.

Resizing Activites and Display Objects

When you click on an activity or stacked block, its borders will turn yellow (to indicate it is highlighted) and you will see its sizers (which look like small black boxes all around the activity or stacked blocks), as shown in the following image:



Clicking and dragging on any of the sizers will allow you to change the size of the object. Hold down the Ctrl key if you want to maintain the activity's aspect ratio while resizing. Activities will automatically align to other activities in the view. To disable aligning, hold down the ALT key while resizing.

Deleting Activities

Select the desired activities. Press the Delete key to remove them from the process flow.

Properly Removing Activities

Do not remove activities through the tree. They can only be removed using the method described in this section.

Removing Connectors and Separating Stacked Blocks

See Adding and Removing Connectors and Separating Stacked Blocks for information about separating stacked blocks if needed.

Editing Activity Properties

Activity properties can be edited two different ways in process flow:

1. Directly on the activity in the process flow
2. In the Quick Properties pane

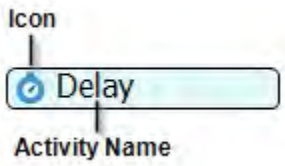
This topic will discuss how to edit an activity's properties using those two methods. It will also discuss how to edit properties that have the Universal Edit feature.

Information About Specific Properties
Each process flow activity has a reference page that describes its properties in greater detail. See Overview of Process Flow Objects for links to each activity's reference page.

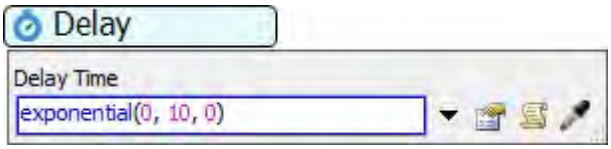
Editing Properties Directly in a Process Flow

To edit an activity's properties directly in a process flow:

1. Click the activity's icon, as labeled in the example in the following image:



2. Clicking the icon will open the activity's Quick Properties, as shown in the following example:



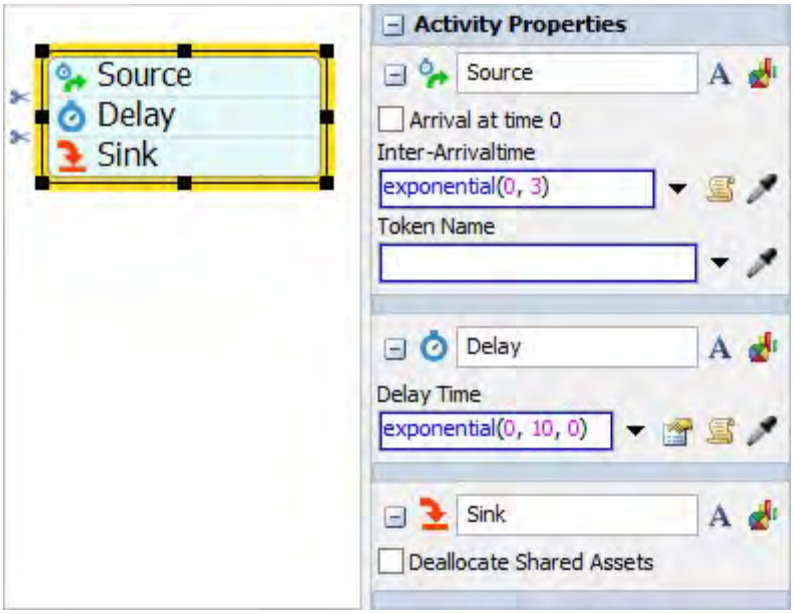
3. Make changes to the activity's properties as needed and then click outside of Quick Properties to save the changes.

Changing the Activity Name
You can change the name of the activity by double-clicking the name to select it and then typing a new name. See Name for more information about this property.

Editing Properties in the Quick Properties Pane

To edit an activity's properties in the Quick Properties pane, simply click the activity to select it. When it is selected, a yellow box will appear around the activity and the properties will appear in the Quick Properties pane (on the left of the FlexSim window), as shown in the following example:

Be aware that if you have joined activities together into a stacked block, you will select all the activities at the same time when you click anywhere on the stacked block. You should also be aware that the Quick Properties pane will display the properties for all the activities in the stacked block. The properties will appear in the order they appear in the block, as shown in the example in the following image:

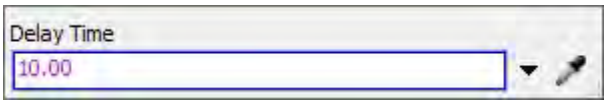


You can click the Collapse button in the top left corner of an activity's group of properties to hide those properties from view.

About the Universal Edit Feature

Many of the activities in process flow have special properties that use the Universal Edit feature. The Universal Edit feature is designed to make it easy to add complex functionality to certain properties without needing to know FlexScript.

You can tell when a property has the Universal Edit feature because it will have a blue border around the property box, as shown in the following image:



Properties that use the Universal Edit feature can accept the following possible types of input:

- Keywords - Keywords can act as a user-friendly substitute for Flexscript commands. Using keywords, you can set a property can access data that could possibly change dynamically during a simulation run such as labels, groups, variables, and more.

- Static Data - Universal Edit boxes can accept fixed, static data that won't change during a simulation run. It includes standard data types such as numbers, strings, and references to specific objects in the simulation model.
- Commands - If you are comfortable with FlexScript commands, you can still use FlexScript commands in Universal Edit boxes. These boxes can also sometimes accept SQL commands.

Each of these types of inputs will be explained in more detail in the following sections.

Not All Properties Support All Input Types

Even though Universal Edit boxes can accept a wide variety of types of input, some properties may not be able to use certain types of inputs. If the property doesn't support a particular type of data, you'll get an error message from the system console when you try to use that type of input. You could possibly avoid this problem by clicking the arrow next to the Universal Edit box to open a menu. This menu will show possible options of input types that are valid for the given property.

Keywords

Keywords can act as a user-friendly substitute for Flexscript commands. Using keywords, you can set a property can access data that could possibly change dynamically during a simulation run such as labels, groups, variables, and more.

When you begin typing a particular keyword, the FlexSim system will automatically generate a menu with possible options that could auto-complete the keyword phrase. For example, if you want a property to use the value from a label on a token, you can type the keyword `token.` . After typing that keyword, the system will search for all the available labels that you've created in the process flow so far and list these labels in a menu, as shown in the example in the following image:

You can then select the appropriate label from the list or continue typing the label name manually. During the simulation run, the property will look for that label on the entering token. The property will then use the value listed for that label.

The following table lists the different keywords that Universal Edit boxes can recognize:

Keyword	While Editing the Property	During a Simulation Run
Group:	Typing this keyword will create a menu listing the names of any groups of objects that you have created in the simulation model so far. Select the appropriate group name from the menu.	During a simulation run, this property can dynamically reference a group of objects rather than a specific object. For example, a Wait for Event activity could listen for any time a flowitem leaves any queue that is in a group of queues.

Variable:	Typing this keyword will create a menu listing all the process flow variables that have been created so far. Select the appropriate variable from the menu.	During a simulation run, this property can reference the value of a particular value. See Process Flow Variables for more information about how variables work.
SubFlow:	This keyword only works correctly with two process flow activities: the Run Sub Flow and Create Tokens activities. You would typically use this keyword with the Destination property to determine which sub flow will be triggered or which sub flow should receive the newly-created tokens. Typing this keyword will create a menu listing the names of any sub flows that you have created so far. Select the appropriate sub flow from the menu. NOTE: Most of the time you will use the sampler button to select the sub flow, rather than typing in a keyword.	During a simulation run, this property will trigger the sub flow listed in the Destination property.
ProcessFlow:	Similar to SubFlow, some activities can reference activities or shared assets in other Process Flow objects. The text takes the form ProcessFlowName: ActivityName. The referenced activity or shared asset must be in a general process flow or sub flow type. NOTE: This is the only keyword that does not display as bold.	During a simulation run, the property will reference the specific activity listed in the process flow.

Static Data

Universal Edit boxes can accept fixed, static data that won't change during a simulation run. It includes the following standard data types:

- Numbers In the property can accept a number value, you can type a number directly in the field. If the property requires a whole number, any decimal values will be removed automatically.
- Strings You can enter strings into a universal edit by placing the text in quotes. For example, "Token Name".
- Objects You can reference a specific object in the simulation model or an activity in the process flow. To select an object, you can use the sampler button  to select the object in the simulation model or process flow. If the type of object you select is not valid for this particular property, you will get an error message.

Commands

If you are comfortable with FlexScript commands, you can still use FlexScript commands in Universal Edit boxes:

- Code If the Universal Edit property cannot resolve the entered text into any of the keywords or standard data types, the text will be assumed to be Flexscript code.
    - Example - duniform(1, 10)
- SQL Some properties associated with resources, lists, and zones allow you to enter SQL queries. These SQL queries may be entered directly into the universal edit or they may be stored on a label or access through a Process Flow Variable.
    - Example - WHERE label < 10

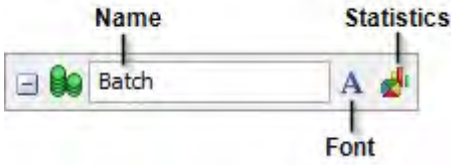
Common Process Flow Object Properties

Some of the process flow objects have properties that they share in common. Each of these properties will be discussed in more detail in the following sections.

- Name
- Font
- Statistics
- Max Wait Timer
- Max Idle Timer
- Executer / Task Sequence
- Assign To

Name

All the activities and process flow objects have a Name box that you can use to edit its name. You can find this setting in Quick Properties under the Activity Properties group. By default, the name of the activities and other process flow objects will be based on what kind of activity or object it is, as shown in the following image:



In the example above, the activity is named *Batch* because it is a Batch activity.

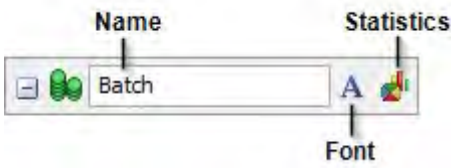
You can change the name of the activity by clicking inside the Name box and typing a new name. After you've changed the name, it will update the display name on the activity.

Be aware that you can also change an activity's name by double-clicking the activity name in the process flow.

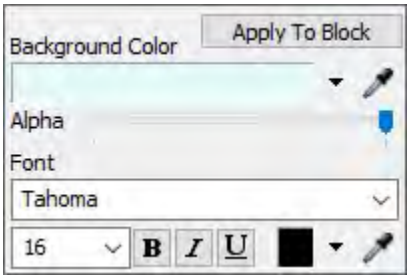
Unique activity names?
Activity names do not need to be unique. However, referencing an activity by name will return the first activity with the given name.

For more information about the Font and Statistics properties, see the following sections.

Font



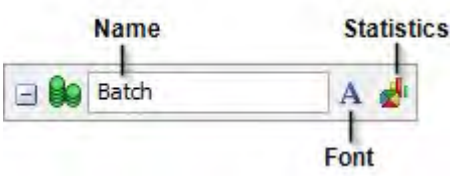
Clicking the Font button A opens a pop-up box where you can edit the activity's visual properties, as shown in the following image:



This pop-up box has the following properties:

- Apply to Block - This button will cause the background color and font of the selected activity to be copied to all activities that are stacked together with this activity.
- Background Color - You can change the color of the activities by using the color selector or using the Sampler button to select a color from any object in your simulation model or process flow.
- Alpha - Sets the transparency of the activities.
- Font - These properties control the text on activities and resources:
    - Font - The font menu has 13 common fonts available. o Size - Changes the font size. o Style - You can make the text bold, italicized, or underlined.
    - Color - You can change the color of the text by using the color selector or using the Sampler button to select a color from any object in your simulation model or process flow.

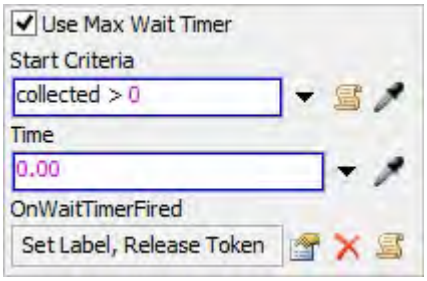
Statistics



The Statistics button opens the activity's statistics dialog box. You can use this dialog box to turn recording for statistics on or off for this particular activity. See Process Flow Statistics for more information.

Max Wait Timer

Some activities have the Max Wait Timer properties, as shown in the following image:



The Max Wait Timer properties are available for activities that might hold a token for a period of time until a specific condition or event occurs in the process flow or simulation model. You can use this timer to:

- Set a maximum wait time that the activity will hold the token for
- Determine when the wait time will begin
- Decide what will happen to the token if it reaches the maximum wait time

For example, an Acquire Resource has a Max Wait Timer. Tokens that enter the Acquire Resource will request access to the resource and will wait in the Acquire Resource activity until the resource becomes available. You could set the Max Wait Timer to expire if the token is unable to get access to the resource after 5 minutes. Then, when timer goes off, you could possibly set the token to create a label named *failed* and then continue to the next downstream activity.

The Max Wait Timer properties are available on the following activities:

- Batch
- Wait For Event
- Push To List
- Pull From List
- Acquire Resource
- Enter Zone
- Join
- Synchronize

By default, only the Use Max Wait Timer checkbox is available at first. Then, when you check the Use Max Wait Timer checkbox, the other properties will become available:

Start Criteria

The Start Criteria box is only available for Batch, Join, and Synchronize activities.

The Batch activity collects incoming tokens and sorts them into groups of tokens (batches). When a batch is ready, the Batch activity will release it to a downstream activity. If you decide to use the Max Wait Timer with a Batch activity, you can cause the Batch activity to release a batch early if a certain amount of time has passed. You'd also use the Start Criteria box to determine when the timer should begin running.

By default, the timer is set to begin running as soon as a batch is created. You can change the value in this box if needed. For example, if you change the Start Criteria to `collected > 3`, the timer will begin when the fourth token in the batch is collected.

This behavior is similar for the Join and Synchronize activities, except that they form waves (a group representing one token from each incoming connector) instead of batches. You can similarly release the entire wave early with this timer.

Time

Use the Time box to set the length of time the Max Wait timer will run. The time is measured in simulation time units. You can enter in a fixed time or create a time dynamically using the menu next to the box.

OnWaitTimerFired

Use the OnWaitTimerFired settings to determine what should happen to the token if the Max Wait Timer expires. You can click the View Properties button to view and edit the default settings, as shown in the following image:

By default, the Set Label operation will create a label on the token that is called *failed* and assign it a value of 1 (which will represent a value that is set to *true*). The Release token operation will then release the token through connector 1. (See Adding and Connecting Activities - Number of Outgoing Connectors for more information about connector numbers.)

You can edit these default operations or delete them using the Delete button next to each one. You can also add your own custom operations using the Add button to open a menu and select other operations.

Batches

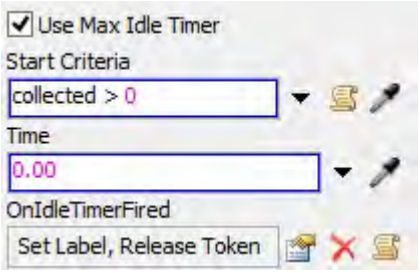
If you are using a Batch activity, there is an option to release the entire batch. From the menu, select Code Snippet then type releasebatch.

Waves

If you are using a Join or Synchronize activity, there is an option to release the entire wave. From the menu, select Code Snippet then type releasewave.

Max Idle Timer

The Max Idle Timer works almost identically to the Max Wait Timer. It has many of the same available settings, as shown in the following image:



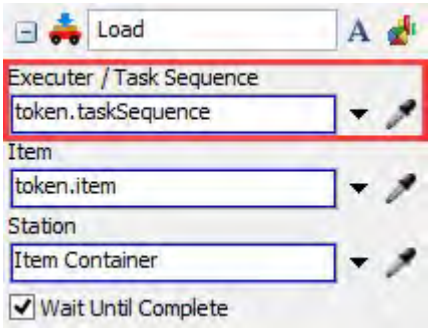
The main difference is that it measures how long a token has been idle in the activity, meaning how long it has gone without receiving any tokens and/or values.

See Max Wait Timer for more information about these properties.

This property is available on the following activities:

- Batch
- Pull From List
- Join
- Synchronize

Executer / Task Sequence



The Executer / Task Sequence box is available on most of the Task Sequence activities. You can use this box to determine which task executer or task sequence should receive the task. If you choose to give this task to a task executer, a new task sequence will be automatically created with this task and then it will be sent to the task executer. You can:

- Assign a specific, fixed task executer in the 3D model
- Dynamically assign the task executer during a simulation run using labels
- Assign the task executer using the `current` command if you are in a task executer or sub flow process flow type
- Append this task to an existing task sequence

Each of these different options will be explained in the following sections.

Should I Assign a Task Executer or a Task Sequence?

If you want to assign this task to a specific task executer and you're not concerned about whether that task executer may get pulled away to work on other tasks, assign the task to a task executer.

However, if your task executer will receive task sequences from multiple sources (more than one Fixed Resource or Task Executer object in the 3D model or via multiple tokens in a process flow), your task executer could possibly get pulled away to work on other tasks it receives. In other words, the task executer might be working on a set of tasks it received from one object and then get interrupted and work on a task

it receives from another object. If you want to prevent or control this interruption, you should use a Create Task Sequence activity and then assign tasks to this task sequence. See the Create Task Sequence activity for more reasons why you might want to use a Create Task Sequence activity.

Fixed Task Executers

To assign this task to a specific task executer, use the Sampler button to select a task executer in the 3D model. During the simulation run, the assigned task executer will always perform this task.

Dynamic Task Executers - Labels

If needed, you can make sure this task is assigned dynamically to a task executer during a simulation run. In other words, you can change these settings so that a different task executer might be assigned to the task based on different conditions during the simulation run.

One way to dynamically assign a task executer is to use the task executer that is listed in a label on a token. To reference a label on a token, you can use the Label keyword:

1. Click inside the Executer / Task Sequence box and type *token.* .

2. A menu will appear after you type *token.* that will list all the available labels in the process flow. You can select a label from the menu to auto-complete the label name. Either double-click a label name or use the arrow keys and press the Enter key to select the label name. Alternatively, you can just finish typing the full label name after the colon. Be aware that the label name does not need to be in quotation marks (" ") and it is case sensitive.

The Current Command

You can use the `current` command in Executer/Task Sequence box to dynamically assign the task to the task executer that is currently attached to a specific instance of the process flow. Be aware that this command can only be used for the Task Executer or Sub Flow process flow types. The keyword `current` will reference the task executer object attached to the process flow. See Process Flow Instances for more information. Also see the Task Sequences Tutorial for an example of how to dynamically assign task executers using the `current` command.

Adding the Task to an Existing Task Sequence

If needed, you can add this task to an existing task sequence that was previously created by a Create Task Sequence activity. (See Creating and Dispatching Task Sequences and the Create Task Sequence activity for more information.)

The Create Task Sequence activity will create a reference to the created task sequence and assign it to a label on the token. This label can then be used in the Executer / Task Sequence box to add the task to the end of that task sequence.

If you add any of the task activities to the end of a Create Task Sequence activity or another task activity, the newly added task activity will automatically put the correct label name in the Executer / Task Sequence box. This can make creating task sequences more convenient. However, if you need to add it manually:

1. Click inside the Executer/Task Sequence box and type *token.* .

2. A menu will appear after you type *token.* in this box. Type the same name of the label used in the Create Task Sequence activity's Assign To property.

Assign To

The Assign To property creates a reference to a new value(s) or object(s) that is created by the activity. These references are usually assigned to a label on a token, but they may be assigned to other labels or nodes. When using the `token.labelName` syntax, the label will be created on the token if it does not currently exist. Otherwise, you will need to ensure that the node you pass in to the Assign To already exists. This can be done by using the `object.labels.assert("labelName")` command.

The reference may be to a single object or value, or it may be to multiple. For instance, pulling entries off of a list may result in one entry or multiple. If multiple entries are pulled, an array will be created with each entry in the array being one of the pulled values.

Creating a reference point allows other activities to easily reference created objects, values pulled from a list, task sequences etc. However, an Assign To label/node is not required for and may be removed by clicking the Remove button.

The value(s) will be set in one of two methods:

Assign

If the Assign box is checked, any data stored on the label or node that was passed into the Assign To box will be overwritten by the new value.

Insert at Front

If the Insert at Front box is checked, any data stored on the label or node that was passed into the Assign To box will remain and the new value(s) will be added to the front. This will cause the data to become an array with the most recent value as the first entry in the array.

Linking Process Flows to 3D Models

Process flows can run independently from a simulation model if needed. But process flows are probably most useful when they are linked to simulation model. Process flows can help you build complex logic into your model, replacing the need for complicated code. Process flows will also make it easier for you to troubleshoot your model logic if needed. With that in mind, this topic covers how to integrate process flows with a standard simulation model.

There are two different ways to link your model to a process flow:

1. Events - You can use process flow activities that will listen for a certain event in the simulation model. When that event occurs, the activity can then initiate a process flow (by creating a token) or trigger another activity in a process flow.

2. Picklists - Some properties on fixed resources and task executers have menu options that can initiate a sub flow.

The following sections will give an overview of each method and will provide links for more information.

Events

Events are the most common way to link to a process flow from a simulation model. For example, you can set an activity in your process flow to listen for an item to enter a processor. Every time an item enters the processor, it would then kick off a series of activities that perform custom logic, perhaps to dynamically determine the processing time for that type of item. If you were not using a process flow to do this, you'd need to manually write code for the OnEntry trigger of the processor instead. The two activities that can listen for events are:

- Event-Triggered Source - This activity will listen for a specific event to occur in the simulation model. When that event occurs, it will create one or more new tokens and send the tokens to the next activity in the process flow. You should use an event-triggered source if you want the triggering event to initiate the process flow. See Event-Triggered Source for more information about this activity.

- Wait for Event - After this activity receives a token, it will hold the token while it listens for a specific event to occur in the simulation model (or in the process flow). When that event occurs, it will release the token to the next activity in the process flow. If needed, you can also make the Wait for Event act like a Decide activity. In other words, the Wait for Event activity could possibly send the token to different activities next based on certain conditions in the simulation model. See Wait for Event for more information about this activity.

See Key Concepts About Event Listening for more detailed information about events.

Picklists

Some properties on fixed resources have menu options (picklists) that can initiate a task sequence sub flow. A task sequence is a series of actions (tasks) in a sub flow that can be assigned to a task executer (such as an Operator or Transporter). (See Task Sequences for more information.)

For example, you can change a Processor's settings so that it will use a task executer during its setup time or processing time. A Processor can also use a task executer to transport flowitems to another downstream object when the item is finished being processed. If you need a task executer to complete a more complex series of tasks during setup, processing, or transportation, you could create a task sequence on a process flow. Then, you could use the ProcessFlow: Use Task Sequence Sub Flow picklist on the Processor's setup, processing, or transportation properties to reference that task sequence.

See Sub Process Flows and Task Sequences for more information. Also, Tutorial 2 - Linking to Simulation Models and Tutorial 3 - Task Sequences can give you some hands-on experience with these concepts.

Key Concepts About Event-Listening

Events are the most common way to link to a process flow from a simulation model. For example, you can set an activity in your process flow to listen for an item to enter a processor. Every time an item enters the processor, it would then kick off a series of activities that perform custom logic, perhaps to dynamically determine the processing time for that type of item. If you were not using a process flow to do this, you'd need to manually write code for the OnEntry trigger of the processor instead. The two activities that can listen for events are:

- Event-Triggered Source - This activity will listen for a specific event to occur in the simulation model. When that event occurs, it will create one or more new tokens and send the tokens to the next activity in the process flow. You should use an event-triggered source if you want the triggering event to initiate the process flow. See Event-Triggered Source for more information about this activity.

- Wait for Event - After this activity receives a token, it will hold the token while it listens for a specific event to occur in the simulation model (or in the process flow). When that event occurs, it will release the token to the next activity in the process flow. If needed, you can also make the Wait for Event act like a Decide activity. In other words, the Wait for Event activity could possibly send the token to different activities next based on certain conditions in the simulation model. See Wait for Event for more information about this activity.

This topic will discuss event-listening key concepts as they relate to process flow in more detail. It includes the following topics:

- Requirements for Event Listening
- Setting up Event-Listening on an Activity
- Listening to Events on a Flowitem or Token
- Listening to Events on Modeling Tools
- Examples of Event-Listening Scenarios

Requirements for Event Listening

The event-listening activities (the Event-Triggered Source and the Wait for Event activity) have two properties that are required in order to listen to events:

1. Object - You need to determine which object the activity should listen to. An activity could possibly listen to: o An object in the model. o An activity in the Process Flow.
    - The current instance object (See Process Flow Instances for more information.)
    - A group of objects. In this case, the activity will listen to the defined event on all members of the group.

2. Event - You need to determine which event on that object will trigger the activity. Be aware that you can listen to more than one event on an object. There are two different types of events:
    - *Standard Events* - Normal events that are fired during the simulation. such as an item entering a fixed resource in the simulation model or a token entering an activity in a process flow.

- *Value Change Events* - Events that happen when some value in the model (usually a statistical value) changes.

See Event Types for more information about the two different types of events and the properties associated with them.

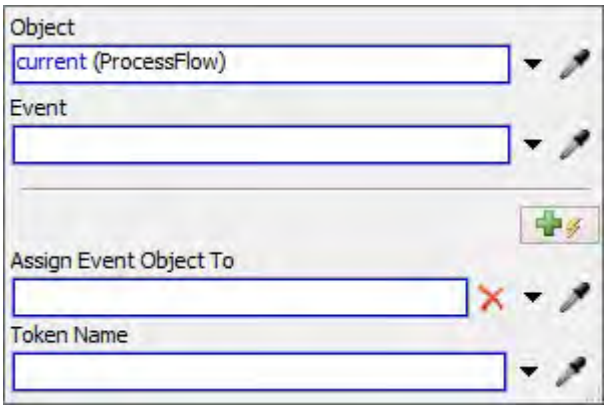
Sometimes the Object and Event Might Be the Same

In some cases, the event will be the same as the object. For example, you may connect directly to a node in the tree such as a user command (do this by pressing the button and then clicking on the Code field in the user command's properties window). In this case the Object field will be a reference to the node, while the Event field will say *Same as Object*.

Setting up Event-Listening on an Activity

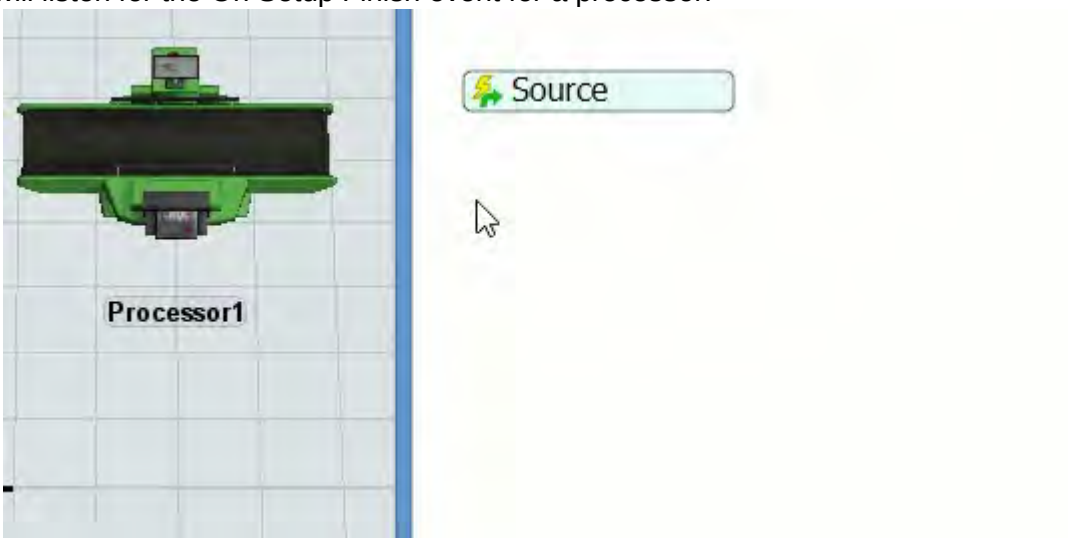
To set up event-listening on an activity:

1. Make sure that the object you want to listen to (such as a processor or another activity) has been added to your simulation model or process flow.
2. You might also want to split the center pane so that it is split between the simulation model and the process flow tab. (See Navigating in Process Flow - Rearranging Process Flow Tabs for more information.)
3. With the process flow tab active, add a Wait for Event or Event-Triggered Soruce to the process flow.
4. Click the activity's icon to open its properties, as shown in the following image:



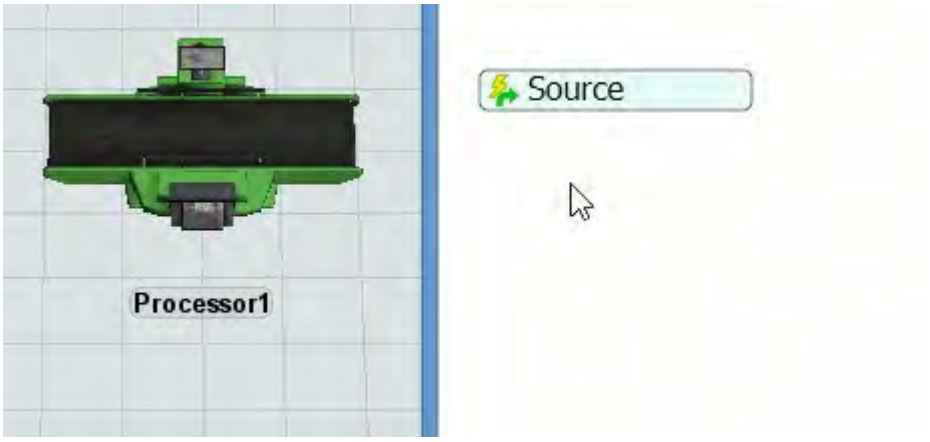
5. First, you'll select the object that the activity will listen to. Next to the Object box, click the Sampler button to enter sampling mode.
6. Click the object you want to listen to. This will open a menu that will list all of the possible events available for that object.
7. Select the appropriate event from the menu. That event will now appear in the Event box in the activity's properties.
8. Based on the type of event you selected, some additional properties will now be available in the activity's properties. See Event Types and Related Properties for more information about these properties.

The following image shows an example of this step-by-step process using an Event-Triggered Source that will listen for the On Setup Finish event for a processor.



Shortcut for Adding Events

There is a faster way to add an event to an Event-Triggered Source or Wait for Event activity. When you first add one of those activities to your process flow, you'll notice that an Exclamation icon next to the activitiy. If you click on this icon, you will enter sampling mode. In sampling mode, you can click object that you want the activity to listen to. This shortcut is the same as clicking the Sampler button in steps 5 and 6.



Listening to Events on a Flowitem or Token

It's often useful to listen to events on flowitems or tokens. Perhaps the path of a flowitem or token is not known or you want to reduce the number of events to listen to. Flowitems have the following events:

- On Entering - Fired when the flowitem enters a fixed resource or is loaded by a task executer object.
- On Exiting - Fired when the flowitem exits a fixed resource or is unloaded by a task executer object.

Tokens have the following events:

- On Entering - Fired when the token enters an activity.

- On Exiting - Fired when the token exits an activity.
- On Manual Release - Fired when the token is manually released from an activity using the releasetoken() command.
- On Asset Allocated - Fired when the token allocates a shared asset. This may be acquiring a resource, entering a zone being pushed onto a list or creating a backorder when pulling from a list.
- On Asset Deallocated - Fired when the token deallocates a shared asset. This may be releasing a resource, exiting a zone being pulled from a list, or completing a backorder.
- On Label Value Change Fired when a label value changes. This is only available if a label is stored as a Tracked Variable. See Value Change Listening below for more information.

Listening for an event on a flowitem or token can be achieved through the following steps:

1. Reference the flowitem/token - In order to listen to an event on a flowitem or token, you'll need to create a reference to it. This is most often done by storing a reference on a label through an Assign Labels or an Event-Triggered Source listening. You can also reference the flowitem/token through code. Flowitems can be easily referenced through the Create Object activity.
2. Run the model - Run the model in order to get a flowitem or token that can be referenced. This can be ANY flowitem/token.
3. Sample the flowitem/token - Use the or click on the sampler for the Object property to enter sample mode. Click on the flowitem/token and select the desired event to listen to. This will populate the Object and Event boxes.
4. Update the object reference - When sampling a flowitem/token, the full path to that object will be displayed in the Object box. This path will not be valid when the model runs. Instead we want to use the referenced flowitem/token from step 1. If your reference is stored on a label then you can reference the label using the syntax token.item

Value Change Listening

It can be useful to wait for a value on a label to change. This can be done by adding a Tracked Variable to a label and then listening to *OnLabelValueChange*. Once a token has a tracked variable set as one of its labels, click on the Event Sampler or and hover over the token to get the option of Token: On Label Value Change. For more information, see the Tracked Variables as Labels from the Assign Labels activity.

Event-Triggered Source

If using an Event-Triggered Source, there is no token entering the activity and the event listening is only set up on model reset, so you are not able to use an Event-Triggered Source to listen to token/flowitem events.

Listening to Events on Modeling Tools

Many of the tools found in the toolbox have events associated with them that you can listen to. This is especially useful for objects like the Time Table that will allow you to listen for down and up times. The following is a list of possible objects you could listen to:

- Time Tables: Listen to the Down/Resume Function or the On Down/Resume.
- User Events: Listen to the User Event firing.
- MTBF/MTTR: Listen to any of the MTBF/MTTR functions found on the Functions page.
- Global Lists Global Lists have many different Standard Events and Value Change events that can be listened to.

- User Commands: User commands execute code stored on a node. Any node that is evaluated using nodefunction() like user commands can be listened to.
- Model Triggers: Similar to user commands, model triggers like On Model Open can can also be listened to.

You can use the sampler to hover over objects in the Toolbox and select the desired event, or you can hover over the picklists in their properties windows to select one of these events.

Examples of Event-Listening Scenarios

The following sections give examples of different scenarios in which you might listen for events.

Example 1 - Static Object Reference, OnExit Event

Imagine you have a single queue in your model and you want to run a process flow every time a flowitem leaves the queue. To build this scenario, you'd need to do the following:

1. Build your simulation model, making sure to include the queue.
2. Create a process flow. Make sure the process flow is a General type.
3. Begin the process flow with an Event-Triggered Source activity.
4. Now link the Event-Triggered Source to the queue. In Quick Properties, next to the Object box, click the Sampler button and select the queue in the simulation model.
5. A menu will appear listing all the possible events for the queue. Select Queue1: On Exit.
6. Build the rest of the logic of the process flow by adding more activities after the Event-Triggered Source.

When the simulation model runs, any time a flowitem leaves the queue, the Event-Triggered Source will create a new token and release it to the next downstream activity in the process flow.

Example 2 - Dynamic Object Reference (Instance Object), OnLoad Event

Imagine you want to create a task sequence for loading and unloading flowitems. Perhaps you want to build one task sequence that will then be used by several different operators at different workstations. The task sequence will begin when they load a flowitem. To build this scenario, you'd need to do the following:

1. Build your simulation model, making sure to add at least two operators and some different processors (the workstations). Make sure that each processor is connected to an operator with a center port connection.
2. Create a new process flow. Make sure the process flow is a Task Executer type.
3. Begin the process flow with an Event-Triggered Source activity.
4. In Quick Properties, leave the Object box set to current (No Instance). This setting will ensure that the object can change dynamically to be whatever operator is using the task sequence in a given instance.
5. Next to the Event box, click the Sampler button and click one of the operators in the simulation model.

6. A menu will appear listing all the possible events for the operator. Select On Load.

7. Build the rest of the loading task sequence in the process flow by adding Load and Unload activities and any other tasks in between.
8. Now attach the operators to the Task Executer process flow. Click on a blank space in the process flow to ensure nothing is selected.

9. In Quick Properties, under Attached Objects, click the Sampler button and click one of the operators in the simulation model. Add all the other operators using this method until they are all attached to the process flow.

10. Lastly, open the Properties dialog box for all the processors in the simulation model. In the Flow tab, check the Use Transport check box so that the processors will use the operators connected to their center port.

When the simulation model runs, the operators will use the task sequence in this process flow whenever their OnLoad command is called by a processor in the simulation model.

Example 3 - Object Group, OnExit Event

Imagine you have several queues in your model and you want an Event-Triggered Source to create a token any time a flowitem exits from any of the queues. You could possibly add them to a group and then listen to the entire group. To build this scenario, you'd need to do the following:

1. Build your simulation model, making sure to include several queues.

2. Right-click on one of the queues to open a menu. Point to Object Groups, then select Add to New Group. This will create a group called *Group1* and assign the queue to that group.

3. Now you'll add the other queues to this group. Right-click another queue to open a menu. Point to Object Groups, then select Group1. Repeat this step until you've added all the queues.

4. Create a process flow. Make sure the process flow is a General type.

5. Begin the process flow with an Event-Triggered Source activity.

6. Now link the Event-Triggered Source to the group. In Quick Properties, next to the Object box, click the arrow to open a menu. Point to Group, then select Group1.

7. Build the rest of the logic of the process flow by adding more activities after the Event-Triggered Source.

8. Next to the Event box, click the Sampler button and click one of the queues in the simulation model.

9. A menu will appear listing all the possible events for the queue. Select On Exit.

10. Add the rest of the process flow activities as needed.

When the simulation model runs, the Event-Triggered Source will create a token any time a flowitem leaves the queues in the group.

Example 4 - Dynamic Object (Flowitem), OnExiting Event

Imagine you want to create a process flow that will listen for flowitems exiting any fixed resource. One problem you might encounter is that there might be multiple items in a fixed resource at the same time (such as a queue or a conveyor). You'll want to make sure it only releases the token that matches the flowitem it is associated with. You don't want to release the other tokens that are associated with flowitems that are still inside the queue or conveyor. To build this scenario, you'd need to do the following:

1. Build your simulation model, starting with a source and making sure there is a conveyor somewhere downstream in the model.

2. Create a new process flow. Make sure the process flow is a General type.

3. Begin the process flow with an Event-Triggered Source activity. Now you'll change the settings so that the Event-Triggered Source will create a new token every time the source creates a flowitem in the simulation model.

4. In Quick Properties, next to the Object box, click the Sampler button and select the source in the simulation model.

5. A menu will appear listing all the possible events for the source. Select Source: On Creation.

6. Now you'll assign each token a label named *item* that tracks the flowitem's unique ID. Notice that the Label Assignment table has now become available in Quick Properties. In the cell that is on the Created Item row underneath the Label Name column, type *item*. This will assign the token an *item* label that tracks the created item's ID.

7. Add any other additional activities to your process flow as needed. Somewhere in the process flow, add a Wait for Event activity.

8. In Quick Properties, in the Object box type token.item. This setting will listen for the *item* label tokens and will track the flowitems that are associated with that label.

9. Now make sure the Toolbox is open in the left pane and that the FlowItem Bin is expanded.

10. In the Quick Properties for the Wait for Event activity, next to the Event box, click the Sampler button and click the Box in the FlowItem Bin. This will open a menu that will list all of the possible events available for flowitems.

11. Select On Exiting from the menu. That event will now appear in the Event box in the activity's properties.

12. Add the rest of the process flow activities as needed.

Now when a token enters the Wait for Event activity, it will only release the specific token that is associated with a specific flowitem.

Event Types and Related Properties

Events are the most common way to link to a process flow from a simulation model. The two process flow activities that can listen for events are the Event-Triggered Source and Wait for Event activities. (See Key Concepts about Event Listening Activities for more information.)

Available Events

Your first question when talking about events might be, what events can be listened to in FlexSim? Events can come from 3D objects, flowitems, tokens, activities, shared assets, modeling tools (Time Table, User Commands, etc). Use the sampler and hover over an object in the view or over a properties window to discover which events can be listened to.

There are two basic types of events: standard events and value change events. In a process flow, the type of event you are listening for will affect which properties are available on the event-listening activities. For example, when a Wait for Event activity is set to listen to an item entering a processor, the Label Assignment property would become available because an item entering a processor is a standard event. The following table explains the two different types of events and their related properties:

Type of Event	Definition	Examples	Related Properties
Standard	Normal events that are fired during the simulation.	An item enters a fixed resource in the simulation model A token enters an activity in a process flow An item finishes being processed in a processor An item exits a fixed resource	Label Assignment
Value Change	Events that happen when some value in the model (usually a statistical value) changes.	OnInputChange OnOutputChange OnContentChange OnStaytimeChange	Change Rule

The properties of Standard and Value Change events are explained in the following sections.

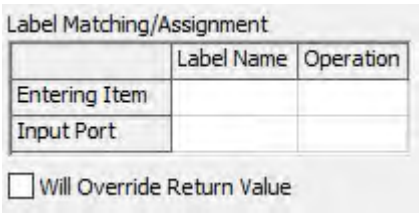
Label Assignment

The Label Assignment table (which might also appear as the Label Matching/Assignment table) will become available on an event-listening activity when that activity is listening for a standard event. You can use this table to assign or match a label to a token for tracking purposes. The Event-Triggered Source will assign the label to the token it creates; the Wait for Event activity will assign the label to the the token that entered the activity and triggered the event-listening.

You can leave this table blank if you don't need to do any label operations.

Parameter Rows

The rows in the Label Assignment table will vary depending on the specific event that the activity is listening for. Each simulation event has a set of parameters (sets of information) that it tracks. For example, the following image shows the Label Assignment table for the OnEntry event on a fixed resource:



In this example, notice that the first row is the *Entering Item*. This row is a reference to the item that is entering the fixed resource. The second row is *Input Port* which is a reference to the port number from which the fixed resource received the flowitem.

Most of the time, the name of the row will be descriptive enough to give you a good idea of its reference point.

Label Name

When using an Event-Triggered Source activity, the cells under the Label Name column determine the name of the label that will be created on the created token for this reference point. To use the previous example, you could create a label called *item* or *itemID* to refer to the entering item.

When using a Wait for Event activity the Label Name column will behave differently based upon the selected Operation (as described below). You are not limited to a label on the entering token. Periods separating label names can be used to reference labels on objects that the token has a reference to. For example, if the entering token has a label called *operator* which references another token or a task executer object in the model you can enter *operator.item* to reference a label on the operator.

Operation

The Operation column is only available for the Wait for Event activity. If you click on any cell under the Operation column, it will open a menu. The option you choose will determine what operation will be performed on the given label. The menu has the following options:

- none - The default value. Does nothing with the label.
- match - This will match the label's existing value to the parameter that is being referenced. The token will only finish the activity when the event fires AND the event's parameter matches the value of the token's label. For example, you may be tracking an item through the system. The token's *item* label points to the flowitem in the model. You want to catch when that specific item enters a fixed resource (like a queue) in the model. To do that, you can subscribe to the OnEntry event of the fixed resource, and then for the Entering Item row of the table, enter *item* in the Label Name column, and *match* as the operation.
- assign - This will assign the involved parameter value to the token's label value.
- insert at front - This operation is like the assign, except that if the label already has a value, it will make the label value into an array and push the new value to the front of the array. You can use this option if perhaps you are looping and you want to accumulate values into an array on the label.

Will Override Return Value

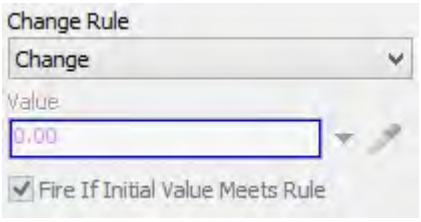
Event-listening activities have the ability to override the return value of events they listen to. For example, you may want to perform some complex logic using Process Flow to define the Process Time of a Processor. To do this, first check the Will Override Return Value checkbox in the activity's properties. Next define the

set of activities that determine the return value. Here you must make sure that none of these activities will cause any type of delay, such as a waiting operation or explicit Delay. Otherwise the return value will not be calculated properly. At the end of your block of activities place a Finish activity. The Finish activity allows you to define a return value for your overriden function and then destroys the token. In our example, the return value will become the Process Time of the Processor.

No Delays When Overriding Return Values
Using the Override Return Value functionality with activities that wait or create delays will cause the Finish activity to evalute the return value before the token arrives at the Finish and may result in unexpected return values.

Change Rule

The Change Rule properties will become available on an event-listening activity when that activity is listening for a value change event. You can use these properties to determine the conditions that will trigger the event. Usually this will be a statistical change of some sort. When these conditions are in place, the Event-Triggered Source activity will create a token and release it to the next downstream activity; the Wait for Event activity will release the token to the next downstream activity. The following image shows the Change Rule properties:



The following sections describe the different Change Rule properties:

- Change Rule You will use the Change Rule menu to determine the conditions that will trigger an event. The menu has the following options:
  - Change - The event will be fired whenever the value changes at all. o Increase - The event will be fired whenever the value increases. o Decrease - The event will be fired whenever the value decreases.
  - Arrive At Value - The event will be fired whenever the value changes to a specific user-defined value.
  - Increase To Exact Value - The event will be fired whenever the value increases to a specific userdefined value. o Decrease To Exact Value - The event will be fired whenever the value decreases to a specific user-defined value.
  - Increase To Or Through Value - The event will be fired whenever the value changes from being less than a user-defined value, to being greater than or equal to that value. o Decrease To Or Through Value - The event will be fired whenever the value changes from being greater than a user-defined value, to being less than or equal to that value.
  - First Increase - The event will be fired whenever the value changes from decreasing to increasing. o First Decrease - The event will be fired whenever the value changes from increasing to decreasing.

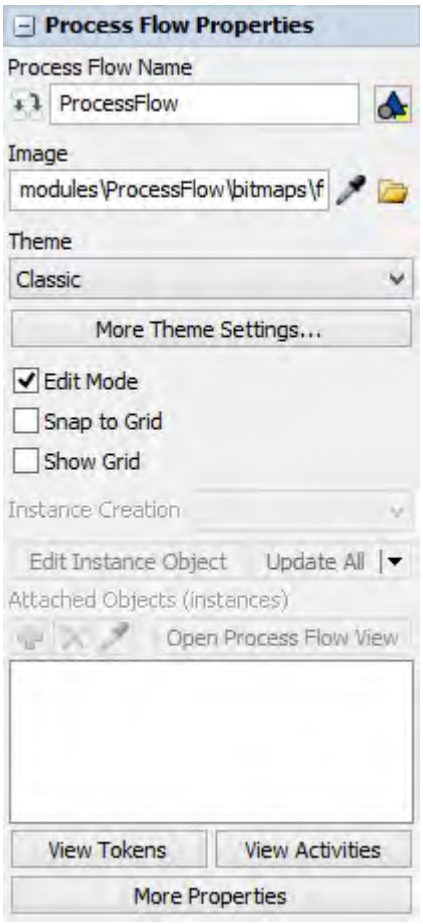
- Value Defines the user-defined value associated with the Change Rule. This is only needed for change rules with required values, such as *Arrive at Value*.
- Fire If Intial Value Meets Rule If checked, the event will fire immediately (finishing the activity) if when the token first arrives, the value already meets the defined rule. For example, if are listening to a Zone's OnContentChange event, and you've defined the Change Rule as *Arrive At Value* with a Value of *5*, then when the token arrives, if the Zone's content is already at 5, the event will immediately fire and the token will pass through the activity. This field is only applicable if the Change Rule uses an involved value, and if the activity is a Wait for Event.

Additional Event Binding Requirements

In some cases, depending on the event you subscribe to, there may be additional fields required in order to properly bind to the event. For example, if you bind to a List's OnContentChange event, you will need to provide the Partition ID that you want to listen to. Just enter the Partition ID in its corresponding field.

General Process Flow Properties

Each process flow that you create in FlexSim has several general properties (settings) that affect the behavior of its activities and tokens. You can access and edit a process flow's general properties by clicking on an empty area somewhere in the process flow (so that nothing else is selected). The general properties will appear in the Quick Properties pane, as shown in the following image:



Each of these properties will be explained in the following sections.

Process Flow Name

You can use the Process Flow Name box to change the name of the current process flow. Simply click in the box and type a new name. By default, the name of the first process flow will be *ProcessFlow*. Subsequent process flows will be named *ProcessFlow1*, *ProcessFlow2*, etc.

Changing the name here will change the name that displays on the process flow's tab and also in the Toolbox.

You can also change the name in the Toolbox

To change a process flow name in the Toolbox, right-click the process flow and select Rename from the menu.

Image

The Image box sets the icon image for the process flow. This icon is displayed on the tab of the process flow and in the Toolbox. By default, all process flows use the flowchart icon . 

To change the icon, you can click the Browse button  to navigate to the icon file in your computer.

Theme

A theme is a group of visual settings that can be applied to an entire process flow. A theme can affect:

- The color of the background, activities, shared assets, tokens, etc.
- The styles of the connectors
- The font and style of the fonts
- The color and visual functionality of the containers (flow chart objects) See Using Process Flow Themes

for more information.

More Theme Settings

Use the More Theme Settings button to open the Theme dialog box. See Using Process Flow Themes for more information.

Edit Mode

The Edit Mode checkbox turns on and off the Edit Mode, which changes the Process Flow to be more user friendly for simply running a model. Turning Edit Mode off is mainly intended for use after creating a Process Flow. It will hide variables that are not set as User Accessible and it will make it so the default action when selecting a Process Flow is to open its properties window and not the model view window.

Snap to Grid

The Snap to Grid checkbox turns snapping to the grid on and off. By default, activities and display objects will automatically align to one another as they are being dragged. The grid offers an additional mechanism to neatly lay out objects.

Show Grid

The Show Grid checkbox displays or hides the the grid.

Instance Creation

The Instance Creation menu is only available on process flows that are using the Sub Flow type. This menu defines when the FlexSim system will create an instance of a sub flow. (See Process Flow Instances for more information about instances and related concepts.) This can possibly affect whether shared assets (Resources, Lists, and Zones) are global or local to instances. The menu has the following options:

- Global - If this option is selected, there will only be one instance of a sub flow running at a time. All shared assets will be shared within that sub flow.
- Per Instance - In this context, an instance refers to an object that is running the sub flow. So, for example, if two processors are using the same sub flow, each processor would be considered a separate instance. By default, when you select this option all shared assets will be local to each instance, meaning they won't share assets.
- Per Block - You might have activity blocks that each have a Run Sub Flow activity in them. Each Run Sub Flow activity might be connected to the same sub flow. When this option is selected, each activity block that runs a sub flow would be considered a separate instance. By default, when you select this option all shared assets will be local to each instance, meaning they won't share assets.
- Per Token - When this option is selected, each token that runs inside the sub flow will be a separate instance. By default, when you select this option all shared assets will be local to each instance, meaning they won't share assets.

Sharing Assets Between Instances

Each shared asset (Resources, Lists, and Zones) has a Type menu in its Quick Properties. You can use this menu to control whether shared assets are global or local to instances. The menu has two options:

- Global - Shared assets will be globally accessible by all instances. For example, if you have a Resource with a count of 3, any instance that uses that Resource will deplete the total count. If one instance uses all three Resources, another instance won't be able to use it.
- Local - Shared assets will only be locally accessible to instances. For example, if you have a Resource with a count of 3, each instance will have a local copy of that Resource with a count of 3. When an instance uses a Resource will only deplete the local count. If one instance uses all three of its Resources, another instance will still have access to its three Resources as well.

Attached Objects

You will use the Attached Objects properties to add, remove, and display attached instance objects. These options are only available for the Sub Flow, Fixed Resource, and Task Executer types of process flows. See Process Flow Types for more information.

View Tokens

You can click the View Tokens button to open a window where you can see an overview of all tokens during a simulation run. The View Tokens window shows the labels and details associated with all the tokens in the Process Flow. This is particularly useful for troubleshooting your process flow. See Troubleshooting Process Flows for more information.

View Activities

You can click the View Activities button to open window where you can see an overview of all the activities in your process flow. You can also use this window to change the statistics settings of the activities in the Process Flow and to set up tracing options, which can help with troubleshooting your process flow. See Troubleshooting Process Flows for more information.

More Properties

You can click the More Properties button to open additional general properties for this process flow. You can use this window to set the variables used in the Process Flow as well as visual settings, labels, and triggers.

Using Labels in Process Flow

This topic will discuss some of the key concepts related to understanding and using labels in the Process Flow module. It will contain the following topics:

- What Are Labels?
- Label Names and Label Values
- Using the Assign Labels Activity
- Other Activities That Can Assign or Match Labels
- The Universal Edit Feature
- Storing Object References

What Are Labels?

In FlexSim, labels store specific pieces of information on objects such as tokens and flowitems. Labels are key to the overall functionality of FlexSim because they can track important information or dynamically change what happens during a simulation based on different conditions in the simulation model.

You use labels to track information and create dynamic logic in your process flows. The Process Flow module activities and shared assets can use custom labels for a variety of purposes. The following list has a few examples:

- Decide activities can use a label to determine which downstream activity should receive an incoming token. For example, imagine you wanted the tokens in a process flow to represent three different products that might go through three different manufacturing processes. To simulate this kind of system, you could create a custom label on all tokens called *ProductType* that either has a value of 1, 2, or 3. When the Decide activity receives these tokens, it will evaluate the *ProductType* label and send the token to one of three possible downstream activities based on the value in that label.

- A Zone shared asset can use a label to restrict access and collect statistics. For example, imagine you want to simulate a dump truck that can only hold a maximum weight of 5 tons and each token in a process flow will represent a varying amount of dirt that will be loaded into the dump truck. To simulate this kind of system, you could create a Zone to represent the dump truck and assign a custom label called *Weight* to each token. When each token tries to enter the dump truck (the Zone), it will evaluate the value of that token's *Weight* label and determine whether it would cause the dump truck to exceed its weight limit. If it would exceed the weight, the Zone would not allow that token to enter.

- A Batch activity can use a label to determine which batch should receive an incoming token. For example, imagine you wanted the tokens in a process flow to represent three different products that will get loaded onto different pallets based on their product type. To simulate this kind of system, you could create a custom label on all tokens called *ProductType* that either has a value of A, B, or C. When the Batch activity receives these tokens, it will evaluate the *ProductType* label and organize the incoming tokens into three different batches based on the value in that label. Each batch will represent a different pallet, so all tokens with a *ProductType* of A will go on one pallet, all tokens with a *ProductType* of B will go another pallet, and so forth. When each pallet is full, it will get released to the next downstream activity.

- Delay activities can use a label to determine how long a specific token should be delayed. For example, imagine you want to simulate a checkout line at a grocery store and each token will represent a customer coming through the line. Each customer could have any number of items. The clerk takes about 5 seconds

to scan each item and put it in a grocery bag. Therefore, the total amount of time the customer spends at the checkout line will depend on how many items she or he has multiplied by 5 seconds. To simulate this kind of system, you could create a Delay activity and assign a custom label called *NumberOfItems* to each token. When the Delay activity receives the token, it would evaluate the token's *NumberOfItems* label and then multiply that value by 5 to determine the delay time for that token.

If you'd like more examples, many of the tutorials demonstrate how to use labels in a sophisticated way in process flow.

As these examples illustrate, you can use labels to create a dynamic simulation model that is capable of adapting to many different circumstances or business systems. The rest of this topic will discuss some of the key concepts related to understanding and using labels in the Process Flow module.

For Further Reading
See the Labels topic in the standard User Manual for more general information on labels.

Label Names and Label Values

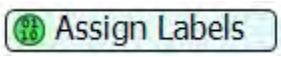
Every label has two elements:

- Name - Every label has a name that describes the type of information it contains. You'll use this label name to refer to the label and get information from it. The name of the label is assigned when the label is first created on the token and it won't change throughout the simulation run until the token is destroyed. If you create a custom label, you will define this name yourself. For example, you might create a label named *Weight* to keep track of a token's weight.

- Value - Every label has a value that can vary from token to token. Values can be any type of data, such as text, numbers, references to other objects, and even arrays. Label values may change during a simulation run. For example, you could randomly assign a number between 10 and 30 to a label named *Weight* on a token when it is first created. Then, after the token finishes a business process (represented by a Delay activity), it could be assigned a new label that subtracts 5 from the original value to represent its new weight.

Using the Assign Labels Activity

The majority of the time, you'll likely use the Assign Labels activity to create, set, and change labels. The Assign Labels activity doesn't delay a token in any way during a simulation run; it only creates or changes a label on a token. This means you can use an Assign Label activity anywhere in a process flow without slowing down the amount of time the token spends in the process flow. For example, Delay activities do not have the ability to assign or change labels, but you can use an Assign Label activity immediately after a Delay activity to change a label. Perhaps the new label can be changed to represent something about the token that changed as a result of going through the Delay activity.



See the Assign Labels activity for more information about this activity's properties.

Creating a New Label

To use an Assign Labels activity to create a new label on tokens in a process flow:

1. Add an Assign Labels activity to the process flow. See Adding and Connecting Activities for more information.
2. In Quick Properties, make sure the Assign To box displays token. This property means that this activity will assign labels to all incoming tokens.
3. Under the Labels group, click the Add button to add a new label.
4. In the Name box, delete the existing text and type a new name that describes the information this label will contain. For example, if the label will represent the token's weight, name the label *Weight*.
5. In the Value box, you will assign the value of the label. There are many different options for setting the value:
    - To set a fixed, static label that will be the same for all tokens, simply type a number or text in this box. For example, if you type *10* in this box, all tokens will be assigned a value of 10 for this label.
    - To make the label refer to a specific object in the 3D model or an activity in the process flow, you can use the Sampler button to select that object.
    - To use a statistical distribution to assign a range of randomly generated values to a label, click the arrow next to the box to open a menu. Select Statistical Distribution to open the Distribution Chooser. Select an appropriate statistical distribution and edit the parameters. For example, if you wanted to create three different product types, you could use a *duniform* distribution strategy with a minimum of 1 and a maximum of 3. See Distribution Chooser for more information about how to use this tool and some of the available statistical distributions.
    - To assign a certain percentage of tokens certain labels, click the arrow next to the box to open a menu. Select By Percentage to open its picklist options. Use the Add button to add as many different percentage groups as you need. Then edit the percentages and the values that will be assigned to those percentages.
    - Feel free to experiment with some of the options that are available in the pull-down menu for this property.
6. Use the Add button to add any additional labels if needed. Use the Remove button to delete a label if needed.

Changing an Existing Label

To change an existing label on a token later in a process flow:

1. Add an Assign Labels activity to the process flow.
2. In Quick Properties, make sure the Assign To box displays token. This property means that this activity will assign labels to all incoming tokens.
3. Under the Labels group, click the Add button to add a new label.

4. In the Name box, delete the existing text and type the exact name of the label that you want to change. For example, if you want to change a label named *Weight*, type that exact name in this box.

5. In the Value box, you will assign the value of the label. A few of the options that might be useful for changing an label are: o Remove Label

   o Increment Label o Conditional Value

Other Activities That Can Assign or Match Labels

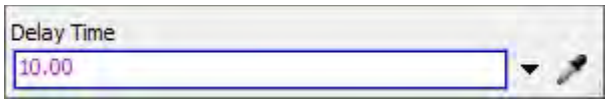
Many other process flow activities and objects can assign or match labels:

- Sub Flows - Managing parent and child labels between sub flows and the main process flow that initiated the sub flow can be crucial to a process flow's logic. See Sub Process Flows - Linking Parent and Child Labels for more information.
- Event-Listening Activities - Event-listening activities such as the Wait for Event or the Event-Triggered Source can match or assign labels on tokens based on relevant objects or values associated with the events they listen to. See Event Types and Related Properties for more information.
- The Batch Activity - The Batch activity has the capability of putting tokens into batches based on the token's label. See Organizing Batches for more information. It can also assign new labels to tokens that are released as part of a batch. See Releasing Batches for more information.

The Universal Edit Feature

Many of the activities in process flow have special properties that use the Universal Edit feature. The Universal Edit feature is designed to make it easy to add complex functionality to certain properties without needing to know FlexScript.

You can tell when a property has the Universal Edit feature because it will have a blue border around the property box, as shown in the following image:



Typing token. into a property field will search for the available labels that you've created in the process flow so far and list these labels in a menu. You can then select the appropriate label from the list or continue typing the label name manually. During the simulation run, the property will look for that label on the entering token. The property will then use the value listed for that label. See Editing Properties - About the Universal Edit Feature for more information.

Most of the time, you can do anything you want using labels and the standard logic on the process flow activities. But occasionally you might need to add additional FlexScript to your properties or picklists.

For example, consider the example of the grocery checkout line used in the first section of this topic. As a reminder, a Delay activity will represent the checkout line and each token will represent a customer coming through the line. The clerk takes about 5 seconds to scan each item and put it in a grocery bag. Therefore,

the total amount of time the customer spends at the checkout line will depend on how many items she or he has multiplied by 5 seconds. In this example, an Assign Labels activity could have been used to create a custom label called *NumberOfItems* with a randomly-generated value between 1 and 50. Then, in the Delay activity's Delay Time property, you could use the following command:

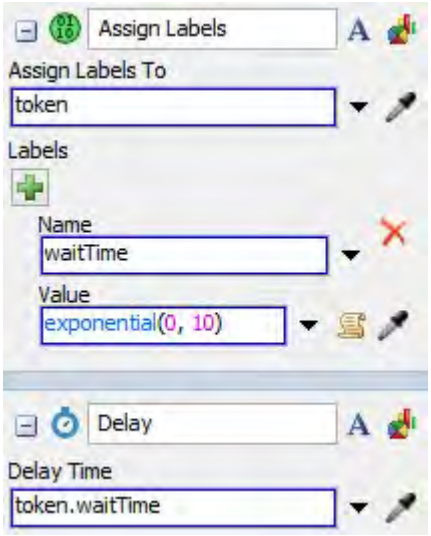
```
token.NumberOfItems * 5
```

As you can see, this command has a few different components:

- `token` - This indicates on which object the label can be found. In this case, it's the entering token.
- `NumberOfItems` - This parameter of the command uses the actual name of the label.
- `* 5` - This part of the command is the mathematical operation that will be performed on the label value.

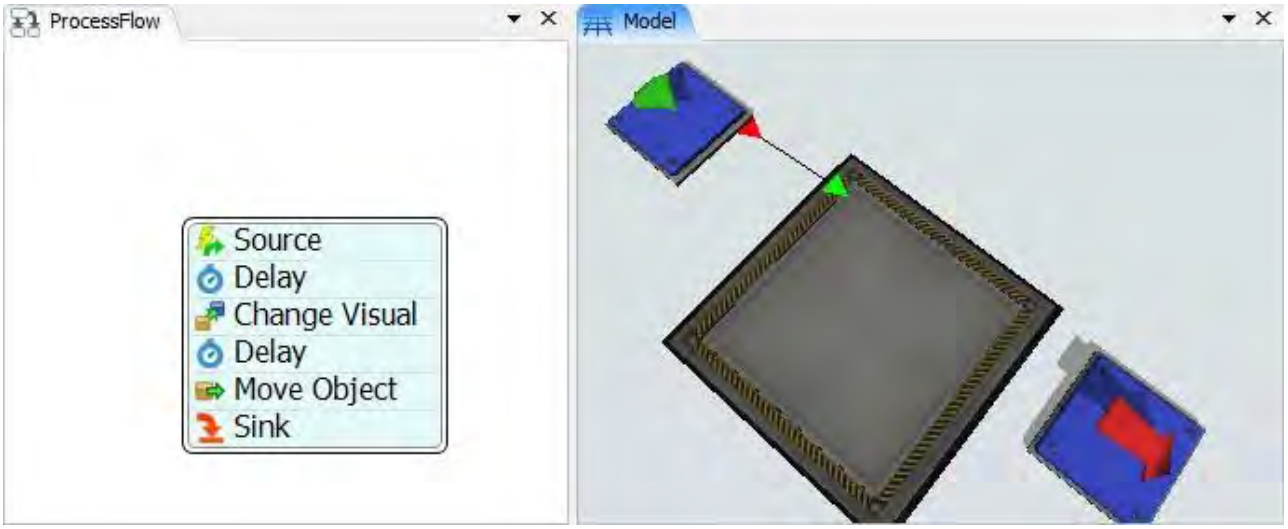
  Operations like `+ 1` or `/ 2` would have also been valid. Technically, this part of the command isn't even necessary if you just want to get the label value for this property.

The following image shows shows an example of setting and using a label. The Assign Labels activity gives each token a *waitTime* label, and the Delay uses the value on that label to determine how long to hold the token.

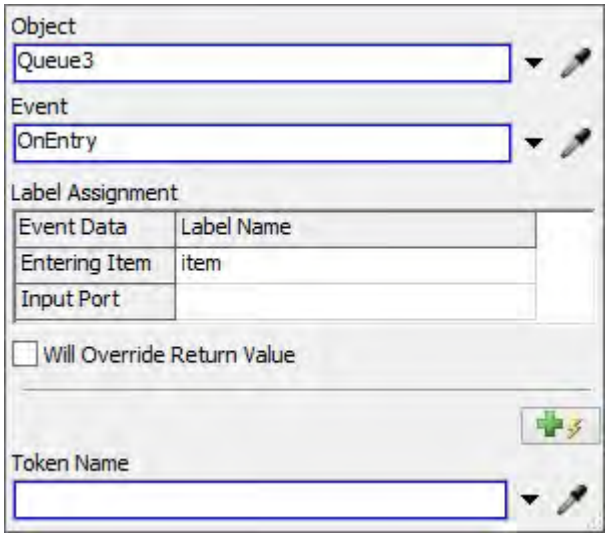


Storing Object References

Labels can be used to store references to other objects. This is how you can connect tokens with flowitems or with other tokens. For example, suppose you wanted to change the color of flowitems on a queue using a Process Flow. You might make a model like the following:



Each time a flowitem enters the queue, the Source activity (an Event Triggerd Source) creates a token. But how can we connect the new token with the flowitem? You can do this easily with labels. The Quick Properties for the Source is shown below:



Notice that the Entering Item of the event is assigned to a label on the new token called *item*. This means that token.item will return the flowitem.

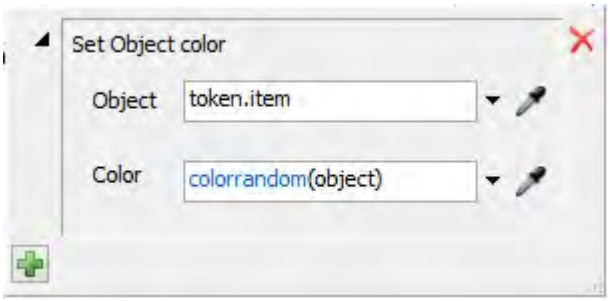
Object References Creating references to an object does not change the object in any way. The reference just provides quick access to that object.

An object reference is a treenode, so you can call any function that takes a node. A few examples are shown below:

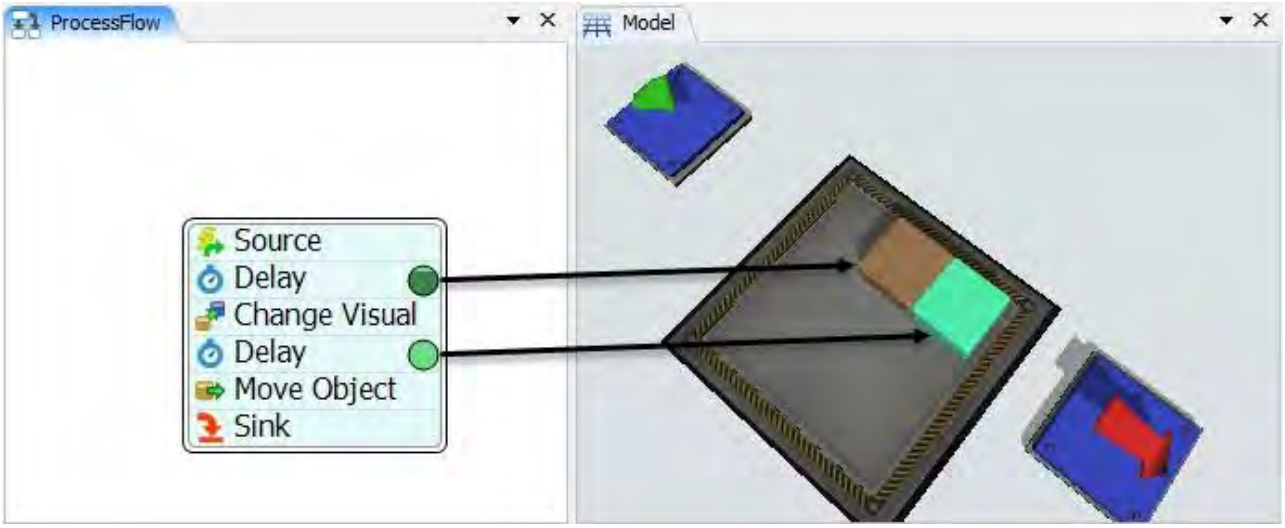
```
// get the name of the item string name =
token.item.name

// determine if the object still exists int exists =
objectexists(token.item);

// get the containing object Object container =
token.item.up; // ...
// ...
```

More applicable commands are discussed on the Basic Modeling Functions page.

Going back to the example model, you might fill out the Set Object Color popup on the Change Visual activity with the following values:



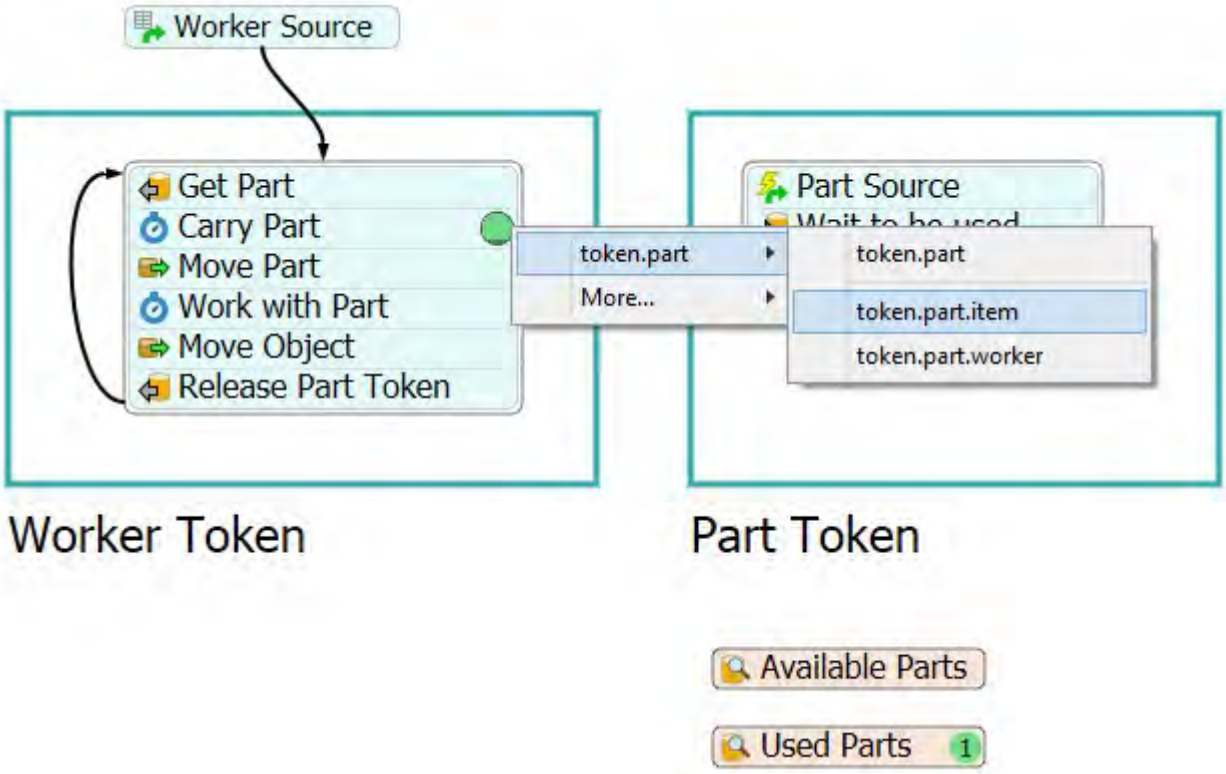
The effect can be seen as the model runs. The following image shows the *item* label of both tokens as black arrows, referencing the appropriate flowitem.



Multiple Label Calls

Sometimes, you may have a token with a label that references another token, and that token has a label that references another object, and so on. For example: token.item.operator

You can type this in a field manually, or tach of the property fields give you a sampler button that can make referencing these nested object references easier. In the image below, the model has been run to a point where tokens are in the activities. The model is then stopped and the properties window of an activity is opened. By clicking on one of the sampler buttons and clicking on a token, all of the valid labels stored on that token will appear.



Sampling this way can be very helpful as you build your model. If you need a reference like this, simply run your model until a token with the right labels is visible. Then stop your model, and sample the token.

Changing Process Flow Visuals

Each process flow has a variety of properties (settings) you can use to change the way process flows appear visually. You can use these settings to:

- Improve how your process flow communicates visually.
- Assist with debugging so that you can better observe potential problems in your process flow.
- Make your process flow more aesthetically appealing, perhaps for branding.

There are basically two tools in process flow that can control process flow visuals:

- The Visualization Tab - The Visualization tab is one of the tabs that are available in the Process Flow Properties. You can use these properties to change how tokens and activities will look during a simulation run. You can change things such as the whether to show or hide links between activities or child and parent tokens. You can also make the color or shape of tokens change dynamically during a simulation run.

- The Theme Dialog Box - You can use the Theme dialog box to change the default colors, text, connection visuals, and other visual properties. You can also use this dialog box to select one of FlexSim's pre-set visual themes or to create a new theme.

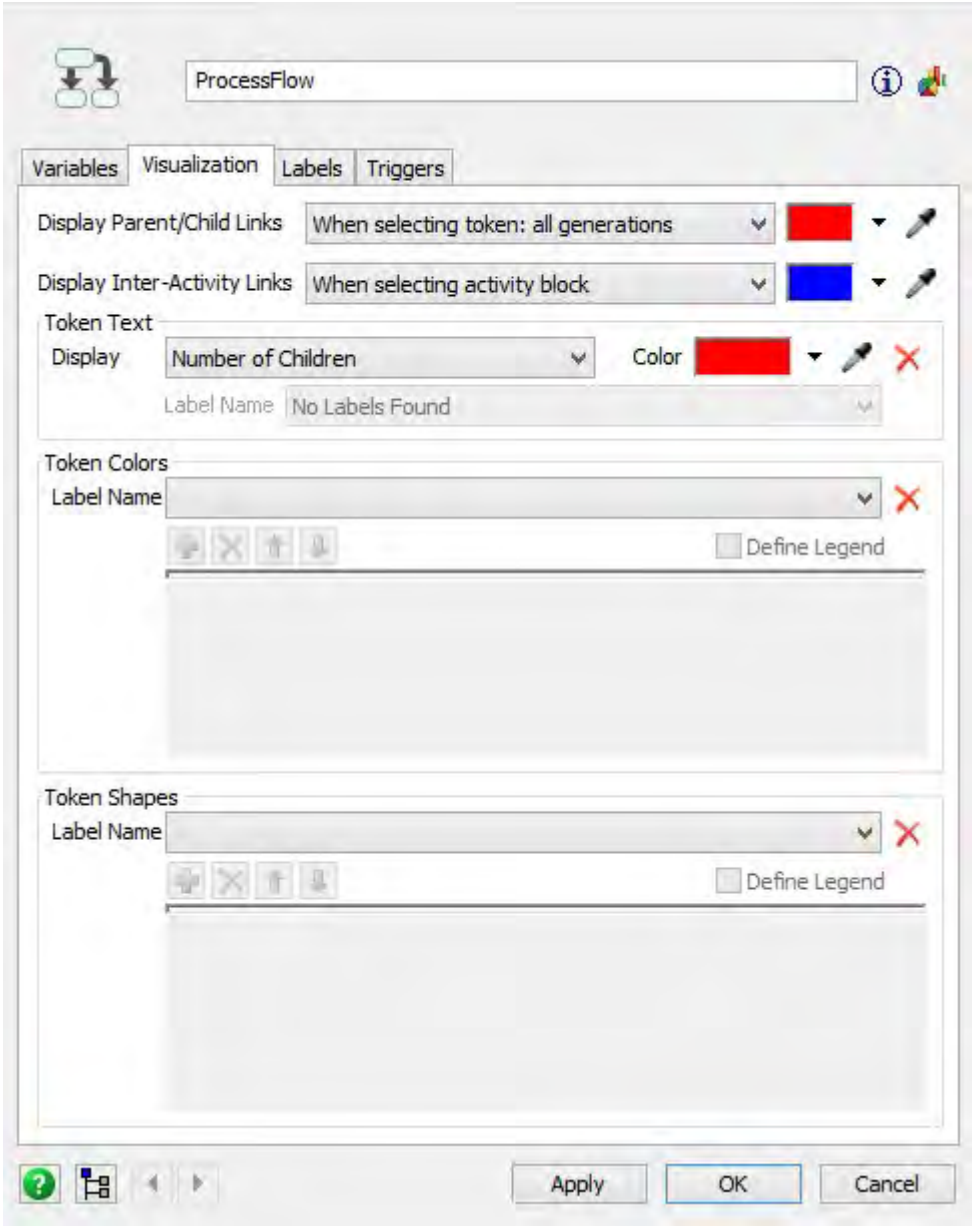
The Visualization Tab properties will be explained in the following sections. See Using Process Flow Themes for more information about the Theme dialog box.

The Visualization Tab

The Visualization tab is one of the tabs that are available in the Process Flow Properties dialog box. You can use these properties to change how tokens and activities will look during a simulation run. To get to this tab:

1. Click on an empty area somewhere in the process flow (so that nothing else is selected).
2. In Quick Properties, at the bottom of the Process Flow Properties group, click the More Properties button.
3. The Process Flow Properties dialog box will appear. Click the Visualization tab.

The properties that are available in the Visualization tab are shown in the following image:



Each property will be explained in the following sections:

- Display Parent/Child Links Use the Display Parent/Child Links menu to control how links between parent and child tokens will be displayed in the process flow. (See Sub Process Flows for an explanation of parent and children tokens.)

  Use the color selector next to the menu to determine the color of the link that will be drawn between parent and children tokens.

  This menu has the following options:

  - When selecting token: all generations - When you click on a token during a simulation run, you'll see a thin line between the parent and any of its children tokens, grandchildren tokens, etc. This option is the default.
  - When selecting token: one generation - When you click on a token during a simulation run, you'll see a thin line between the parent and any of its children tokens but not its grandchildren tokens. o Never display - The links between parent and children tokens will never be displayed.
  - Always display - The links between parent and children tokens will always be displayed, even when a token is not selected.
- Display Inter-Activity Links Use the Display Inter-Activity Links menu to control how links between activities will be displayed in the process flow. Links are different from connectors. Links are logical connections between activities that do not necessarily control how tokens flow from one activity to another.

  Any shared asset activities usually require an inter-activity link. For example, Acquire Resource activity will usually be linked to a Resource activity so that it can get permission to access that resource if it is available.

  Any activities that initiate a sub flow will require an inter-activity link. For example, Create Tokens and Run Sub Flow activities are usually linked to activities on a sub flow. (See Sub Process Flows - Linking to Sub Flows for more information.)

  Use the color selector next to the menu to determine the color of the link that will be drawn between linked activities.

  This menu has the following options:

  - When selecting activity block - When you click on a linked activity, you'll see a thin line between the linked activities. This option is the default.
  - Allow Toggling - You can manually show or hide a thin line between the linked activities directly in your process flow. To show or hide the links, select the activity block and click on the Eye symbol to switch between hiding and showing. o Never display - Inter-activity links will never be displayed.
  - Always display - Inter-activity links will always be displayed.
- Token Text Use the Display menu in the Token Text group to determine what text will display inside tokens during a simulation. The menu has the following options:
  - Number of Children - If a token has children tokens, it will display how many children it currently has, as shown in the following image:

  - Label Value - The token will display the value of a label. Use the Label Name menu to select the label that the token will get the value from.
  - Custom - The token will display custom information. Use the Custom box to select the information that will be displayed.

  Use the color selector to determine the color of the text that will display inside the token.

- Token Colors You can use the Token Colors settings to dynamically change the color of tokens during a simulation run based on a label value. For example, in the following image, the tokens will each have a different color based on the value in the itemType label:

The Label Name menu sets the label that you will use to define the color scheme for the tokens in the Process Flow. It will dynamically search for any labels that exist in the process flow.

If the Define Legend checkbox is checked, you will be able to set the colors that will be associated with certain values on the label. Click the Add button to add a color. Click inside the newly added color to change the color or the value it should be associated with in the label.

- Token Shapes You can use the Token Shapes settings to dynamically change the shape of tokens during a simulation run based on a label value. For example, in the following image, the tokens will each have a different shape based on the value in the itemType label:

The Label Name menu sets the label that you will use to define the shapes for the tokens in the Process Flow. It will dynamically search for any labels that exist in the process flow.

If the Define Legend checkbox is checked, you will be able to set the shapes that will be associated with certain values on the label. Click the Add button to add a shape. Click inside the newly added shape to change the shape or the value it should be associated with in the label.

Using Process Flow Themes

A theme is a group of visual settings that can be applied to an entire process flow. A theme can affect:

- The color of the background, activities, shared assets, etc.
- The styles of the connectors
- The font and style of the fonts
- The color and visual functionality of the containers (flow chart objects)

FlexSim comes with six different pre-set themes you can use. You can use these themes to quickly implement these visual styles. You can also adapt these visual themes or make your own custom themes using the Theme dialog box.

This topic will explain how to use the process flow themes. It will include the following topics:

- Opening the Theme Dialog Box
- The Six Pre-Set Themes
- Applying a Pre-Set Theme
- Creating a Custom Theme
- Theme Dialog Box Properties

Opening the Theme Dialog Box

You can use the Theme dialog box to change the default colors, text, connection visuals, and other visual properties of the process flow. You can also use this dialog box to select one of FlexSim's pre-set visual themes or to create a new theme. You can also save your custom themes and use them in future process flows.

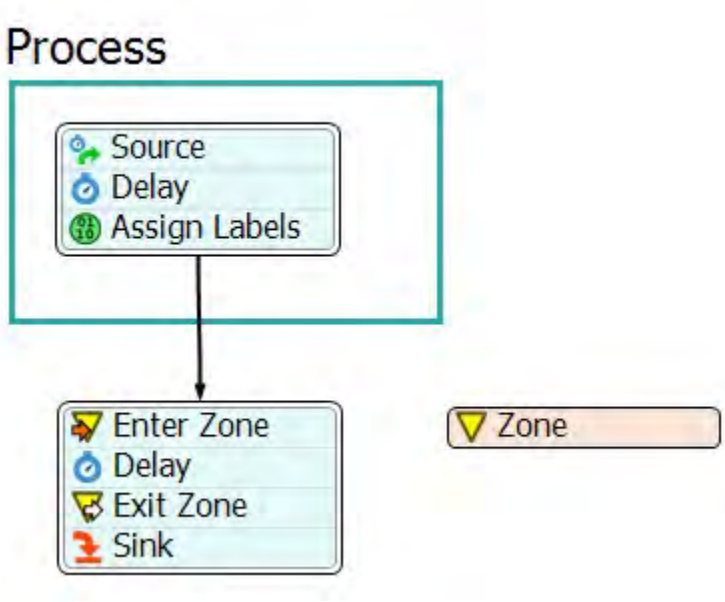
To get to this dialog box:

1. Click on an empty area somewhere in the process flow (so that nothing else is selected).
2. In Quick Properties, under the Theme menu, click the More Theme Settings button to open the Theme dialog box.

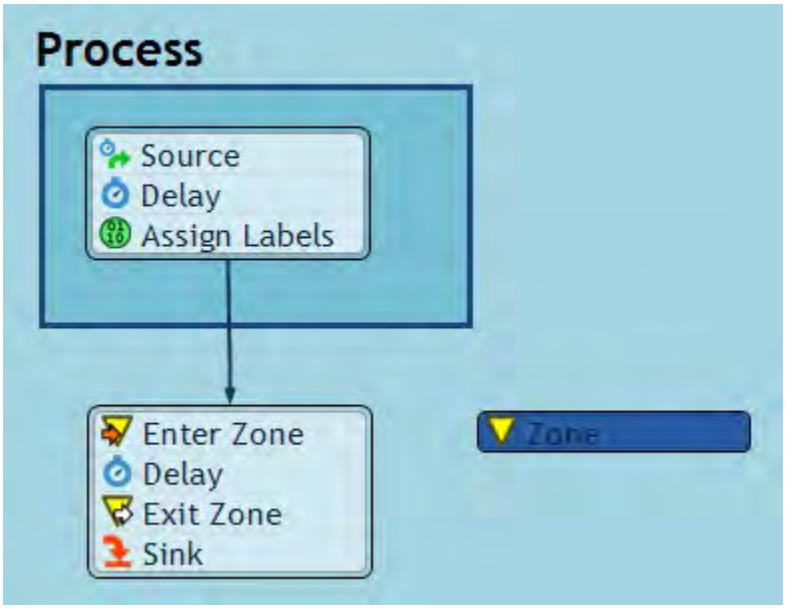
The Six Pre-Set Themes

FlexSim comes with six different pre-set themes you can use. You can use these themes to quickly implement these visual styles:

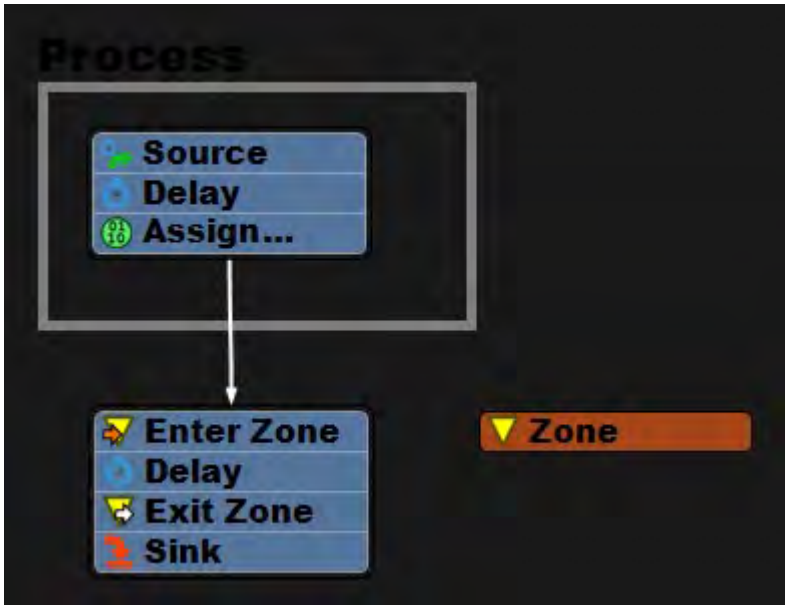
The Classic Theme



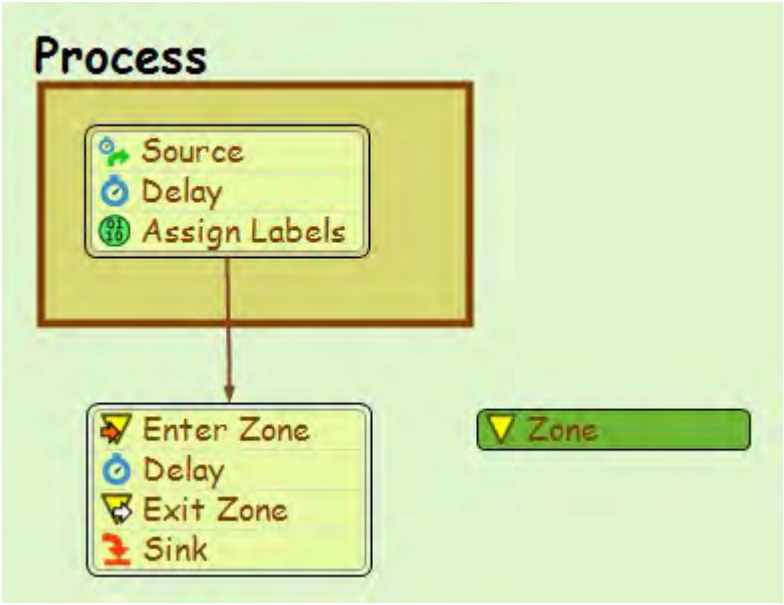
The Aqua Theme



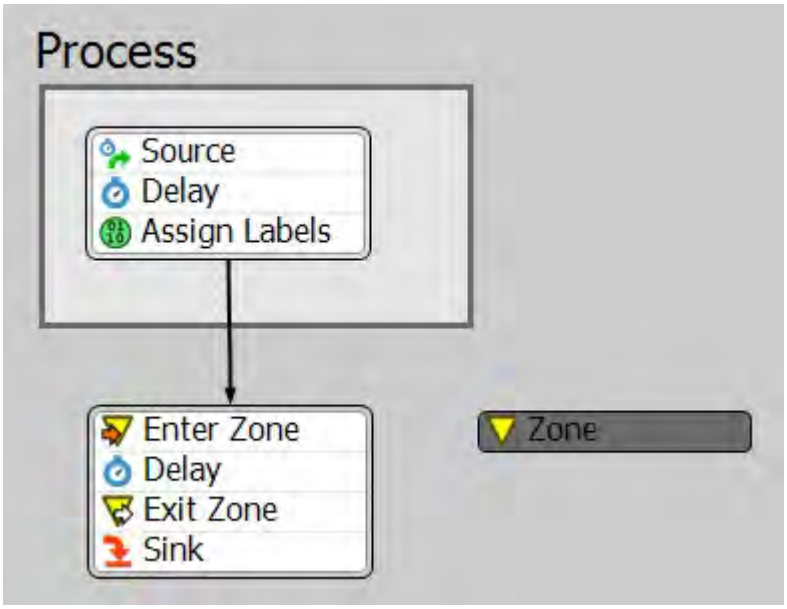
The Dusk Theme



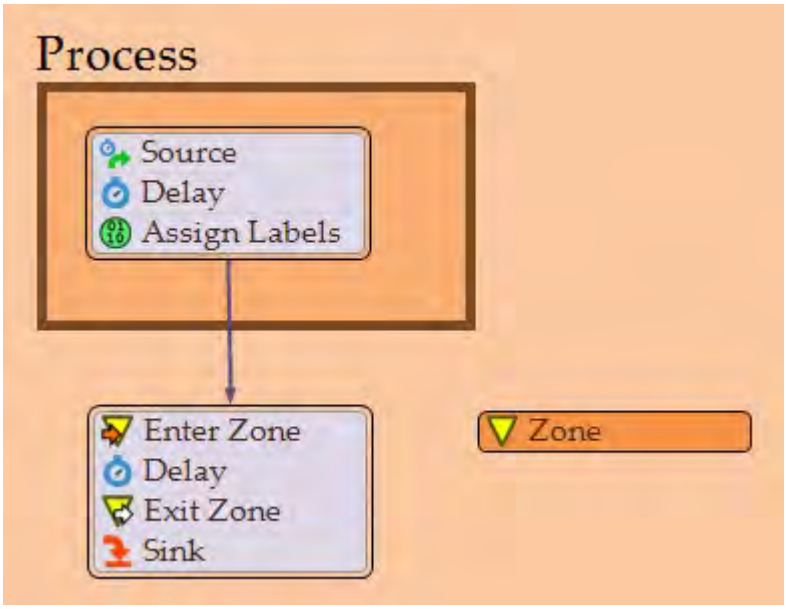
The Forest Theme



The Gray Theme



The Sunset Theme



Applying a Theme

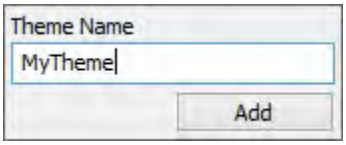
To apply a pre-set or custom theme:

1. Click on an empty area somewhere in the process flow (so that nothing else is selected).
2. In Quick Properties, select a theme from the Theme menu. The theme will automatically be applied to the current process flow.

Creating a Custom Theme

To create a custom theme:

1. Open the Theme dialog box. See Opening the Theme Dialog Box for more information.
2. If you would like to use one of the pre-set themes as a the starting point for your custom theme, select the desired theme from the Themes menu at the top of the dialog box. You'll notice that the properties for that theme will appear in all of the properties in the dialog box. However, the theme will not immediately be applied to the current process flow.
3. Click the Add button next to the Themes menu to add a new theme. The Theme Name poop-up will appear, as shown in the following image:



4. Type a descriptive name and click the Add button next to the Themes menu.
5. Change any of the visual properties that you would like to make part of this theme. (See Theme Dialog Box Properties for a complete description of each property.)
6. Click the Update button next to the Themes menu to save your changes to the currently selected theme.
7. Click the Apply Theme to Objects button to apply this theme to your current process flow. NOTE: If you only want the theme to be applied to future objects rather than the current process flow objects, do not press the Apply Theme to Objects button. Simply click the Apply button at the bottom of the dialog box instead.
8. Click the OK button to close the Themes dialog box.

Your custom theme will now appear in the Themes menu in Quick Properties and will be avaiable for any future process flows you create.

Deleting a Custom Theme

To delete a custom theme, open the Themes dialog box. Select the theme from the Themes menu and click the Delete button to delete the theme. Be aware that you can't delete any of the pre-set themes.

Theme Dialog Box Properties

The properties that are available in the Theme dialog box are shown in the following image:

For your reference, each property will be explained in the following sections:

- Themes You can use the Themes menu to create your own custom theme. See Creating a Custom Theme for more information.
- Apply Theme To Objects Button that applies the theme to previously created objects within the Process Flow. See Creating a Custom Theme for more information.
- Default Connector Properties The Default Connector Properties affect the way connections are drawn between activities. See Adding and Connecting Activities for more information about connectors.
    - Type - You can select different arrow styles from this menu. In this version of FlexSim, the two available styles are straight and Bezier. Future FlexSim versions might have more styles.
    - Width - Sets the width of the connector's line.
    - Style - You can select the connector's line style (such as a solid, dashed, or dotted line) from this menu.

- Start Cap - You can use this menu to select the visual style of the start of the connector. You can give the start of the line a shape such as a square, a circle, or another arrowhead.
- End Cap - Similar to Start Cap menu, you can use this menu to select the visual shape at the end of the connector.

- Default Text Properties The Default Text Properties affect the style of the text in a process flow. It will affect text such as the name of containers or any other display text that has been added. NOTE: Changing the text settings here will not affect the text in activities, which is affected by the Default Activity Properties.

  - Font - The font menu has 13 common fonts available. o Size - Changes the font size. o Style - You can make the text bold, italicized, or underlined.
  - Color - You can change the color of the text by using the color selector or using the Sampler button to select a color from any object in your simulation model or process flow.
  - Color Alpha - This slider changes the transparency of the text.
  - Outline - If you check the Outline checkbox, text objects will appear to have a border around them. There are a variety of properties you can use to change the border display:
    - Style - Changes the line style of the border.
    - Thickness - Changes the thickness of the border.
    - Color Selector - You can change the color of the border by using the color selector or using the Sampler button to select a color from any object in your simulation model or process flow.
    - Color Alpha - This slider changes the transparency of the border.
- Default Container Properties The Default Text Properties affect the style for containers such as flow chart objects. See Flow Chart Objects for more information about these objects.

  - Hide Contents at Magnification - You can specify a zoom level where when you zoom out further than that level the contents of this container will be hidden. See Flow Chart Objects for more information about magnification effects.
  - Fill - When the Fill checkbox is cleared, you will only be able to select the flow chart by clicking on its edge. This setting allows you to prevent clicking and selecting the background flow chart on accident. If you want to be able to click anywhere inside the flow chart object, check the Fill box. When the box is checked, the following properties will become available:

- Color - Change the color of the flow chart object's background by using the color selector. Alternatively, you can use the Sampler button to select a color from any object in your simulation model or process flow.
- Alpha - This slider changes the transparency of the flow chart object.

- Outline If you check the Outline checkbox, the border of the Flow Chart object will be visible. There are a variety of properties you can use to change the border display:
  - Single or Double - Use these options to switch to from a solid black line or a double line (which looks like it has a white stripe running through it). o Width - Sets the border's width.
  - Style - Changes the line style of the border. o Thickness - Changes the thickness of the border.
  - Color - You can change the color of the border by using the color selector or using the Sampler button to select a color from any object in your simulation model or process flow.
  - Alpha - This slider changes the transparency of the border.

- Default Activity Properties

- Set Activity Size - If you check this property you will be able to specify the size of activities. When this box is checked, the following properties become available:
    - X - Sets the default width of activities.
    - Y - Sets the default height of activities. o Activity Color - You can change the color of the activities by using the color selector or using the Sampler button  to select a color from any object in your simulation model or process flow.
- Resource Color - You can change the color of the shared assets by using the color selector or using the Sampler button  to select a color from any object in your simulation model or process flow.
- Alpha - Sets the transparency for resources.
- Font - These properties control the text on activities and resources:
    - Font - The font menu has 13 common fonts available.
    - Size - Changes the font size.
    - Style - You can make the text bold, italicized, or underlined.
    - Color - You can change the color of the text by using the color selector or using the Sampler button  to select a color from any object in your simulation model or process flow.

Troubleshooting Process Flows

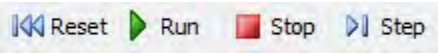
One of the main advantages of using the Process Flow module is that it is much easier to debug and troubleshoot than code. That's because you can visually see the logic of your simulation model in a process flow. This topic will discuss various tools and strategies you can use to troubleshoot your process flow.

This topic contains the following sections:

- Using the Step Button
- Tracing Tokens Visually
- Tracing Tokens Using a Tracing History
- Using 0 Second Delays
- Token Properties
- Tokens Window
- Lists

Using the Step Button

One of the best places to start when troubleshooting in process flow is to observe each step in the process flow. You can observe each step in process flow the same way you would observe it in a normal simulation model, using the Step button on the simulation control bar:



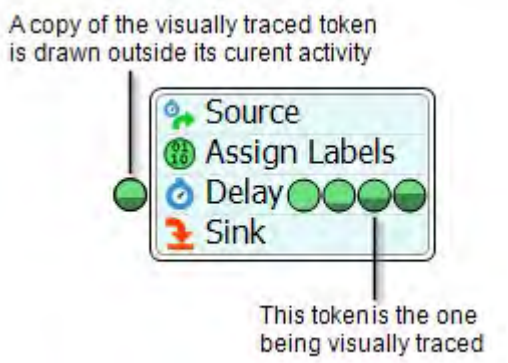
Instead of clicking the Run button, you can click the Step button to advance the simulation to the next event that will occur in the process flow model. You'll then be able to notice which tokens cause error messages or fail to do what you expected them to do.

You can also click on a specific token to look at detailed information about its properties. See Token Properties for more information about how to view a token's properties.

Sometimes if your process flow has many different tokens, it might be difficult to track a specific token through each step of the process flow. You can solve this problem by tracing a token. See Tracing Tokens Visually and Tracing Tokens Using a Tracing History for more information.

Tracing Tokens Visually

You can visually trace a token as it moves through the process flow if needed. When a token is being visually traced, a copy of the token will appear outside of the activity blocks so that you can easily track one specific token as it moves through the process flow.



To turn on visual tracing directly from the process flow:

1. Click the token you want to trace to select it.
2. In Quick Properties, check the Visually Trace Token checkbox. See Token Properties for more information.

To view a high-level overview of all the current tokens and select one to trace:

1. Click a blank space in the process flow to ensure nothing is selected.
2. In Quick Properties, click the View Tokens button to open the Tokens window.
3. Look at the Tokens list to view the names or IDs of all the tokens that are currently in the process flow. Clicking on any token will show you its detailed information. After you've determined which token you want to visually trace, make sure it is still selected in the Tokens list.
4. Near the bottom of the window check the Visually Trace Token checkbox. See Tokens Window for more information.

Tracing Tokens Using a Tracing History

It's possible to get a pretty detailed view of a token's history as it moves through a process flow. To set up the ability to view a tracing history, you need to first turn on the ability to trace history. Then you can view the trace history in a variety of ways.

Turning on the Trace History Ability

To turn on the trace history, you first need to decide which activity in the process flow will be the beginning of the trace history and which activity will be the end. For example, if you wanted to trace an entire process flow, you'd probably begin tracing on the Source activity at the beginning of the process flow, then tracing will automatically end when a token is sent to a Sink or Finish. You can turn tracing on and off at any activity.

To set up the trace history:

1. Click a blank space in the process flow to ensure nothing is selected.
2. In Quick Properties, click the View Activities button to open the Activities window.

3. Look at the Activities list to view the names of all the activities that are currently in the process flow. After you've determined which activity will be the beginning of the trace history, make sure it is still selected in the Activities list.

4. In the Tracing History group, click the Tracing Option menu and select Begin Tracing. NOTE: If you'd like to use a specific set of criteria to restrict the tokens that are traced, you can use the Requirement box to determine the criteria.

5. Now select the activity that will be the end of the tracing history in the Activities list.

6. In the Tracing History group, click the Tracing Option menu and select End Tracing. See the Activities Window for more information.

Alternatively, you can right-click any activity in a process flow to open a menu. Point to Tracing History, then select Begin Tracing or End Tracing to turn tracing on or off.

Viewing a Trace History

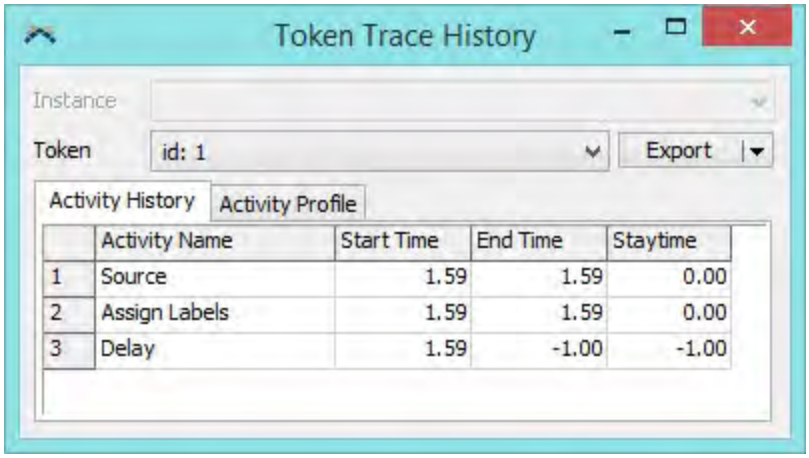
To view a trace history:

1. Click a blank space in the process flow to ensure nothing is selected.

2. In Quick Properties, click the View Tokens button to open the Tokens window.

3. Look at the Tokens list to view the names or IDs of all the tokens that are currently in the process flow. Clicking on any token will show you its detailed information. After you've determined which token for which you want to view the trace history, make sure it is still selected in the Tokens list.

4. Near the bottom of the window, click the Show Trace History button to bring up the Trace History window, an example of which is shown in the following image:



The -1 tells you the token has not yet left that activity.

You can also click the Show Trace Histories button below the list of tokens to open a Trace History window and then select the desired token from the drop down.

You can also view a token's trace history by clicking on it in the process flow view and clicing the Trace History button in Quick Properties.

When a Trace History window is open, the process flow view will display that token's history out to the left of each activity that it entered in the form *id.index*. Where id is the token id number and the index corresponds to the index in the tracing history data.

The Trace History window displays two sets of information, the Activity History and the Activity Profile.

- Activity History - This displays the order in which the token moved through the process flow as well as the times in which it entered and exited each activity.
- Activity Profile - This displays the total amount of time the token spent inside each activity. If your process flow contains any kind of loop or tokens go through the same activities multiple times throughout their life, this data will show you how long the token is spending in each of these activities.

Using 0 Second Delays

Some problems in a model may occur because of synchronous execution of activities that are triggered by some event in the model. For example, you might have multiple flowitems on a fixed resource such as a conveyor. If your process flow uses an event-listening activity such as an Event-Triggered Source or Wait for Event to determine when to release a flowitem, it's possible that the conveyor might accidentally release all the items on the conveyor at once.

This problem is the result of how FlexSim handles the logistics of event-listening activities. (See Key Concepts About Event-Listening for more information.) It often happens when a Wait For Event or EventTriggered Source is listening for a specific event. Then, when that event occurs, the event may be followed by one or more other activities that may also inadvertently cause events in the model to fire, without any delays between the original triggered event and the subsequent event-triggering logic. The logic happens so fast that it doesn't give the original event enough time to "breathe" and finish what it was doing. The kind of events that might be affected by these problems are:

- Opening inputs or output ports
- Receiving or releasing items
- Moving objects
- Destroying objects, etc.

In these cases, you can fix the problem by adding a Delay activity after the event-listening activity and setting the delay time for 0 seconds. Adding this activity will allow the original triggering event to finish what it was doing before the token moves on. (See Delay for more information about this activity.)

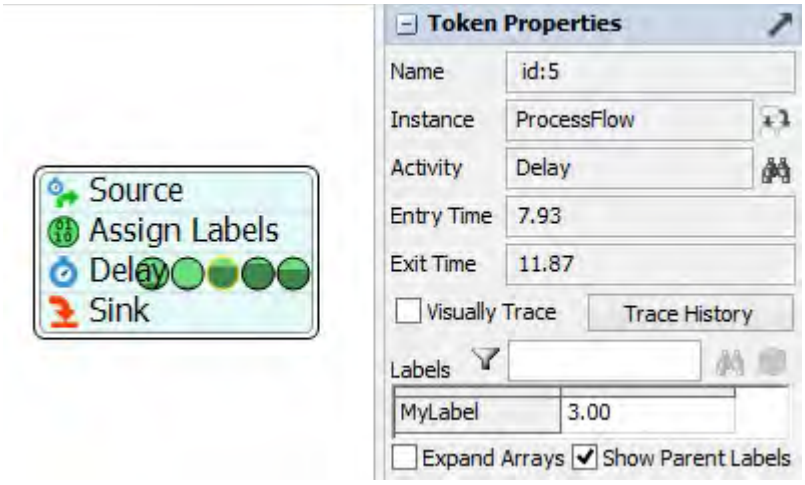
Another advantage of using 0 second delays is that by doing that it makes it easier to debug using the Step button. (See Using the Step Button for more information.) When activities are triggered by external events, process flows must force all activity execution to happen synchronously, because you may actually want to do calculations, define return values, etc. immediately before anything else happens. So unless you explicitly instruct it otherwise by adding a Delay activity with a 0 delay time, event-triggered activities will execute synchronously. This delay will allow the object that triggered the event to finish its event logic.

For more examples of this problem and possible solutions, see the Fixed Resource Process Flows Tutorial and Key Concepts About Event Listening Activities - Example 4.

Token Properties

The Token Properties are especially useful for understanding exactly what is happening to a particular token because it gives detailed information about the token. During a simulation run, you can press the

Stop button pause a simulation. Then, when you click on a specific token, the Quick Properties pane will show the Token Properties, as shown in the following image:

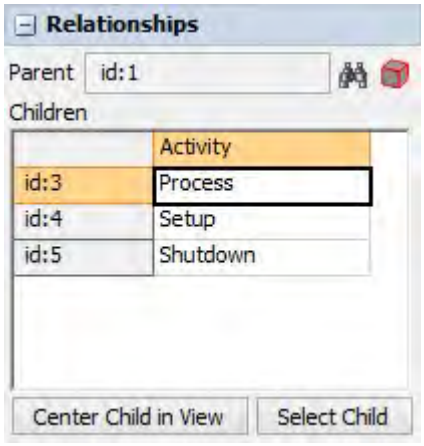


This pane has the following settings:

- Name - Displays the token's name (if the token was assigned one) and id number.
- Instance - Displays the specific instance of the process flow that this token is in. See Process Flow Instances for more information about instances.
- Activity - Displays the current activity the token is in.
- Entry Time - Displays the time on the simulation clock when the token entered the current activity.
- Exit Time - If applicable, displays the time on the simulation clock when the token will exit the current activity. Applicable activities include the Delay, Task Sequence Delay and Run Animation.
- Visually Trace - If you check this checkbox, the token will be drawn to the left of its current activity as it moves through the process flow. See Tracing Tokens Visually for more information.
- Trace History - Displays the trace history for this token. See Tracing Tokens Using a Tracing History for more information.
- Labels - Displays all of the token's labels and their current values in the form of a table.
    - - This field allows you to filter the list of labels by label name.
    - - If the selected label in the token's label table is an object or another token, this will center that object/token in its view.
    - - If the selected label in the token's label table is an object or another token, this will select the object/token.
- Expand Arrays - Affects the way labels are displayed in the table. If checked, any label that contains array data will display each entry in the array as a separate row.
- Show Parent Labels - Affects the way labels are displayed in the table. If checked, any labels owned by a parent ancestor tokens will be displayed in the table.

Token Relationships

If the selected token is a child or has children, the Relationships pane will also be displayed:

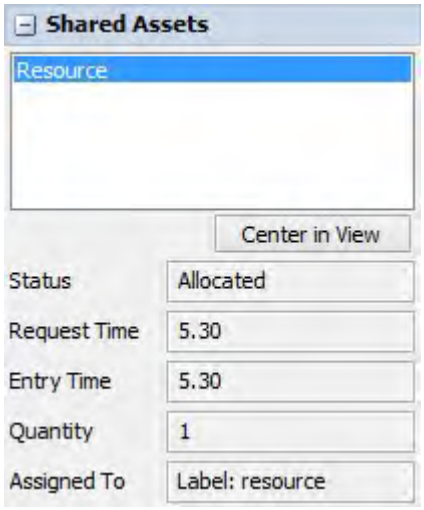


The pane has the following settings:

- Parent - If the token has a parent, the name and id number of the parent token will be displayed here.
    - o - This will center the parent token in the process flow view.
    - o - This will select the token, changing the quick properties to display the parent token's properties.
- Children - If the token has any children, they will be displayed in this table. The name and id number will be displayed in the first column and the current activity of the child token will be displayed in the Activity column.
- Center Child in View - This will center the selected child token in the process flow view.
- Select Child - This will select the child token, changing the quick properties to display the child token's properties.

Token Shared Assets

If the selected token has allocated any shared assets (acquired a resource, pushed/pulled from a list or entered a zone), the Shared Assets pane will also be displayed:

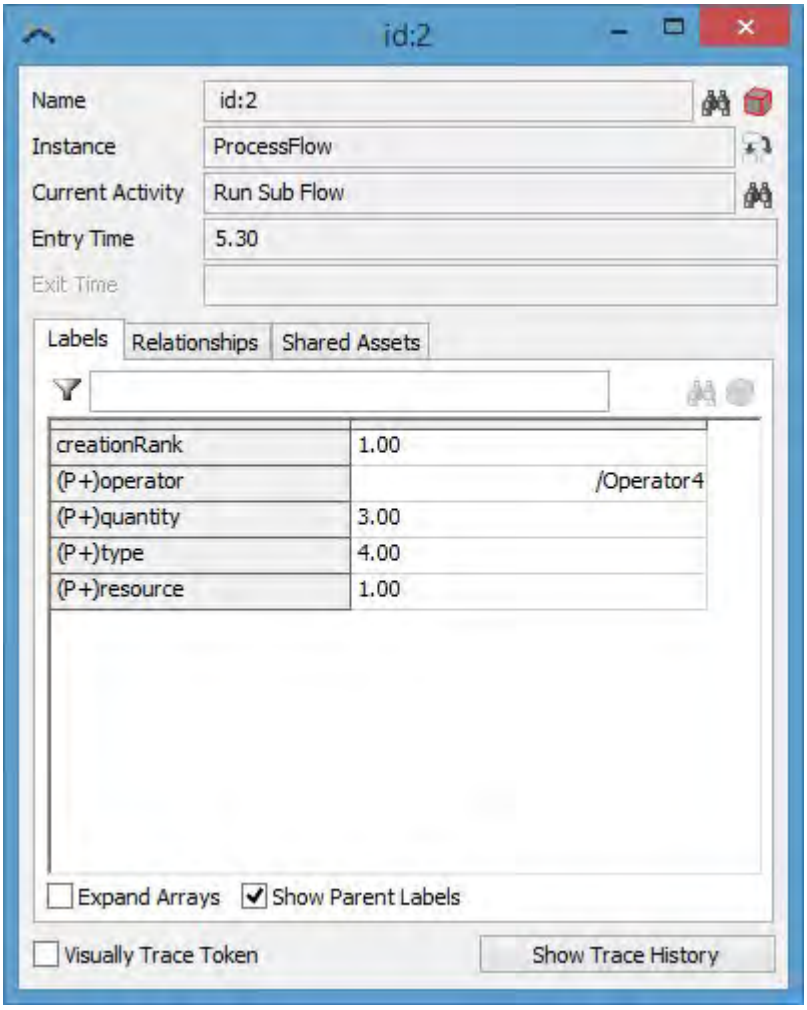


The pane has the following settings:

- Allocated List - A list of all allocated shared assets will be displayed here, by name.
- Center in View - This will center the selected shared asset in the process flow view.
- Status - Displays the status of the allocation. The following statuses are possible:
    - Requested - The token has attempted to allocated the shared asset but it is not currently available. The token is waiting for the shared asset to become available.
    - Allocated - The token has successfully allocated a shared asset.
- Request Time - The simulation time the token attempted to allocate the shared asset.
- Entry Time - The simulation time the token allocated the shared asset.
- Quantity - Only used with Resources. This is the number of requested/allocated resources.
- Assigned To - Only used with Resources. This is the label that the resource was assigned to.

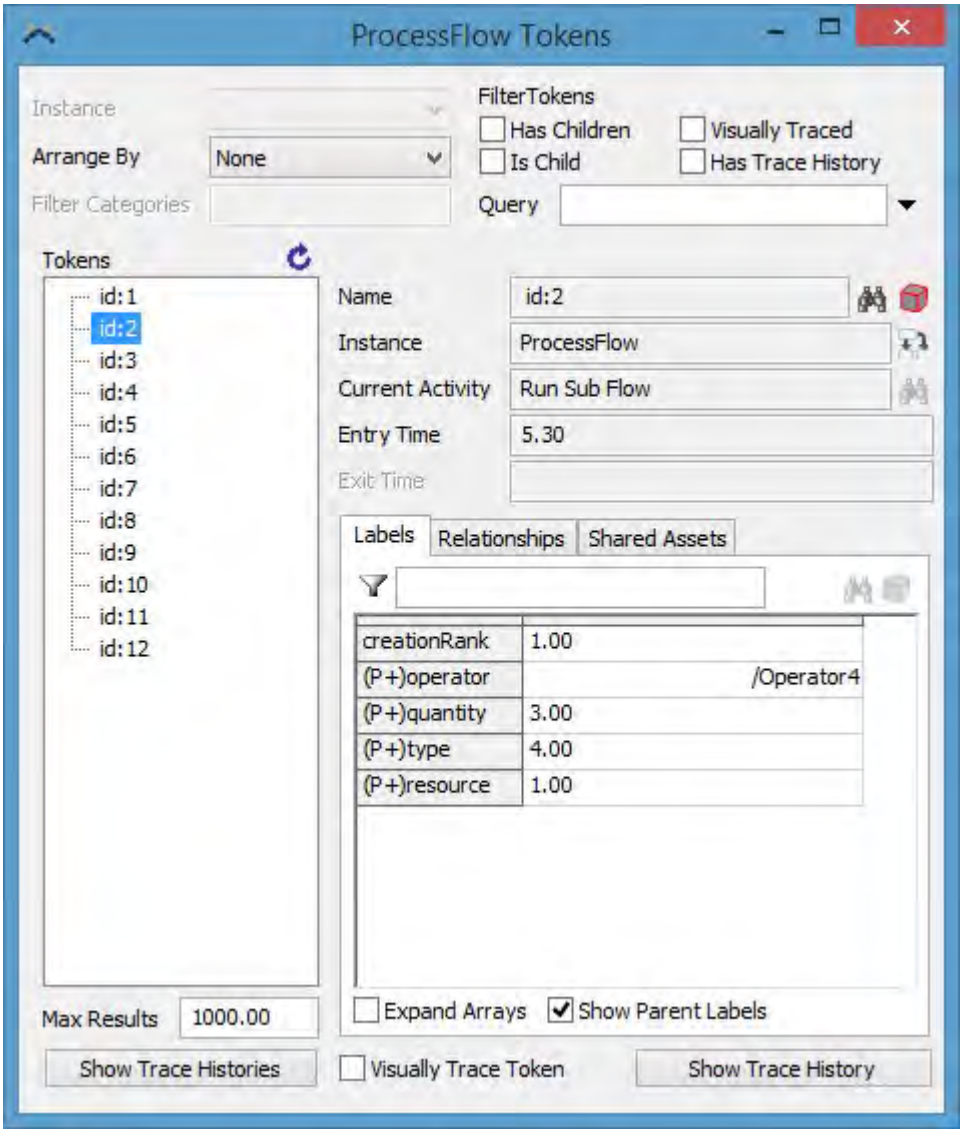
Token Properties Window

All of the above properties can also be opened in the Token Properties Window which is available either through the Pop-out button on the Token Properties pane in the quick properties, or by double clicking a token in the process flow view.



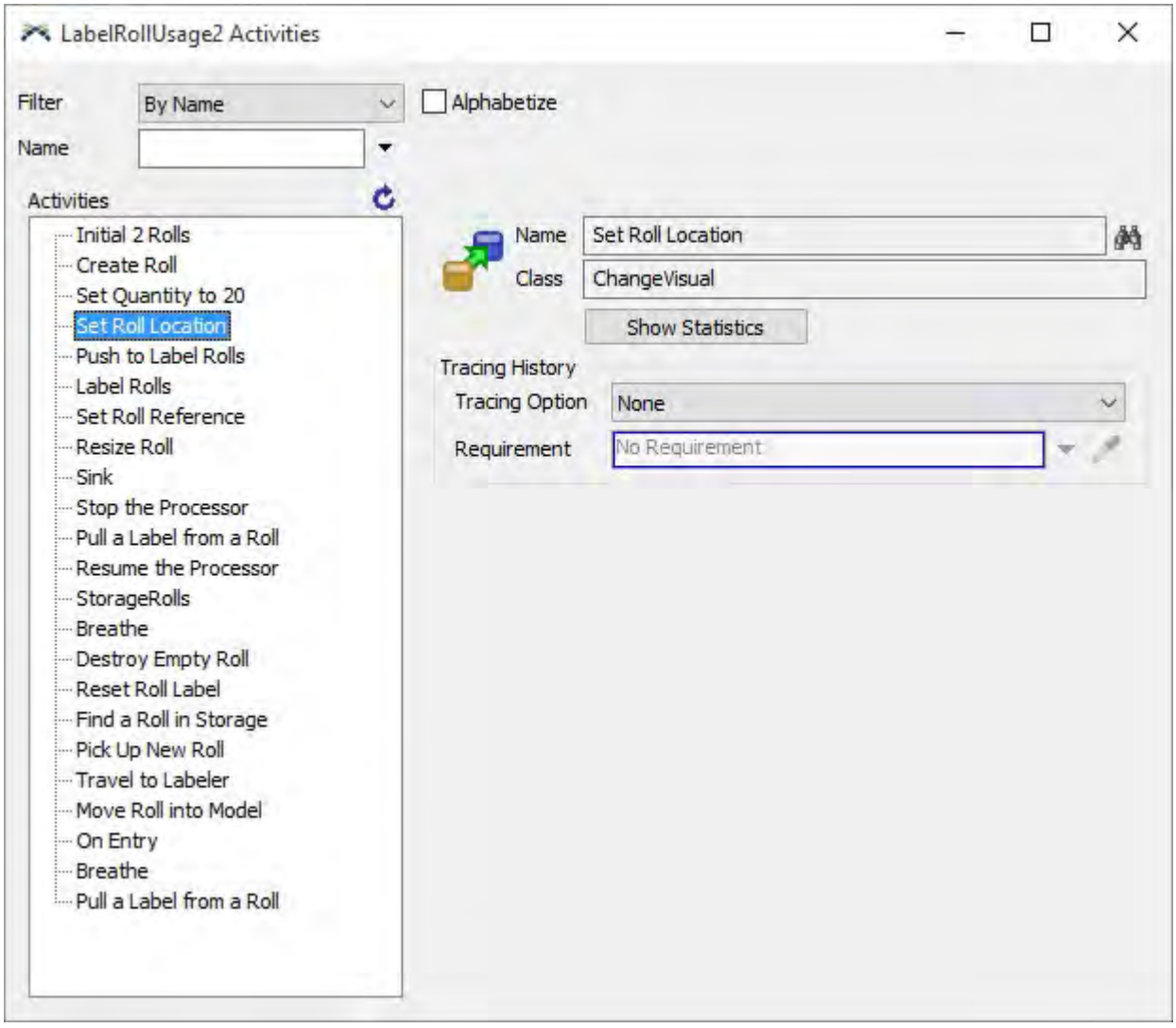
Tokens Window

The Tokens window allows you to view, edit, and manage of all the tokens that are currently in your process flow. You can open this window by clicking a blank space in your process flow (to make sure nothing else is selected). Then, in Quick Properties, click the View Tokens button to open the window.



Activities Window

The Activities window allows you to view, edit, and manage all the activities that are currently in your process flow. You can open this window by clicking a blank space in your process flow (to make sure nothing else is selected). Then, in Quick Properties, click the View Activities button to open this window.



Lists

If your Process Flow involves any interaction with Lists, you will probably need to check the entries on that List. You can view those entries and any backorders by clicking the View Entries button or View Back Orders button on the List Properties. See List Concepts for more information on using these windows.

Getting Information from a Process Flow

This chapter of the User Manual will explain how to get information from process flows. All of the methods for getting data that you have used for normal simulation models are still available (such as the Experimenter). However, the Process Flow module has some additional statistics or ways to track and display statistics that are either new or somewhat different from previous versions of FlexSim. This chapter will explain these new statistics and tools. This chapter contains the following topics:

- Creating and Using Charts - One of the typical ways to gather and display information from a process flow is to create a chart inside a dashboard to display relevant information. The process flow charts is

more feature-rich and has some additional functionality that isn't yet available in other kinds of charts. This topic explains these differences and discusses how to use these charts.

- Chart Properties - This topic is a reference page that describes the Chart Properties window in more depth. If you still have questions after reading the previous topic, this topic might help.

- Statistics in Process Flow - This topic is intended to be a reference page for all of the different possible statistics that are available in process flows. If you are unclear about what kind of information is collected by a particular statistic, you can reference it in this topic. It will also provide information about statistical values, different uses for statistics, and commands related to statistics.

- About the Zone - This topic will introduce you to the basic concepts of the Zone, which is one of the shared asset objects. After reading this section, you should have a basic idea of what a Zone is and when you might need one. Zones represent a fairly abstract concept and so the next two sections provide analogies for understanding. The final section discusses use cases for the Zone asset.

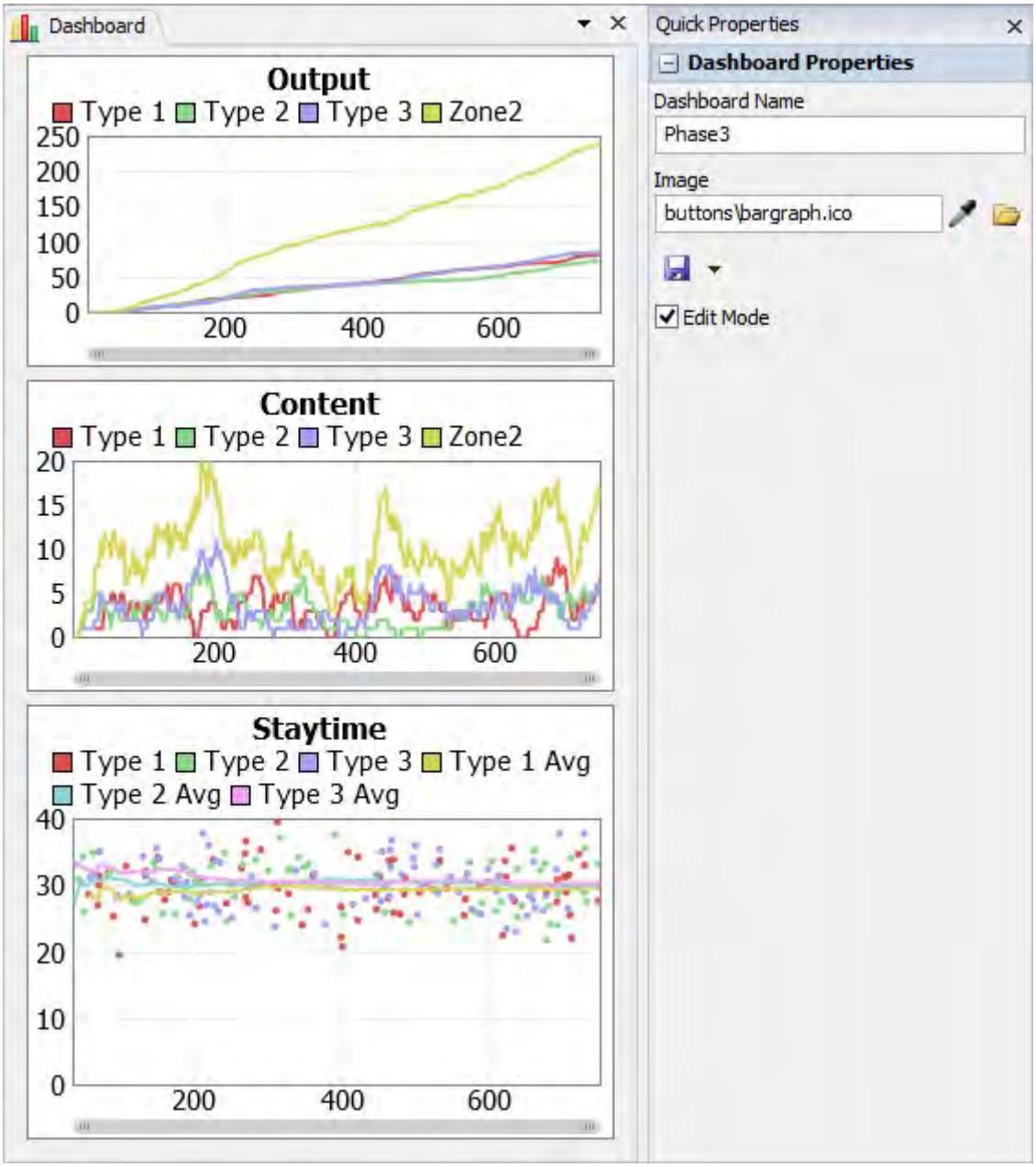
Creating and Using Charts

You can create charts inside of dashboards to help you visualize the statistics gathered in a simulation run in process flow. Although the method for making charts and dashboards is very similar to the method you'd use in a normal simulation model, the process flow charts have some added functionality:

- You can attach the objects that your chart will track in a more sophisticated way. You can now add multiple objects at a time based on their object type. You can also track objects in different instances of a process flow.

- You can easily change the type of chart you are using to display a particular statistic. For example, if you realize that a bar chart would be better than a pie chart for displaying a particular statistic, you can easily change the chart type without having to create a new chart outright.

- You can now chart the historical change of a specific statistic over time in a simulation run. In the past, if you wanted to see how a statistic changed over time, the only option was to watch it during a simulation run. Now you can plot its change over time directly in a chart.



This topic will explain how to make dashboards and charts in process flow. It will contain the following sections:

- Creating Charts and Dashboards in Process Flow
- Adding Objects and Statistics to a Chart
- Displaying Statistics in Groups
- Changing Chart Colors
- Changing the Chart Type
- Additional Properties

Creating Charts and Dashboards in Process Flow

There are two ways to make charts and dashboards in process flow:

1. Using the Toolbox and Chart Library
2. Pinning an object directly in a process flow

The following sections will explain each method.

Using the Toolbox and Chart Library

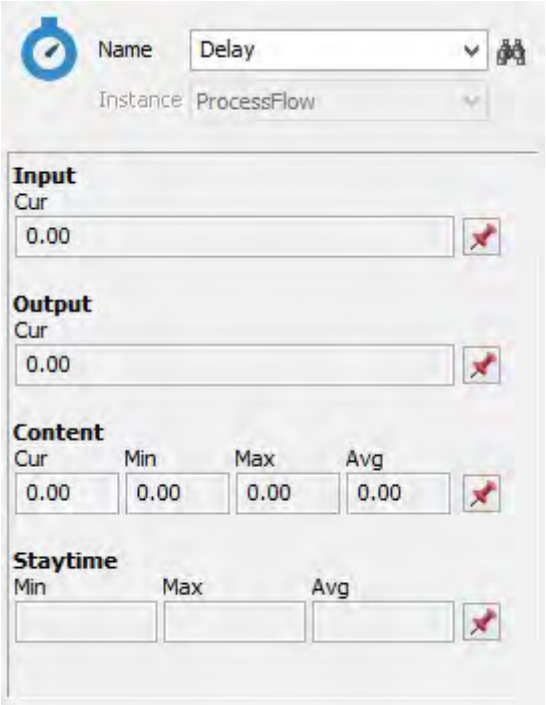
To make a chart using the Toolbox and Chart Library:

1. Open the Toolbox by clicking the Toolbox tab in the left pane (where the Library is).
2. Click the Add button to open the tools menu. Select Dashboard to create a new dashboard.
3. When the dashboard is open and active, the Library will display the charts that are available. The charts that are specifically designed for process flow are in the Process Flow group. NOTE: These charts are near the bottom of the Library, so you might have to scroll to get to them.
4. Click any chart and drag it into the dashboard to add that type of chart.
5. When you first add a chart to your dashboard, Chart Properties window will automatically open. You can use this window to add objects and statistics to the chart and adjust its visual properties. See Adding Objects and Statistics to a Chart for more information.

Pinning an Object Directly in a Process Flow

To make a chart directly from an object in a process flow:

1. With the process flow open, click the activity you want to track to select it.
2. In Quick Properties, click the Statistics button next to the Name box to open the Statistics window for that activity, such as the example shown in the following image:



3. The Statistics window displays all the statistics that are available for that particular activity. Find the statistic you are interested in and click the Pin button to open a menu.

4. If you haven't already created a dashboard, select Pin to New Dashboard from the menu. Otherwise, select the name of the dashboard to which you want to add the statistic. The dashboard will automatically create a chart for the statistic you are tracking.

5. Double-click the chart to open the Chart Properties window. You can use this window to add objects and statistics to the chart and adjust its visual properties. See Adding Objects and Statistics to a Chart for more information.

Adding Objects and Statistics to a Chart

At its most basic level, every chart needs two things in order to function correctly:

1. Object(s) - The chart needs to know which object(s) it should collect data from (in this case, process flow activities or shared assets).

2. Statistics - The chart needs to know which of the object's specific statistics it should track and display.

If you created your chart by pinning an object directly in the process flow, you've already added at least one object and statistic to the chart already. However, you might want to add additional objects and statistics to the chart. Also, if you created a chart directly from the Toolbox, you still need to add objects and statistics.

To add objects and statistics:

1. Double-click the chart to open the Chart Properties window.

2. In the Objects tab, click the Sampler button to enter sampling mode. Click an activity or shared asset in the process flow to add that object to the chart. NOTE: You should only add process flow objects to a process flow chart.

3. Repeat this process to add additional objects to the chart.

4. Now you need to add the relevant statistics to the chart. Click the Statistics tab.

5. Click the Add button to open a menu of all the different kinds of process flow statistics that are available.

6. Select the statistic you would like to track.

7. Now you need to decide which statistical value you should display. For example, the average value, the current value, etc. In the Stat Settings, click the Value menu. The following values are available:

   o Current o Minimum o Maximum o Average o Standard Deviation

8. Change the other statistical properties as needed. See Chart Properties - Statistics Tab for more information.
9. Repeat this process to add additional statistics to the chart.

10. Make sure you check the Validation icon at the bottom of the Chart Properties window to determine if the statistic you added is valid for that type of object. You want to ideally see the OK icon . The following table explains the meaning of the different icons:

Icon	Explanation
Error	None of the objects have the specified statistics. This icon also displays when there are no objects or statistics on the chart. The chart is not yet valid.
Warning	Some of the objects don't have the specified statistics. The chart is not yet valid.
OK	All of the objects have the specified statistics. The chart is valid.

See Process Flow Statistics for a detailed explanation of the meaning of each process flow statistic.

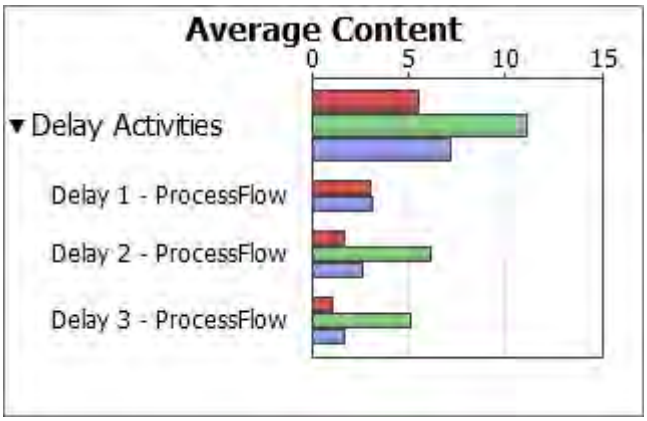
Which Statistics Can Be Charted?

Any statistic that can be queried using the getstat command can be charted with the Process Flow Chart.

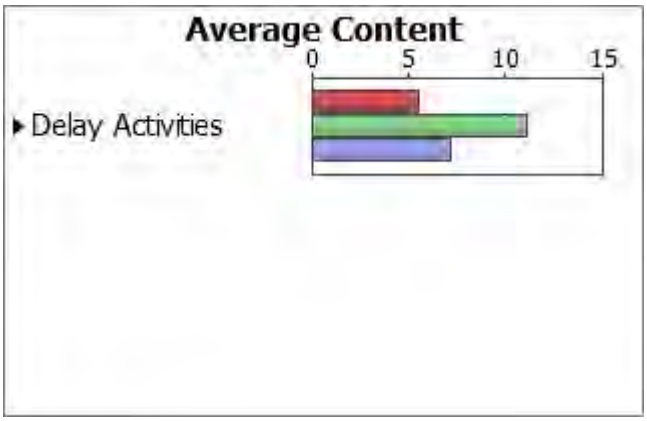
Each Entry contains all the arguments required for the getstat command, except for the statistic name. The only exception is the Standard Deviation statistic.

Displaying Statistics in Groups

If you are tracking multiple statistics for multiple activities, process flow charts can display these statistics in groups on some types of charts. For example, the following image shows the average content for three different Delay activities:



In this example, all of the Delay activities in the process flow have been grouped together. The top bar chart shows the combined average content of the three Delay activities. The three bar charts underneath the main chart show the average content of each individual Delay activity. If you press the Collapse arrow next to the top bar chart, it will hide the bar charts for the individual Delay activities, as shown in the following image:



You can group statistics by activity or by different process flow instances. See Process Flow Instances for more information about instances.

Which Chart Types Support Grouping?

Grouping can only work on bar charts and data type graphs. See Changing the Chart Type for more information.

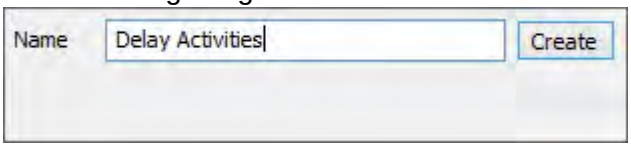
To group multiple statistics:

1. After you've added multiple objects and statistics to your chart in Chart Properties (see Adding Objects and Statistics to a Chart), click the Groups tab.

2. Each object is referred to as an *entry* in this tab. By default, all the different objects you've added will appear in the Ungrouped Entries group, as shown in the following example:

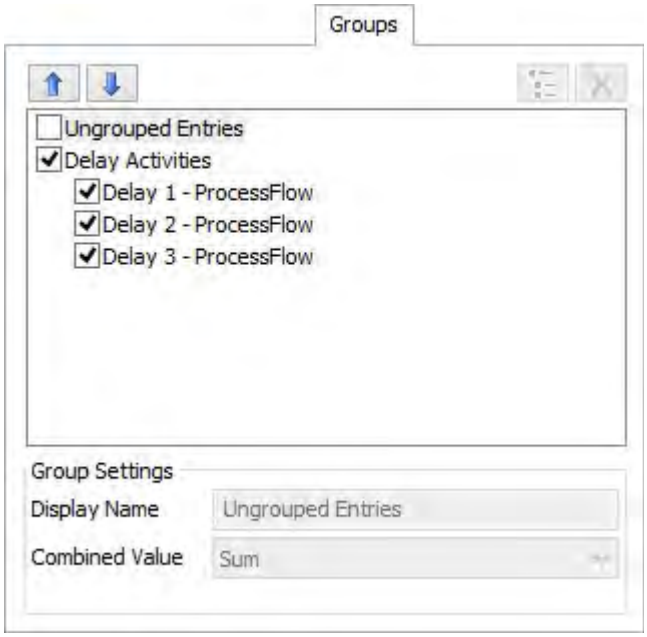
3. Click on an entry to select it.

4. Click the Groups button to open a menu. Select Add to new group to open the Group Name pop up, as shown in the following image:



5. Type an appropriate name in the Name box and click the Create button to create the new group.

6. The selected entry will now appear in the newly created group. To add more entries to the group, click the entry to select it. Click the Groups button and select Move to [Group Name] from the menu to add the entry to the new group.



7. To remove an entry from a group, click the entry to select it. Click the Groups button and select Remove from group from the menu. The entry will return to the Ungrouped Entries group.

Tracking Statistics in a Specific Instance
In the Process Flow Chart, you simply provide the objects and statistics you want on the chart. After that, the chart will automatically generate a list of all possible data sources for each statistic. Each data source is called an Entry.

For example, suppose you had a Fixed Resource process flow containing a Delay activity, with three objects attached. If you wanted to get the Input of the Delay, you would add the Delay to this list of objects on the chart, and you would add Input to the list of statistics. Then the chart would create three Entries, one for each instance of the Delay activity. You can then choose which Entries should be used by the chart. If you only wanted one of them, you would simply exclude the other two.

See Process Flow Instances for more information about instances.

Changing Chart Colors

Use the Colors tab in the Chart Properties window to change the chart's color scheme. See Chart Colors for more information.

Changing the Chart Type

To change the type of chart:

1. Click the General tab.
2. Click the Display Type menu and select the appropriate chart type from the menu.

3. Use the other properties on the tab to adjust other visual settings. See Chart Properties - The General Tab for more information about these properties.

Additional Properties

See Chart Properties for more information about the other properties that are available on various tabs in the Chart Properties window.

Chart Properties

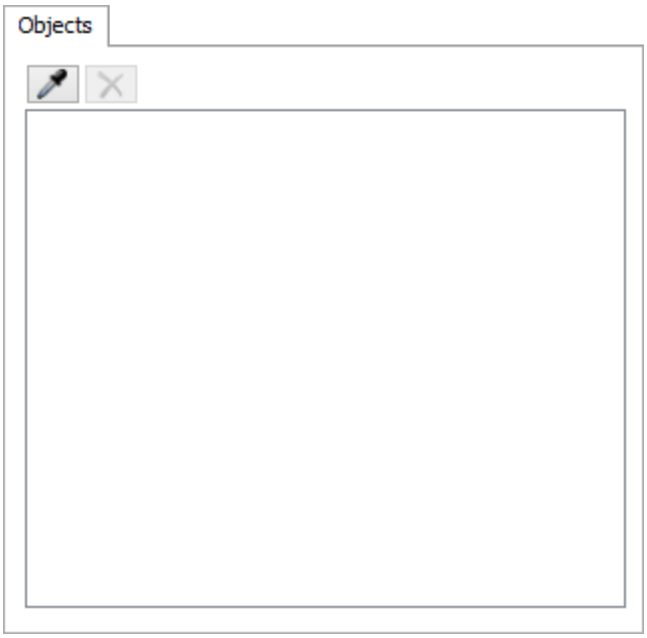
This topic is a reference page that will describe and explain each of the properties available in the Chart Properties window. See Creating and Using Charts for more information about opening and using this window.

The properties are organized by the name of the tab in which they appear in the Chart Properties window:

- Objects
- Statistics
- Runtime
- Groups
- Series
- Colors
- General

Objects

Use the Objects tab to add activities, shared assets, or lists to a Process Flow chart. Other objects cannot be added to this chart type. See Adding Objects and Statistics to a Chart for more information. The following image shows the Objects tab of the Chart Properties window:

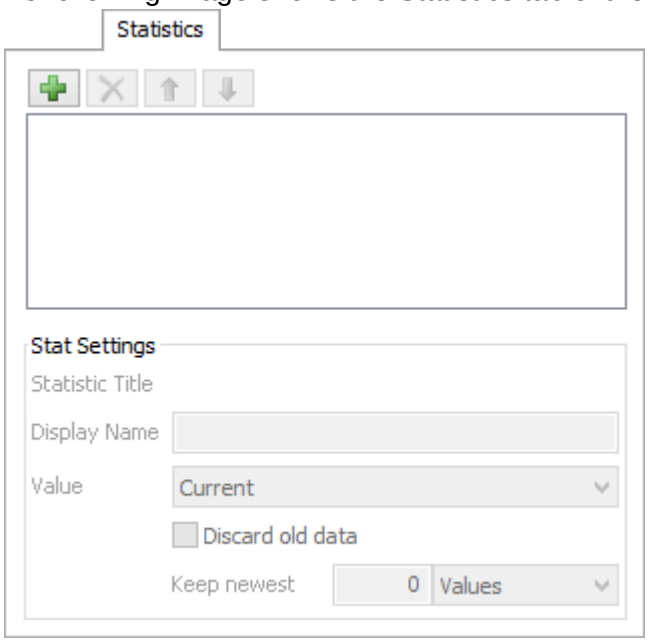


See Adding Objects and Statistics to a Chart for more information about using this tab to add an object to the chart.

Statistics

Use the Statistics tab to add the specific statistics the chart should track and display.

The following image shows the Statistics tab of the Chart Properties window:



See Adding Objects and Statistics to a Chart for more information about using this tab to add statistics to the chart.

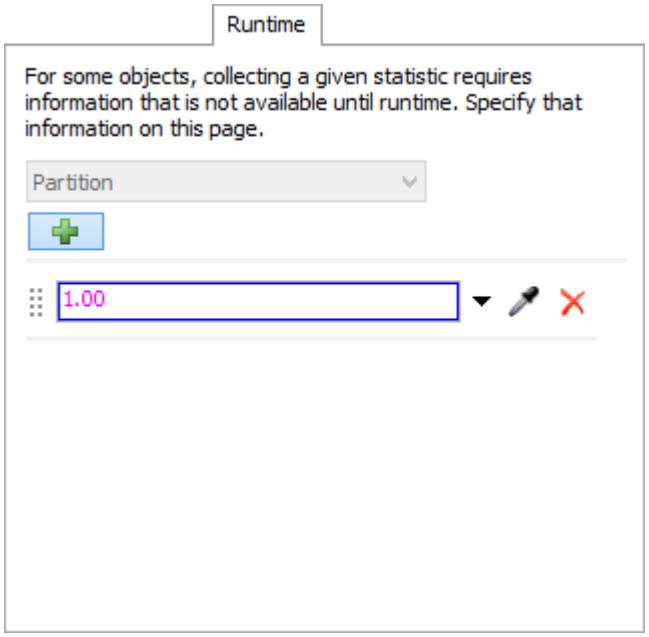
This tab has the following properties:

- Add The Add button opens a menu, displaying a complete list of all the statistics that can be displayed using the Process Flow chart.
- Up/Down The Up and Down arrows can be used to reorder the statistics. This affects the column order on Table charts, and the bar order on Bar charts. It has no effect on Time charts.
- Statistic Title The Statistic Title displays the true title of the statistic. This is the name of the statistic that is passed to the getstat command.
- Display Name The Display Name is a custom name for each statistic that displays on the chart. This affects column names in Table charts, and bar names in Bar charts. It does not affect Time charts.
- Value Use the Value list drop-down to select which the value, based on this statistic, that you want the chart to display. o Current o Minimum o Maximum o Average o Standard Deviation
- Discard old data Check this box to discard old data when calculating a derivative value (minimum, maximum, average, standard deviation).
- Keep newest Use these two fields to specify how long of a history to keep. For example, if you want to keep only the last 10 values for an average, you would enter a 10 in the field, and select Values from the list drop-down. If you want to keep the last hour of data, you would enter a 1 in the field, and select Hours from the list drop-down.

Runtime

Some statistics may not exist until the model runs. This true for List partitions, which are created on demand. The Runtime tab allows you to specify which partitions (by partition ID) you want to chart data for. If you don't have a List object on the chart, this page is irrelevant.

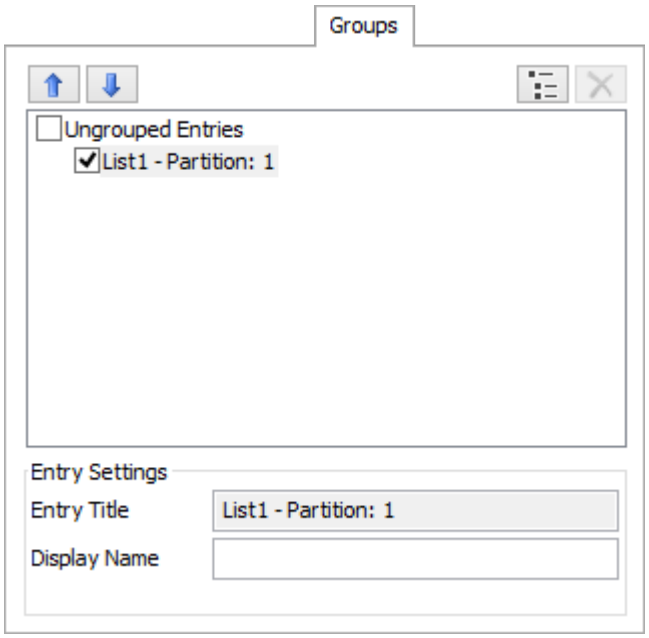
The following image shows the Runtime page of the Chart Properties window:



To add a List partition to the chart, select one of the List statistics in the Statistics page. Then click the Add button to add a field. In that field, enter in the partition ID, usually a number or a string.

Groups

Use the Groups tab to group statistics by activities or process flow instances. The following image shows the Groups tab of the Chart Properties window:



See Displaying Statistics in Groups for information about how to use the Groups tab.

Group Tab Availability

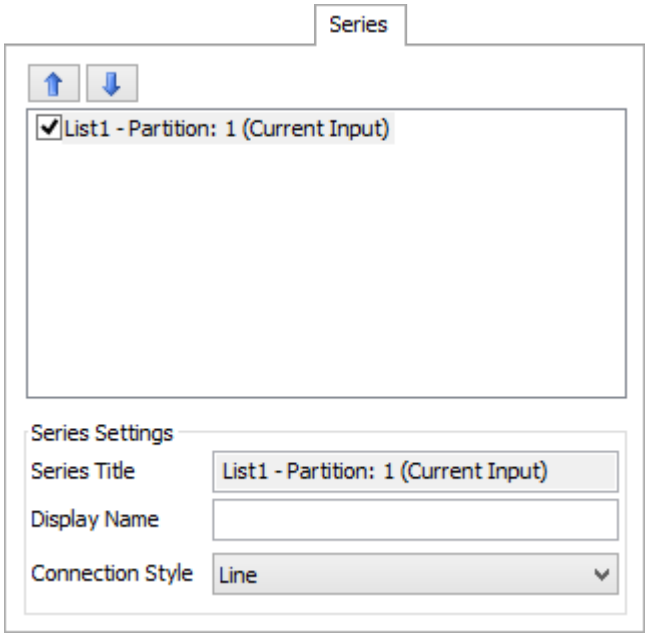
The Group tab is only present when the chart type is a Bar Chart or a Data Table. The properties on this page are explained in the following sections:

- Up/Down The Up and Down buttons move the selected item (either a group or an Entry) up or down, respectively.
- Group The Group button is used to move a selected Entry into a new or existing group.
- Entry List The Entry List shows a list of all Entries, and which group they are in. Each Group and Entry have a checkbox next to their name. If an Entry is checked, it will be included on the chart. If a Group is checked, then the chart will display the entries as members of the group. Otherwise, the entries will be included on the chart individually.
- Entry Title The Entry Title field is visible if you have selected an Entry. It displays the auto-generated name of the Entry, which describes how the data for each statistic is being retrieved.
- Display Name Both groups and Entries have a Display Name. This is the name that appears on a Bar Chart or Data Table. For Entries, if the Display Name is blank, the auto-generated name is used.
- Combined Value The Combined Value field only appears when you have selected a group. This determines how the statistics for each member of the group are combined for the group statistic. The only options are Sum and Average.

Series

Use the Series tab to manage which data series appear on your chart. It shows a list of every valid Entry/Statistic combination, which is called a Series.

The following image shows the Series tab of the Chart Properties window:



Series Tab Availability

The Series tab is only present when the chart type is a Time Chart. The properties on this page are explained in the following sections:

- Up/Down The Up and Down buttons move the selected Series up or down, respectively.
- Series List The Series List shows the plottable data for the chart. A Series can be excluded from the chart by unchecking the box next to its name.
- Series Title The Series Title is the name of the Entry and the Statistic that were used to create the Series.

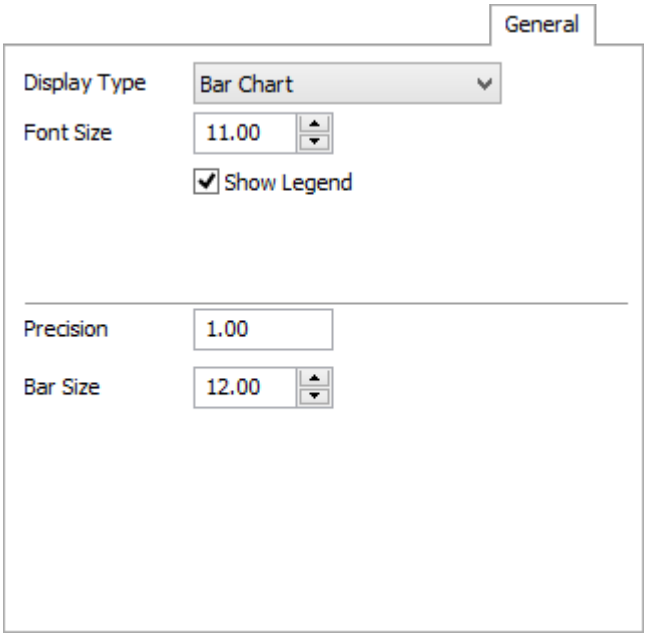
- Display Name The Display Name is shown on the graph. If it is blank, the Series Title is used instead.
- Connection Style The Connection Style determines how data points in a Series are connected on the graph.

Colors

Use the Colors tab in the Chart Properties window to change the chart's color scheme. See Chart Colors for more information.

General

Use the General tab to change the chart type and adjust other general display settings. The following image shows the General tab of the Chart Properties window:



The properties on this page are explained in the following sections:

- Display Type The Display Type drop-down list box allows you to change the type of chart being displayed.
- Font Size The Font Size field allows you to change the size of font used to draw the chart.
- Show Legend The Show Legend checkbox controls whether or not the chart displays a legend.
- Precision The Precision field is only available for Bar Charts and Data Tables. You can use this field to specify the precision the chart will use to display data.
- Bar Size The Bar Size field is only available for Bar Charts. You can use this field to specify the size of the bars.
- Time Scale The Time Scale field is only available for Time Charts. You can use this field to scale the x axis by the specified amount.
- X Axis Title The X Axis Title is only available for Time Charts. You can use this field to set the name of the x axis on the chart.

- Y Axis Title The Y Axis Title is only available for Time Charts. You can use this field to set the name of the y axis on the chart.

Process Flow Statistics

The following topic is intended to be a reference page for all of the different possible statistics that are available in process flows. If you are unclear about what kind of information is collected by a particular statistic, you can reference it in this topic. It will also provide information about statistical values, different uses for statistics, and commands related to statistics. This topic contains the following sections:

- Activity Statistics
- Shared Asset Statistics
- About Statistic Values and Types
- Statistics as Events
- Commands

Activity Statistics

All activities in process flow keep the following two statistics:

- Input - The number of tokens that have entered an activity
- Output - The number of tokens that have exited an activity

In addition, any activity that might cause a token to wait keeps the following two statistics:

- Content - The number of tokens currently in an activity
- Staytime - The amount of time each token spends in the activity

The Batch activity keeps many additional statistics. You can read about these statistics on the Batch Statistics page.

Shared Asset Statistics

Shared assets (the Resource, List, and Zone objects) keep a minimum of 8 statistics, as described below:

- Input - The number of tokens that have begun using the asset
- Output - The number of tokens that have finished using the asset
- Content - The number of tokens currently using the asset
- Staytime - The amount of time each token spends using the asset
- RequestInput - The number of tokens that requested to use the asset
- RequestOutput - The number of tokens that requested to use the asset, and have since either used the asset or failed
- RequestContent - The number of tokens waiting to use the asset
- RequestStaytime - The amount of time a token spends waiting to use the asset

Using Shared Assets

What it means to "use the asset" might vary depending on the type of shared asset it is. See About Shared Assets for more information.

You can find more details about these statistics, along with information about any additional statistics, on each Shared Asset's reference page.

About Statistic Values and Types

Each statistic provides access to four values:

- Current - The current value of the statistic
- Minimum - The lowest recorded value of the statistic
- Maximum - The highest recorded value of the statistic
- Average - The average value of the statistic

In addition, each statistic can be classified as one of three types:

- Incremental - These statistics always increase. Any statistic with *Input* or *Output* in its name is an incremental statistic. Because incremental statistics only increase, the minimum, maximum, and average values are meaningless. The current value can also be called the *total* value.

- Level - These statistics increase and decrease over time, like the level of fluid in a tank. Any statistic with *Content* in its name is a level statistic. Level statistics provide meaningful current, minimum, maximum, and average values; the average is time-weighted.

- Stream - These statistics represent values obtained from individual objects. Staytime, for example, is updated whenever a token leaves an activity. The current staytime is simply the last token's staytime, and so is not usually helpful. Stream statistics do provide helpful minimum, maximum, and timeweighted average values.

Statistics as Events

In process flow, you can listen to statistics in addition to listening to events. These events are called Value Change Events, and are fired whenever the value of the statistic changes. See Event Listening Activities and Event Types and Related Properties for more information.

Commands

You can use the `getstat` command to get data from an activity or shared asset in FlexScript. This command requires an object, a statistic name, and a value flag. The following example shows how to use this command:

```
// This is the code header given by most process flow objects.
// The comments are added here for clarification.
```

```
treenode current = param(1); // This is the instance, meaning the object that owns the Process Flow treenode activity
= param(2); // This is the activity or shared asset treenode token = param(3); treenode processFlow =
ownerobject(activity);


// Here is where you would put your code. The following code gets the average content double avgContent = getstat(
            // getstat requires between 4 and 6 parameters:          activity,      // 1. the object (shared asset or
activity) that has the statistic
                  "Content",     // 2. the name of the statistic
    STAT_AVERAGE,   // 3. the desired value from the statistic (could also be STAT_CURRENT, STAT_MIN, or
STAT_MAX)
                                                  // 4. an optional parameter the instance object (usually current)
            // 5. an additional parameter to help resolve which data to get          // 6. an additional parameter
                                    to help resolve which data to get
);
```

For example, to get the current content of a Delay activity, you could use the following command: `getstat(delayActivity, "Content", STAT_CURRENT)`

To get the output of any Process Flow activity, you could use the following:
`getstat(activity, "Input", STAT_CURRENT)`

All statistics accessible through the `getstat` command use Tracked Variables, so you can access the current, minimum, maximum, and average values for any of the statistics. You specify which value you want using the third parameter, the flag. To get the average staytime of an activity, for example, you could use the following:
`getstat(activity, "Staytime", STAT_AVERAGE)`

Notice that the statistic name is very important. The name must match the name of a statistic kept by the object. Source activities, for example, do not keep a "Content" statistic; the `getstat` command will return 0 in this case.

Statistics for Instanced Process Flows

If you want to get statistics for an activity in a Task Executor or Fixed Resource Process Flow, then you have to give a fourth parameter to the `getstat` command. This fourth parameter is the name of the instance object. For example, if Processor3 is attached to a Process Flow, and you want to get a statistic from that flow, then you would need something like the following: `getstat(activity, "Output", STAT_CURRENT, "Processor3")`

For general process flows, or for process flows with only one instance, this parameter is ignored.

List Statistics

The List activity keeps statistics for the list as a whole:

- Input - the total number of entries that have been pushed on the list
- Output - the total number of entries that have been pulled from the list
- Content - the total number of entries currently on the list
- Staytime - the staytime for entries on the list
- BackOrderInput - the total number of back orders that have been created by the list
- BackOrderOutput - the total number of back orders that have been fulfilled
- BackOrderContent - the current number of back orders on the list
- BackOrderStaytime - the amount of time each back order stays on the list

These same statistics are kept for each partition on the list. Simply prefix the word "Partition" to the statistic name.

For example, these two lines of code get the number of entries on a list, and on a specific partition:

```
getstat(globallist("list"), "Content", STAT_CURRENT); getstat(globallist("list"),
"PartitionContent", STAT_CURRENT, 0, 1);
```

The fourth parameter, the 0, is the instance object parameter, ingored for the general process flow case. The fifth is the partition ID. The value returned is the statistic for that specific partition.

Please note that when a partition is empty, the list destroys it by default. This also clears the statistics for that partition. You can avoid this by checking the "Keep Dead Partitions" box in the List Properties window. Be aware that for large numbers of partitions, using this option can use a lot of memory.

Zone Statistics

The Zone keeps the statics kept by the other shared assets. However, it also keeps those statistics for each subset. For example, if you have a subset within a zone called "SmallItems", you can get the content of that subset as follows:

```
getstat(zone, "SubsetContent", STAT_CURRENT, 0, "SmallItems")
```

Again, the fourth parameter is the instance parameter, which is ignored in general Process Flows. The fifth is the name of the subset for which you want the statistic.

You can also get the data of a subset calculation. Simply provide the name of the desired calculation as the sixth parameter:

```
getstat(zone, "CalculationInput", STAT_CURRENT, 0, "SmallItems", "TotalWeight")
```

About Zones

This topic will introduce you to the basic concepts of the Zone, which is one of the shared asset objects. After reading this section, you should have a basic idea of what a Zone is and when you might need one. Zones represent a fairly abstract concept and so the next two sections provide analogies for understanding. The final section discusses use cases for the Zone asset. This topic contains the following sections:

- Purpose of the Zone
- Collecting Statistics
- Restricting Access
- Zone Subsets

Purpose of the Zone

You can use a Zone for two reasons:

1. To collect statistics across multiple activities
2. To then use those statistics to restrict access to those activities

A real-world example of uses for the Zone might help to explain these two functions. Imagine that you wanted to use the process flow module to simulate a self-serve copy center. Perhaps you have a group of color-copy machines and a group of black-and-white copy machines. Customers who come to the copy center might want to make either color copies or black-and-white copies. However, there is a limit to how many machines can be in use at the same time. When either the color-copy or black-and-white machines are all in use, the Zone can restrict access to the machines. In other words, the customers will have to wait in line until the kind of machine they need becomes available. Using a Zone makes it possible to simulate this kind of behavior. You could possibly create one Zone for the color-copy machines and another for the black-and-white machines.

Imagine also that you want to collect data about things such as:

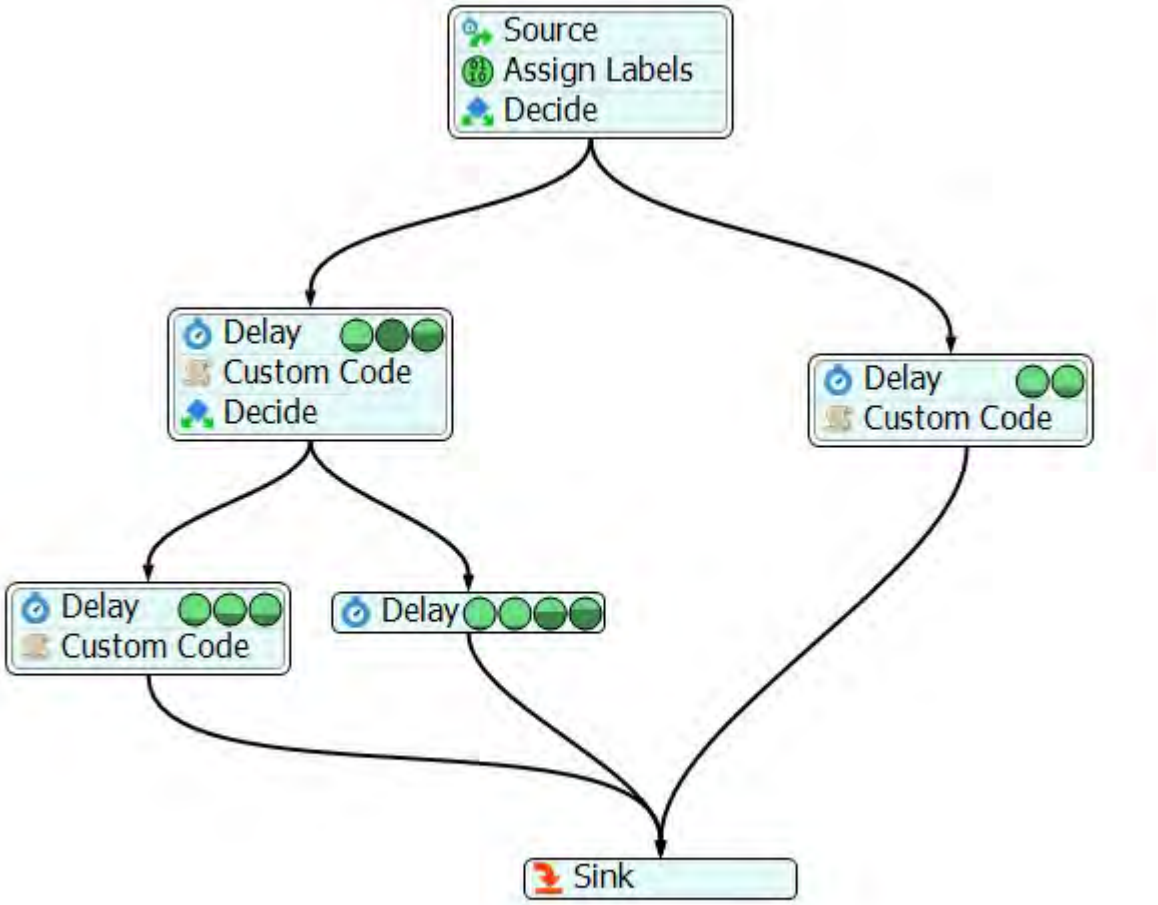
- The average number of color or black-and-white copy machines that are used in a given period of time
- The average wait time for customers who are waiting to use either color or black-and-white copy machines
- The average amount of time it takes for customers to complete their copies

Zones are capable of collecting this kind of statistical data as well.

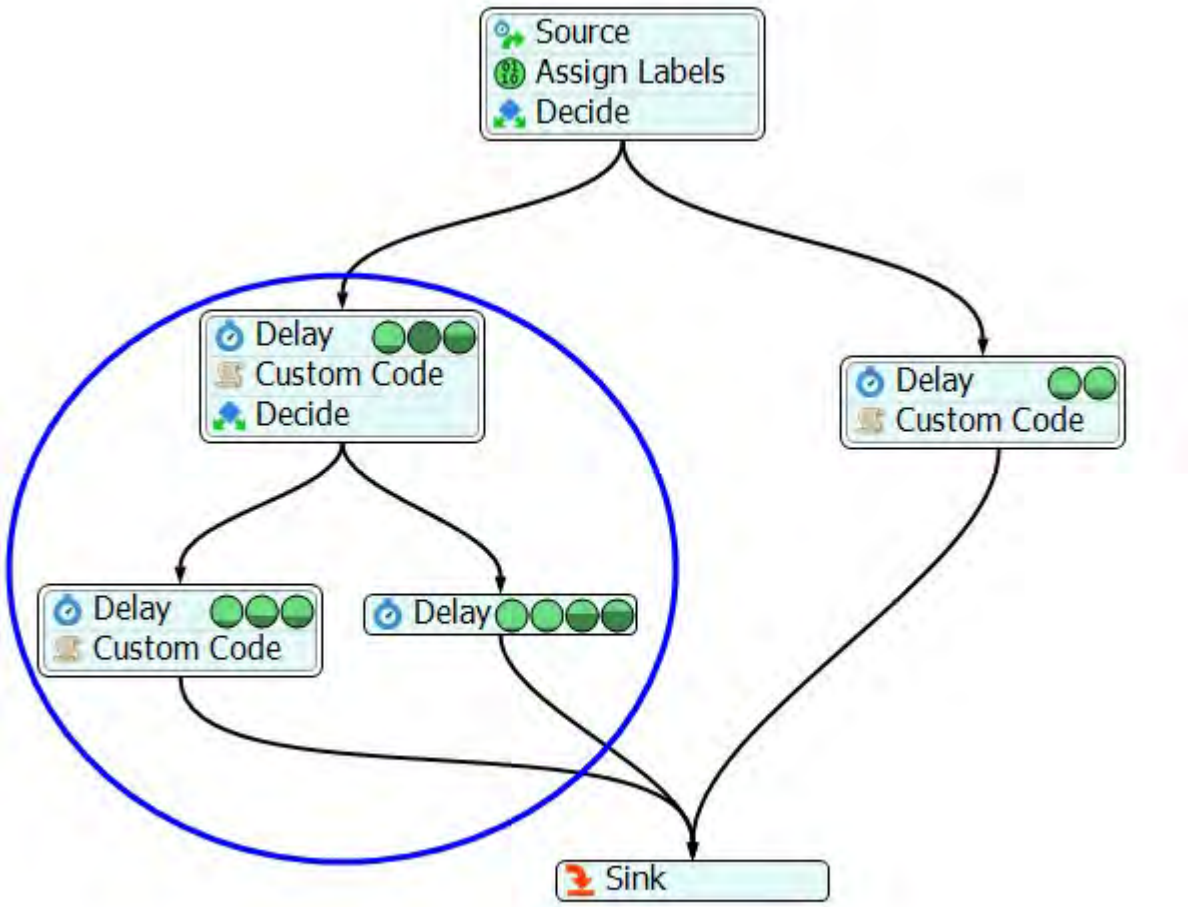
The following two sections will explain how each of these two functions (collecting statistics and restricting access) will work in an actual process flow.

Collecting Statistics

A Zone can be used to collect statistics for an area of your process flow. For example, suppose you made the process flow shown in the following image:



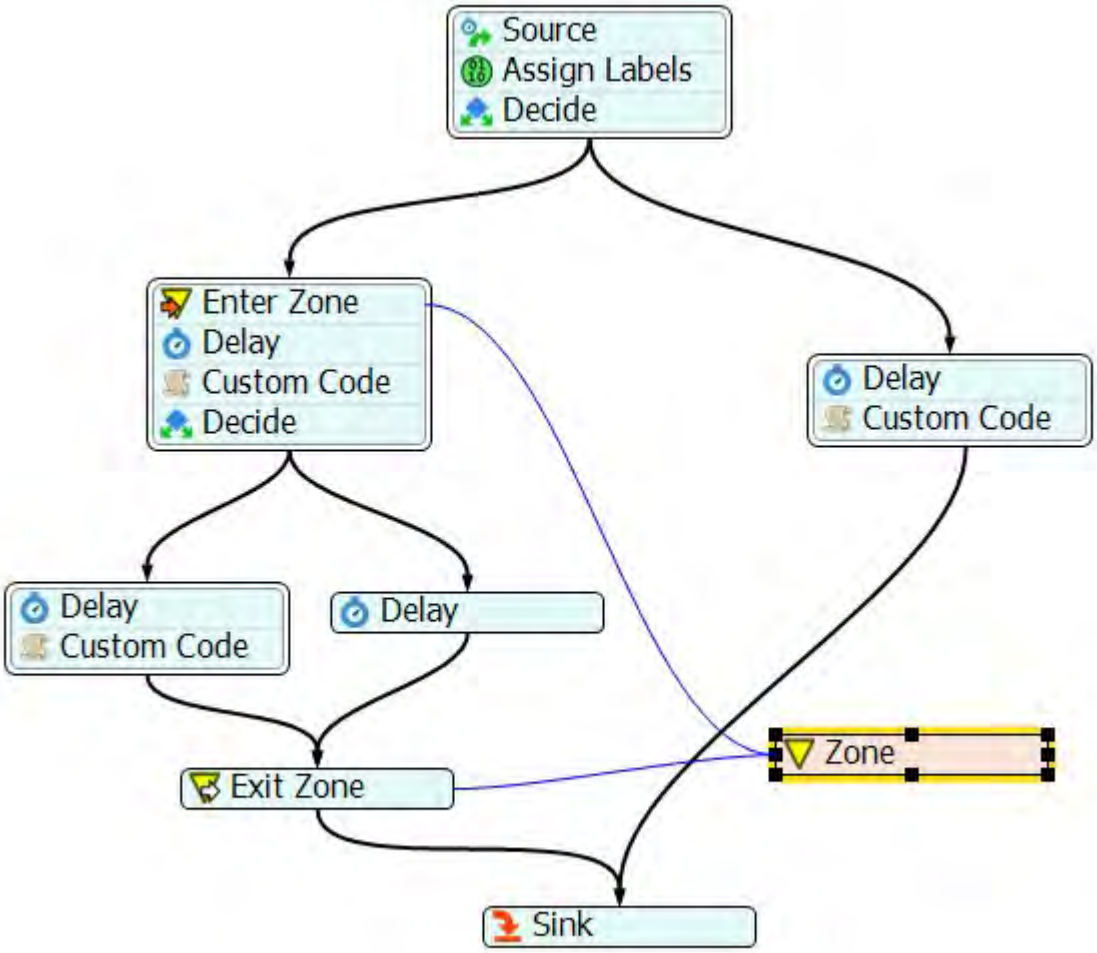
Suppose that you want to know statistics about just the left portion of the process flow, as shown in the following image:



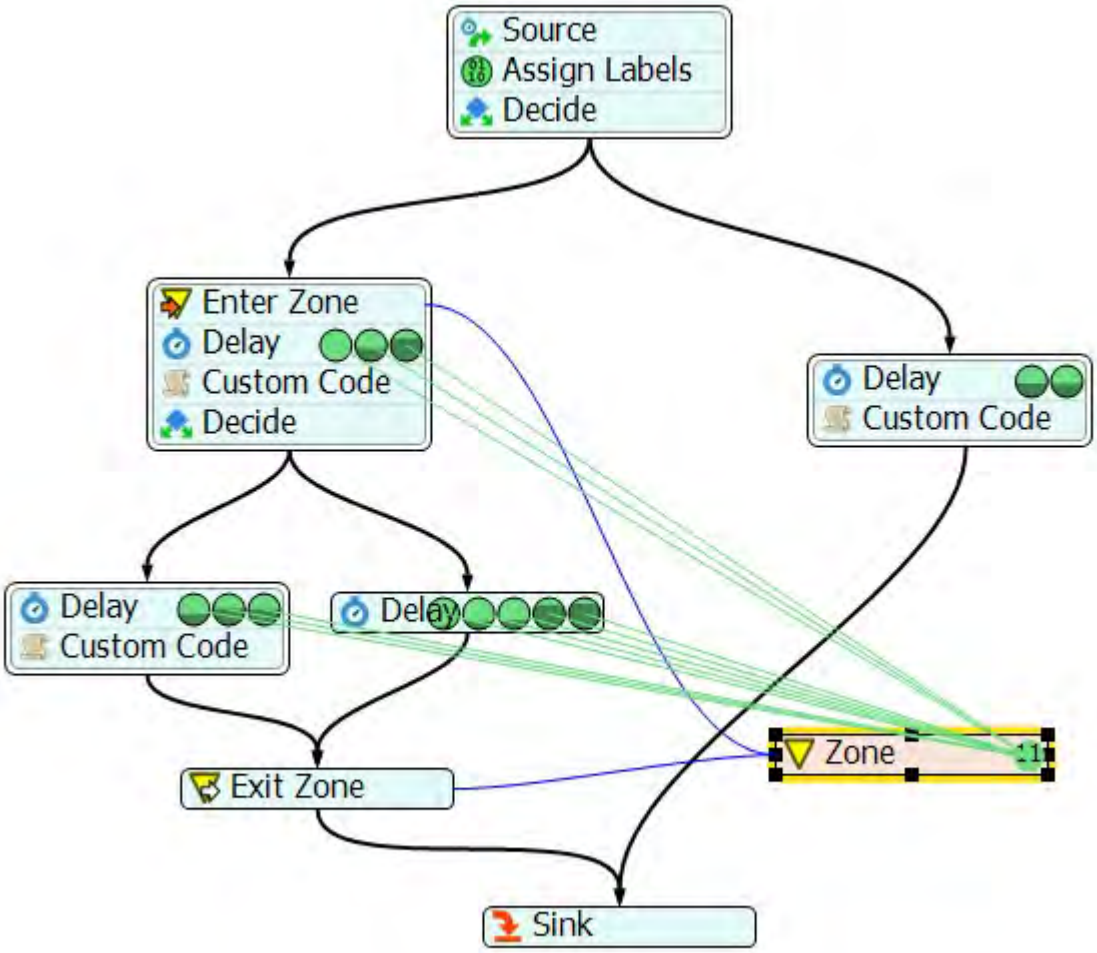
Specifically, say you wanted to know the answer to the following questions:

- What is the maximum content of the left side?
- What is the average content of the left side?
- How long, on average, does it take tokens to get through the left side?

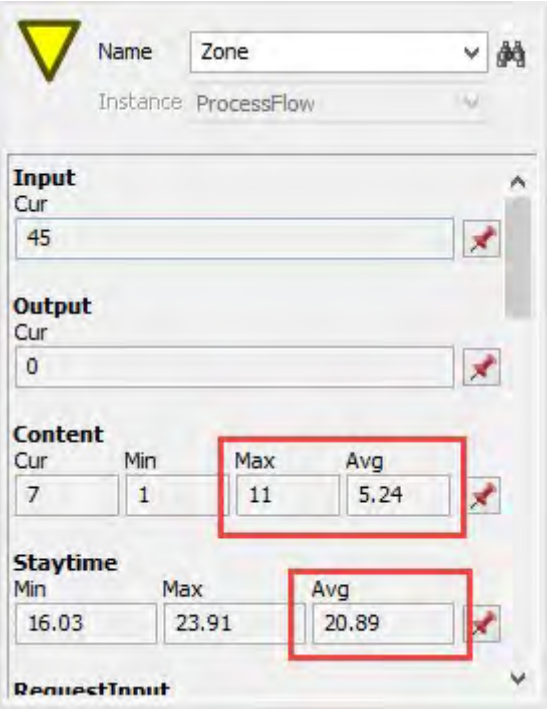
A Zone can be used to answer these questions. In order to create the statistical boundary shown in the preceding image, you would add an Enter Zone activity and an Exit Zone activity, along with a Zone, as shown in the following image:



The Zone will automatically record the input, output, content and staytime for all tokens that enter and exit the Zone. The following image shows the process flow during a simulation run, with many activities in the Zone.



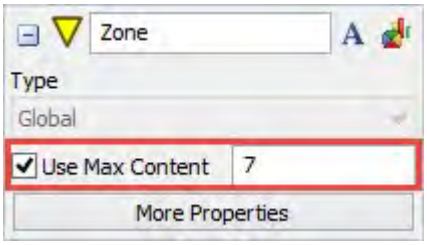
The next image shows the Zone's Statistics Window, with the maximum content, average content and the average staytime highlighted.



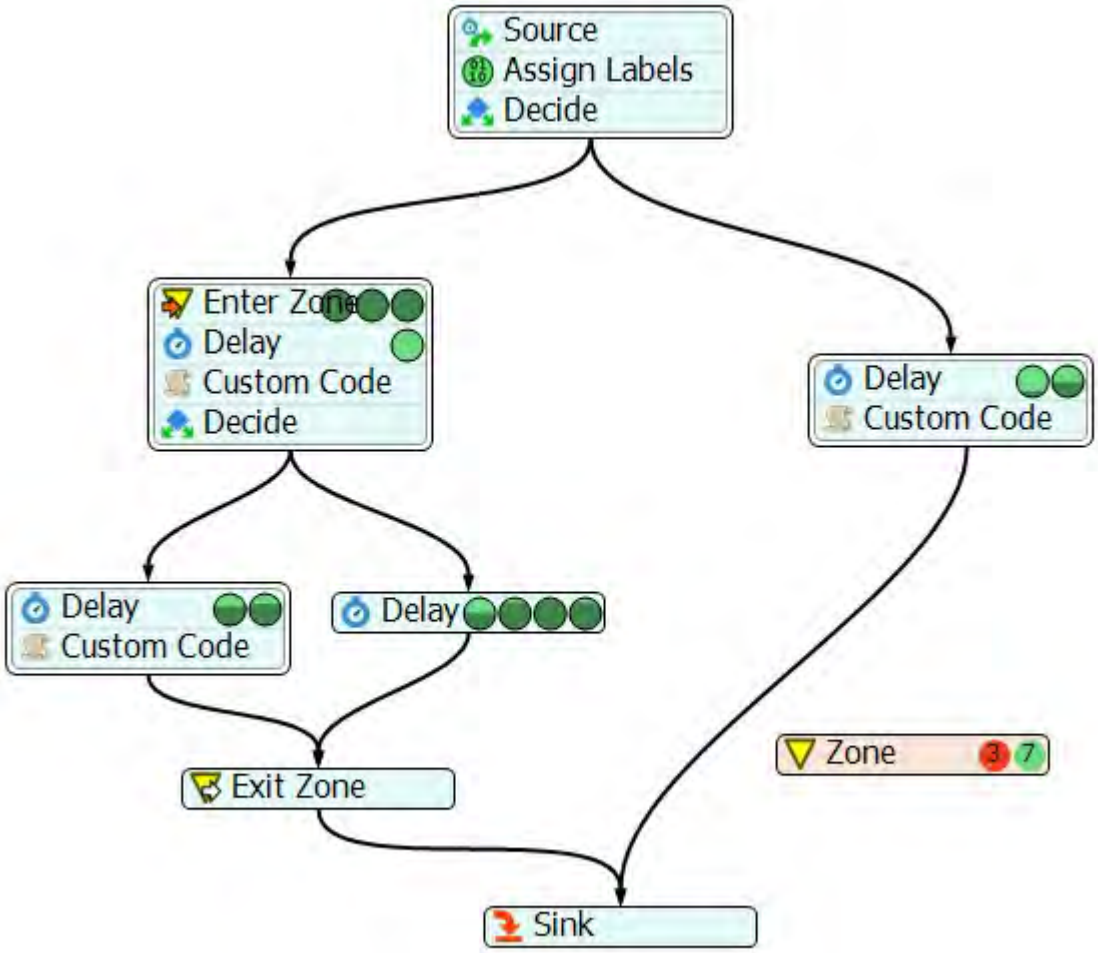
Restricting Access

Zones can be used to restrict access to a set of activities. For example, a Zone might represent a warehouse or a shipping container, and tokens might represent items. Any token that enters the Zone would take up space or weight (or both) within the Zone. Usually, there is some limit on space or weight. The Zone can be used to enforce those limits.

The simplest way to restrict access to a Zone is to limit the number of tokens allowed in the Zone. Using our previous example, suppose at most 7 tokens (which represent items) can fit in the Zone. The Zone has a Use Max Content option, shown in the following image, that enforces that rule:



The following image shows a snapshot during execution, with the max content set to 7:



Notice that there are three tokens on the Enter Zone activity waiting to be allowed into the Zone, which currently has 7 tokens in it. Each token must wait for room to free up in the Zone before entering.

Zone Subsets

You can also use a Zone to gather statistics about a particular subset of tokens. For example, you might be interested in knowing the number of tokens that have an itemtype label value of 1, or the number of tokens that have a weight label value greater than 250. You can collect these statistics by creating a Zone Subset. You can restrict access to a Zone based on Zone Subset statistics as well. It is possible to restrict the number of tokens within a subset, or a total quantity of label values (like the total of all weight labels). A token will not enter a Zone until it complies with all subset restrictions.

Sub Process Flows

A sub process flow (also called a *sub flow* for short) is a separate process flow that begins running when it is triggered by another activity or event in a different process flow. Think of sub flows as chunks of selfcontained logic that will get executed when they are triggered by certain events in the simulation model

or general process flow. If you are familiar with programming terms, you could think of a sub flow as a *function* or a *subroutine*.

You can use sub flows to perform calculations or to simulate more complex procedures that are triggered by specific activities or events in the main process flow. A sub flow could run a simple calculation to dynamically determine a processor's processing time. Or a sub flow could contain a task sequence that will be used by several different task executers at certain points during a simulation run.

For example, perhaps you want to simulate an assembly line of some sort. When items come off an assembly line they will need to go through a set of Quality Assurance procedures after which the item will either pass or fail. You could simulate this using a sub flow that is triggered once a token gets to a particular activity in the process flow.

This topic contains the following sections:

There are three ways of utilizing a Sub Flow:

- Run Sub Flow Activity
- Create Tokens Activity
- executesubflow() command (for more information see the Command Helper or Commands section of the User Manual)

How Sub Flows Work

The following list explains how sub flows work from beginning to end:

1. During a simulation run, a token enters a Run Sub Flow or Create Tokens activity. (See Run Sub Flow or Create Tokens for more information.)

2. The Run Sub Flow or Create Tokens activity then creates one or more *child tokens* on the sub flow's Start activity. (See Start for more information.) The child token is linked to its *parent token*, which is the original token that first entered the activity that triggered the sub flow. When a child token is first created inside a sub flow, it will be a copy of its parent token, meaning that it will inherit all of the labels and data from the parent token. (See Label Access for more information.)

3. The child token will begin to run through the activities in the sub flow. While the child token is running through the sub flow, the parent token will wait on the activity that initiated the sub flow. NOTE: A parent token can't be destroyed if it has child tokens in the model.
4. When the child tokens reach the Finish activity on the sub flow, the child tokens will be destroyed and any necessary information will be updated on the parent token. (See Finish for more information.)

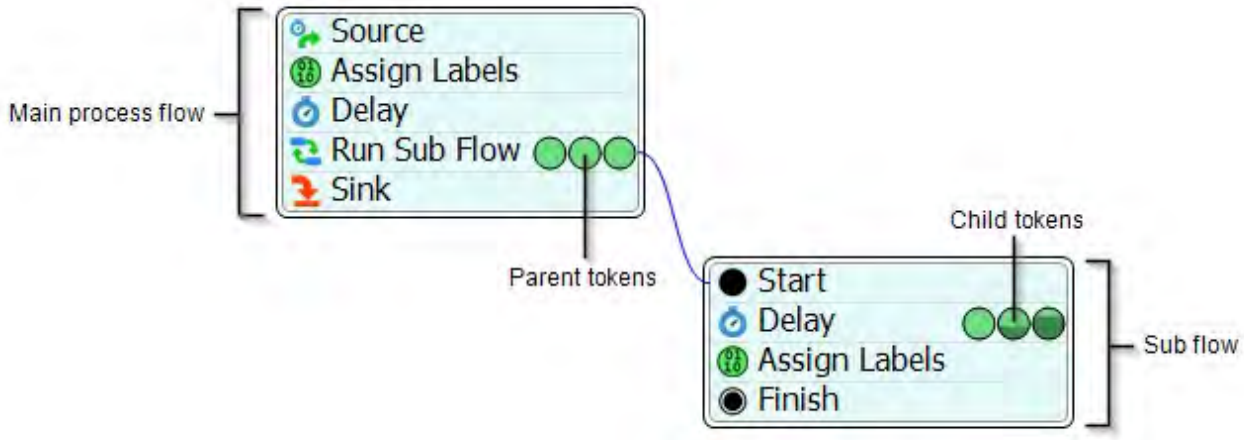
5. The parent token will be released and will move to the next downstream activity in the main process flow. NOTE: You can set the Create Tokens activity to release right after creating the child tokens on the sub flow. The parent token does not necessarily have to be dependent on what happens to the child tokens in the sub flow.

Create Tokens vs. Run Sub Flow

Technically, a Create Tokens activity can send tokens to any activity in a sub flow, but a Run Sub Flow activity can only send tokens to a Start activity at the beginning of a sub flow.

Creating Internal Sub Flows

The following image shows an example of a simple internal sub flow during a simulation run:



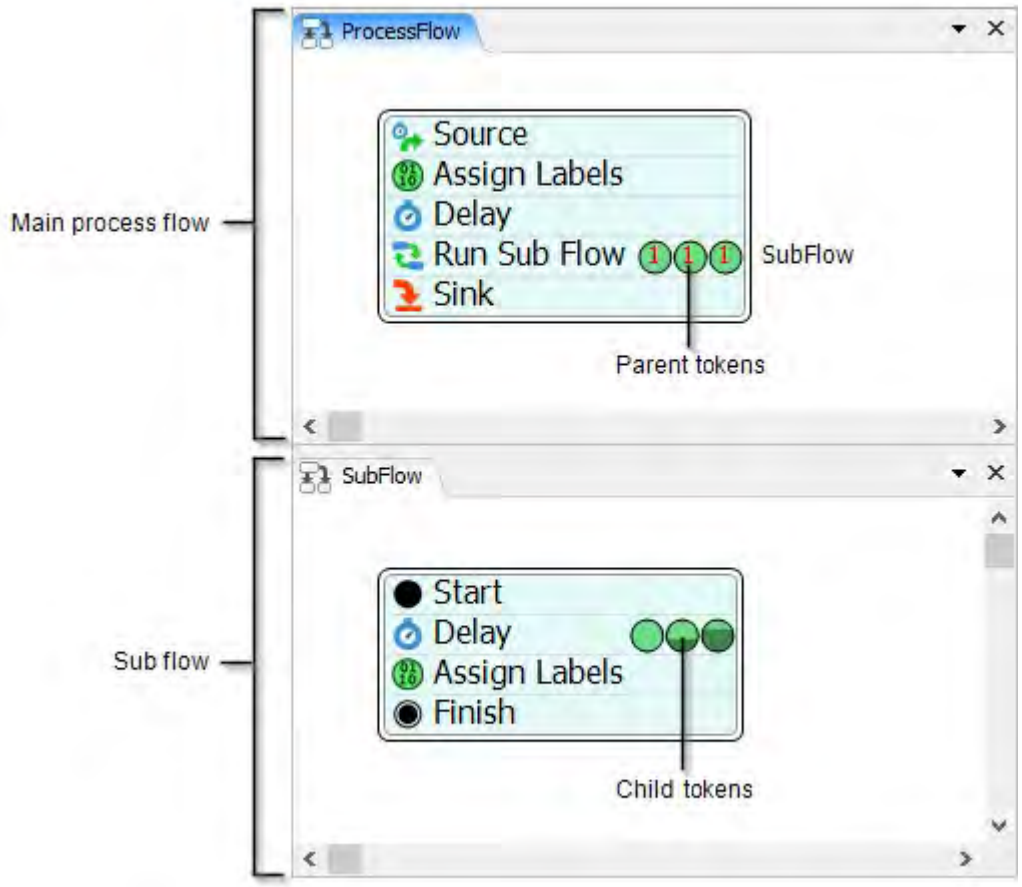
An internal sub flow is a sub flow inside of an existing process flow. Internal sub flows are basically an independent block of activities that begin with a Start activity and end with a Finish activity. Therefore, to create an internal sub flow, simply create a block of activities that begin with a Start activity. After creating the block of activities, link it to the main process flow. See Linking to Sub Flows for more information.

How many internal process flows can be made?

There is no limit to the number of sub flows you can create inside a process flow.

Creating External Sub Flows

The following image shows an example of an external sub flow during a simulation run:



An external sub flow is a separate process flow that was assigned the Sub Flow type when it was first created. It will have its own separate window and will be listed as a separate process flow in the Toolbox. To create an external sub flow:

1. Inside the Toolbox, click the Add button to open a menu.
2. Select Process Flow, then Sub Flow.
3. The new sub flow window will appear and you can begin adding activities to it. NOTE: Sub flows function best when they begin with a Start activity and end with a Finish activity.
4. After creating the block of activities, link it to the main process flow. See Linking to Sub Flows for more information.

Linking to Sub Flows

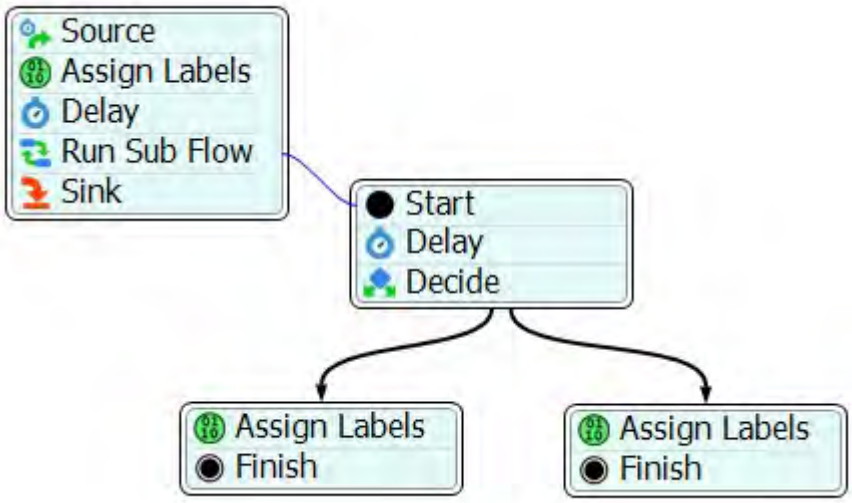
If you are linking to sub flow from a Create Tokens activity, you can use the sampler button next to the Destination property to select the activity (in the sub flow) where you want to create a child token. To link to a sub flow from a Run Sub Flow activity:

1. You'll notice that when you first add a Run Sub Flow activity to the main process flow, there will be an exclamation mark to the right side of the activity. This mark means that you have not yet linked the Run Sub Flow activity to a sub flow, which is required.
2. Click on the exclamation mark to the right side of the Run Sub Flow activity to enter sampling mode. You will know you are in sampling mode because your mouse cursor will change to the sampler icon .
3. Now click on the Start activity at the beginning of the target sub flow. If you are linking to an external process flow, you can click anywhere inside a sub flow window to link to that sub flow.
4. The Destination property will update to show the sub flow to which it is now linked. You can edit this property if needed.

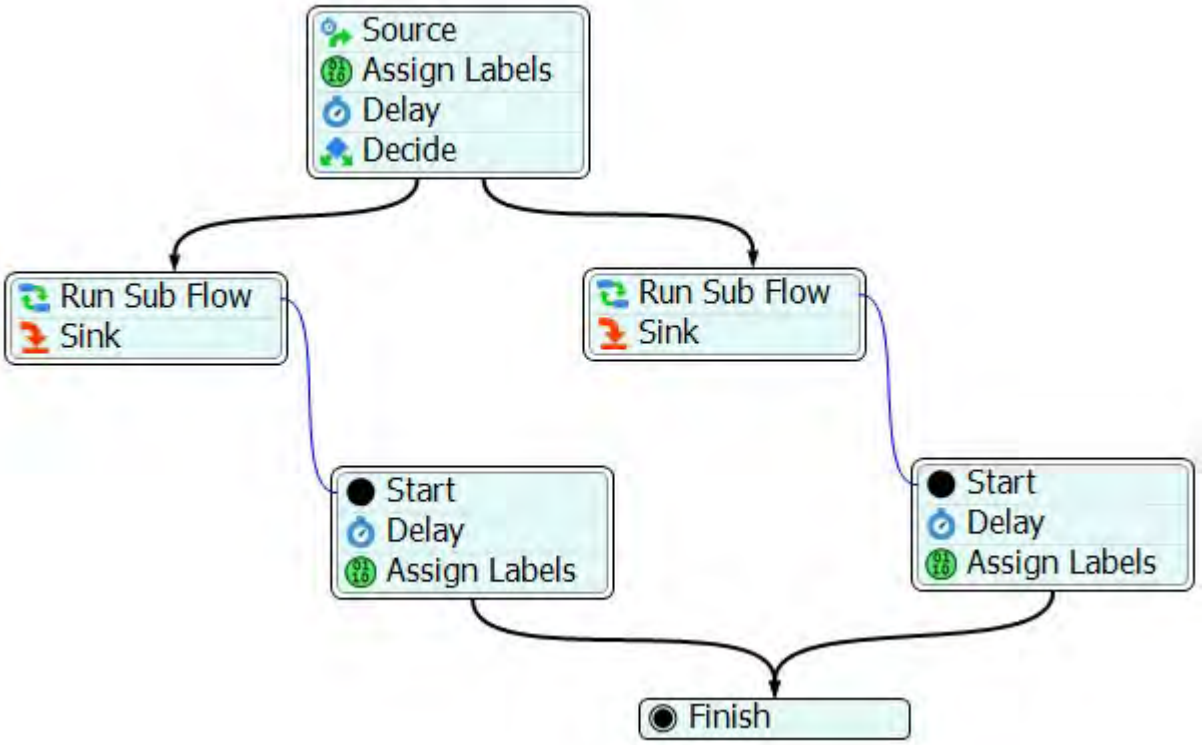
Link directly to Start activities when possible
If linking to an external sub flow window, the Run Sub Flow activity will search through that sub flow's list of activities and find the first Start activity. The Run Sub Flow activity will send any newly created tokens to this Start activity. If the Sub Flow Process Flow contains multiple Start activities, you should link the Run Sub Flow activity to the specific Start activity it should be sent to.

Multiple Start or Finish Activities

A Start activity may be connected to multiple Finish activities, as illustrated in the example sub flow in the following image:



In that same vein, multiple Start activities may share the same Finish activity. Technically, each Start activity would be the beginning of a different sub flow. The following image shows an example of this kind of sub flow:



Linking Parent and Child Labels

One of the advantages of using sub flows it that it will allow you to create links between the labels on parent and child tokens. Child tokens can possibly be set to update the labels on the parent token (and vice versa) during a simulation run.

The following image shows the label-related properties on the Create Tokens activity:

The Run Sub Flow activity's label-related properties are nearly identical to the Create Tokens activity with only two exceptions: 1) there is no Create Tokens As menu, and 2) the Copy Labels to Tokens on Create property is named Copy Labels to Children on Create instead.

- Create AsDefines the relationship of the original token to the created token(s). o Independent Tokens - Created tokens will have no association with the original token.
    - o Child Tokens - Created tokens will be child tokens of the original token.
    - o Sibling Tokens - Created tokens will be sibling tokens of the original token. In other words, they will have the same parent as the original token.
- Label Access on Parent Only By default, the Label Access on Parent Only checkbox is cleared on the Create Tokens activity and checked on the Run Sub Flow activity. When it is cleared, any activity can create, modify, or read labels on either the parent or the child tokens.

  When you check the Label Access on Parent Only checkbox, access to the labels on the child tokens will be restricted. In other words, activities will only be able to create, modify, or read labels on the parent token. This restriction essentially causes your parent token to be in multiple activities at the same time because child tokens will just be duplicates of the parent token (as opposed to being independent tokens). NOTE: This option is the only way a child token can add labels to its parent token.

  When the Label Access on Parent Only checkbox is checked, the next two properties will not be available.
- Parent Label AccessThis menu determines if and how child tokens may access labels owned by their parent. The menu has three available options:
    - o None - The child tokens have no access to their parent's labels. All reading, writing and adding of labels are done on the child. Select this option if you want the child to be a completely separate entity from the parent.
    - o Read - This option only allows child tokens to get information from labels on their parent; the child tokens will not be able the add or set the parent's labels. Select this option if you want to set and add labels on the child rather than the parent. If an activity in a sub flow tries to set a label on the child token that is owned by the parent token, the child will add a label to itself with the given label name and value.
    - o Read / Write - This option allows child tokens to read or set the parent's labels, but the child will not be able to add labels to the parent token. Setting labels that are not on the child or parent will cause a label to be added to the child with the given label name and value.

  Parent / Child Relationships
  When accessing labels on a token, the token will first look at labels owned by itself. If the label doesn't exist and the token has read access of its parent's labels, it will try and access the label on the parent. If the label doesn't exist on the parent, it will check the parent's label access and if able, look up to the next parent's labels, and so forth.
- Copy Labels to Tokens/Children on Create By default, the Copy Labels to Tokens/Children on Create is cleared. When this checkbox is checked, the entering or parent token will copy all of its labels to the created tokens/children. On a Create Tokens activity, if the Create As is set to Sibling Tokens, only labels owned by the entering token will be copied. Otherwise, all labels from all ancestors of the token will be copied (based upon each token's label access type). The copied labels can be read or written on the child regardless of the the option you selected in the Parent Label Access menu. However, new labels added to the parent may still be read or written if the label access type allows it.
- Assign Labels to ChildrenYou can use this group of properties to set labels on the created tokens/children. Each label value will be evaluated for each created token/child. The functionality is identical to the Assign Labels activity. See the Assign Labels activity for more information about these properties.

Understanding Parent-Child Label Access

When accessing labels and setting labels on tokens the tokens look at their parents depending on the label access they have. For example, suppose that you had a token called `theToken`. Suppose that `theToken` doesn't have a *weight* label. Then suppose that `theToken` has a parent token, and that the parent token does have a *weight* label. If `theToken` has been given read access to its parent's labels, then the code `theToken.weight` will get the value of the parent's *weight* label.

Similarly, you can use `theToken.weight = 5;` to set the value of the parent's *weight* label to 5. Again, this only works if `theToken` has a parent, and doesn't have a *weight* label.

This behavior is governed by the type of label access the parent token gives to its children. See Label Access above for more information.

The following table shows how getting and setting labels behave when the given token has a parent. The child token is highlighted, and the parent token is shown above the child. The values on each token represent the *weight* label.

Access Rule	*weight* label value	token.weight	token.weight = 5;
Parent Only	2	2 (from the parent's label)	5
None	2	A null value is returned (the child doesn't have the label)	2 5
Read	2	2 (from the parent's label)	2 5
Read/Write	2	2 (from the parent's label)	5

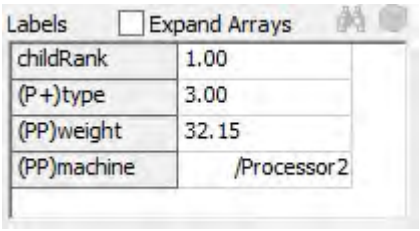
Accessing labels this way is helpful for two reasons. First of all, you don't have to create unique copies of the parent's labels, in order for the children to use them. This helps you avoid duplicating data. This leads to the second benefit. If child behavior is governed by a parent label, then you can change the behavior of all of the children by changing a single label. This helps you keep your logic more centralized.

Viewing Parent/Child Relationships

Once a parent/child relationship has been created, the parent token will display the number of children it has in the process flow view by default. You can change these settings on the Process Flow Properties page if needed. See Changing Process Flow Visuals for more information.

Viewing Labels

If you click on a child token, the Quick Properties will display all of the labels owned by the child token as well as any labels from the parent token that the child has access to, as shown in the example in the following image:



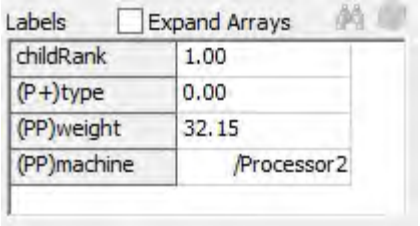
If a label is not accessible to the child, it will not display in this list.

The following table explains the meaning of the symbols on these labels:

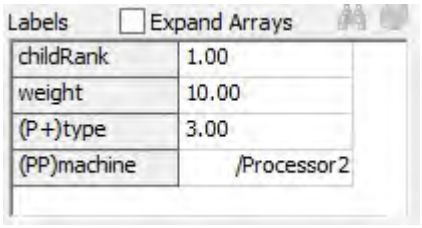
Symbol	Meaning
(P)	The label is owned by the parent
(PP)	The label is owned by the parent's parent (grandparent), and so forth.
+	When it is next to the P, it means that the child token has the ability to write (set) the label.

For example, in the example from the image above:

- Setting the "type" label on the child would cause the "type" label to be set on the parent token.



- Setting the "weight" label on the child would cause a "weight" label to be added to the child. The child would no longer have access to its grandparents "weight" label as reading the label would read it off of the child.



Accessing Child/Parent Tokens

To access a parent token, enter the following in a property field:
token.parent

Child tokens can be accessed by rank through the parent using the command: token.children[1]

To get the # of children the parent has:
token.children.length

Task Sequences

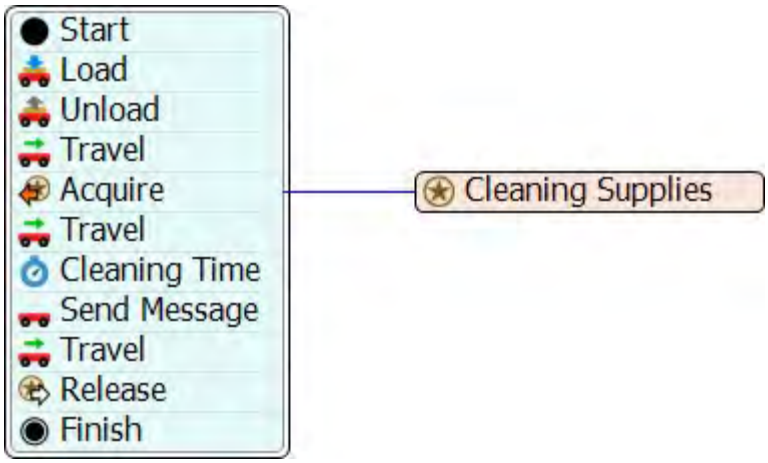
In previous versions of FlexSim, creating custom task sequences required writing FlexScript code. Now with the process flow module, you can create complex task sequences without code. You'll also be able to more easily visualize and edit task sequences because they'll be listed in a step-by-step pattern.

This topic will explain the key concepts related to task sequences. It contains the following sections:

What is a Task Sequence?

A *task* is an individual task activity that could be assigned to a task executer such as loading a flowitem, traveling to a specific destination, etc. A *task sequence* is a set of tasks, a series of actions that can be assigned to a task executer (such as an Operator or Transporter). The task executer will usually complete each task in a task sequence in sequential order.

The following image shows an example of a task sequence:



In this example, a task executer will clean a processing station after transporting an item, using the following task sequence:

1. Load or pick up a flowitem from a processor
2. Unload a flowitem at a new destination
3. Travel to the closet where the cleaning supplies are stored
4. Acquire the cleaning supplies (a Resource)
5. Travel back to the processor

6. Clean the processor (represented by a timed Delay)
7. Send a Message to the processor indicating it is now clean and can accept more flowitems (represented by a Custom Task)

8. Travel back to the closet to return the cleaning supplies

9. Release the cleaning supplies (return them to the supply closet)

Once you've created a task sequence such as this, you can easily assign any task executer to the task sequence without writing Flexscript code or using a Dispatch object.

How Task Sequences Work

The following list explains how a basic task sequence works from beginning to end:

1. During a simulation run, a token enters any Task Sequence activity (such as a Travel or a Load activity).

2. The task is assigned to a task executer or task sequence.

    - If the task is assigned to a task executer, a unique task sequence is created on the task executer that represents that specific task.
    - If the task is assigned to a task sequence, the task will be appended to the end of the task sequence.

3. The task executer will complete task sequences in the order they are received. If task sequences have different priorities, they will be performed in the order of their priority.

4. If the task executer finishes a task sequence and nothing else is added to the task sequence, the task sequence will be deleted.

5. After the task sequence is deleted, the task executer is free to work on other task sequences.

If multiple tokens or objects give task sequences to a task executer, you may need to adjust how the tasks are implemented in the process flow to ensure they are performed in the proper order. This can be done by using a Resource to prevent multiple tokens from giving the task executer tasks simultaneously, or by assigning the tasks to a task sequence (not a task executer). See the Executer / Task Sequence property for more information.

If you want to create a more complex system for handling tasks, there are a few different ways you can change the way a task executer handles task sequences:

- You can create a complete task sequence and dispatch it to a Global List rather than a task executer. See Creating and Dispatching Task Sequences for more information.

- You can prevent a task sequence from being deleted because it is empty. See Deleting Task Sequences for more information.

- You can assign priorities to task sequences so that important task sequences are completed before less important ones. You can also make a task sequence preempt (interrupt) another less important task sequence. See Task Sequence Priorities and Preemption for more information.

Creating and Dispatching Task Sequences

As was stated in the previous section, using any of the task sequence activities in a process flow will automatically create a task sequence any time that task sequence is used in a simulation run. However, you can use a Create Task Sequence to:

- Associate a set of tasks together, not allowing the task executer do other tasks in the middle.
- Create a task sequence up front, and then dispatch it later in the simulation run.

When you use a Create Task Sequence activity, you are essentially grouping a set of tasks together into a single unit, so the task executer will not do anything else until that entire group of tasks is finished. Once the task executer has completed all of the tasks in a task sequence, the task sequence will be deleted. The Dispatch Task Sequence activity can dispatch a task sequence to a task executer. Generally you won't need to use the Dispatch Task Sequence activity with your task sequences because the Create Task Sequence activity and other task sequence activities will be dispatched immediately by default. However, you could use a Dispatch Task Sequence if you would prefer instead to build a series of task sequences first and dispatch them later. There are two different methods you can use to build a task sequence that is dispatched later:

- You could use a Task Sequence Global List. (See Global Lists for more information.) Using this method, the task sequences you create will be pushed to the global list where it will wait until a task executer becomes available. When a task executer becomes available, a token could pull the task sequence from the Global List will then dispatch the task sequence to the Task Executer.
- You could use a Create Task Sequence activity to create a task sequence but set the Task Executer / Dispatcher property to *None*. (See Create Task Sequence for more information about this property.) You can then add tasks to the end of this task sequence (not waiting for them to be finished). After the task sequence has been built, the token can later dispatch it using the Dispatch Task Sequence activity. (See Dispatch Task Sequence for more information.)

Deleting Task Sequences

When a task executer finishes all the tasks in a task sequence, the task sequence is deleted. In other words, if you don't want a task sequence to be deleted, you should make sure that the task sequence is continually supplied with new tasks. You might experience error messages if you try to reference a task sequence that has been deleted. You can avoid these problems two different ways:

- You could give tasks to the Task Executer directly using the task sequence activities, instead of using the Create Task Sequence activity.
- You could use a Wait for Task activity in your task sequence at the point where a delay may occur.

You may have noticed that there is no Wait for Task activity in the Process Flow Library. To use a Wait for Task activity:

1. Add a Custom Task activity to your process flow.

2. In Quick Properties for the activity, click the Task Type menu to open it. Select Wait for Task from the menu.

Task Sequence Priorities and Preemption

By default, task executers will complete task sequences and activities in the order they were received (first in, first out). However, you might want a task executer to complete more important tasks first. You can use the Priority box on the Create Task Sequence activity to cause the task executer to work on higher priority tasks sequences before lower priority task sequences. You can also use the Preemption menu to determine if requests for higher priority tasks should force a task executer to stop its current task sequence when it receives a higher priority task sequence. See the Create Task Sequence activity for more information.

Assigning Task Executers

The Executer / Task Sequence box is available on most of the Task Sequence activities. You can use this box to determine which task executer or task sequence should receive the task. If you choose to give this task to a task executer, a new task sequence will be automatically created with this task and then it will be sent to the task executer. You can:

- Assign a specific, fixed task executer in the 3D model
- Dynamically assign the task executer during a simulation run using labels
- Assign the task executer using the `current` command if you are in a task executer or sub flow process flow type.
- Append this task to an existing task sequence

See Common Process Flow Object Properties - Executer / Task Sequence for more information about how to assign task executers to a task.

For More Information

See the Task Sequence tutorial for an example of how to build a task sequence and link it to a 3D simulation model.

If you would like a deeper understanding of how tasks sequences work without process flow, see Task Sequences - Concepts and the subsequent topics in this chapter. You might get a deeper understanding of how task sequences work by reading this chapter.

Process Flow Instances

In the Process Flow module, an instance is a single occurrence of a sub flow, fixed resource process flow, or task executer sub flow. Usually, you'll create one of these kinds of process flow to act as a basic template. When an object (such as a fixed resource or task executer) uses the process flow during a simulation run,

it will become a separate instance of that process flow. For that reason, there might be multiple instances of the same process flow running at the same time during a simulation run.

General Process Flows Are the Exception
General process flows can only have one instance at a time. See Process Flow Types for more information.

This topic contains the following sections:

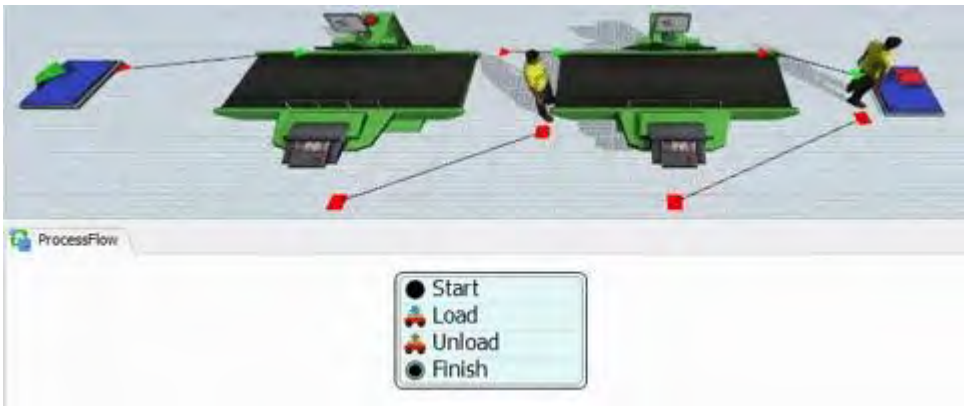
- About Process Flow Instances
- Reasons to Use Instances
- Properties Related to Instances
- Viewing Instances

About Process Flow Instances

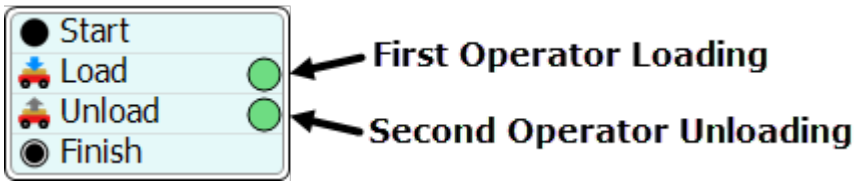
The concept of instances is somewhat abstract. For that reason it might help to compare it to something concrete, such as cooking with a recipe. When you build any kind of process flow other than a general process flow, it's as though you are creating a basic recipe for a particular meal. Individual chefs can then make that recipe on their own. Each time someone makes the recipe, they are making an *instance* of that recipe. Several individuals might be cooking the meal using the recipe at the same time. But they are all using the same basic recipe as their starting point. Also, if you change the recipe, you will change all instances that use that recipe from that point forward.

In that same vein, when you build a process flow, you are building a "recipe" for the basic logic that will control the objects in your simulation model. During a simulation run, many different objects could use the same process flow logic (the recipe), but each one would be a separate instance. If you change the main process flow (the recipe) process flow, it will change all the instances of that process flow as well.

The Task Sequence Tutorial provides a useful example of process flow instances. In this tutorial, you create a sub flow containing a simple task sequence for loading and unloading flowitems from a processor. The model has two processors, each with its own operator. Both operators use the task sequence in the main process flow to load and unload the flowitems. The following image shows what happens during a simulation run:



In this case, the main process flow is the recipe because it contains the basic task sequence that will be used by both operators. Because of the way the general properties were defined on the main process flow, each operator that runs the task sequence in the sub flow will be considered a separate instance of that process flow. During a simulation run, each token in the main process flow represents one of the operators, as shown in the following image:



The following image shows what it would look like if you were viewing the main process flow and each individual instance at the same time:

If you were to make a change to the task sequence in the main process flow, it would immediately update all the instances of that process flow as well.

Reasons to Use Instances

There are a two advantages in using process flow instances:

1. Being able to manage shared assets across instances
2. Creating a dynamic reference to an instance

Each of these concepts will be explained in more detail in this section.

Managing Shared Assets Across Instances

The most important reason for understanding and using process flow instances has to do with managing shared assets across instances. As a reminder, a shared asset is a special type of process flow object that functions differently from other activities in a process flow. There are three types of shared assets:

- Resource - Represents a limited supply of some resource that can be acquired and released. It can be used to simulate a supply of goods, services, time, materials, employees, etc.
- List - Can be used to add or remove tokens, flowitems, task executers, etc. to a list of things that are available for a specific purpose. Process flows can use a list that is local to the process flow itself or could be tied to a Global List in the simulation model.
- Zone - Can collect statistical information not available for standard activities. It can also restrict access to a section of the process flow based on certain statistics or other criteria.

Whenever a process flow can have multiple instances, you can determine how any shared assets are distributed among the instances of that process flow. For example, imagine you build a sub flow that uses a Resource with a count of 3. The Resource can be managed one of two ways:

- Global - Shared assets will be globally accessible by all instances. For example, if you have a Resource with a count of 3, any instance that uses that Resource will deplete the total count. If one instance uses all three Resources, another instance won't be able to use it.
- Local - Shared assets will only be locally accessible to instances. For example, if you have a Resource with a count of 3, each instance will have a local copy of that Resource with a count of 3. When an instance uses a Resource it will only deplete the local count. If one instance uses all three of its Resources, another instance will still have access to its three Resources as well.

You can change these settings on each shared asset you add to your process flow. Each shared asset has a Type menu in its Quick Properties. You can use this menu to control whether shared assets are global or local to instances. See Resource, List, and Zone for more information. Note that this setting is only available if the process flow type not general. For general process flows, there is always just one instance, so shared asset access is always global.

Referencing the Instance

Another advantage of using process flow instances is that you can easily reference the object associated with the instance. When using Process Flow instances, the `current` keyword gives a reference to the token's instance object. Technically, this is not a critical feature because you can always store labels on the token that point to the object that you would otherwise use as the instance object. Nevertheless, using `current` can be more intuitive.

Properties Related to Instances

The properties that affect instances depend on what type of process flow it is:

- General process flow types cannot have more than one instance.
- Each fixed resource that runs a Fixed Resource process flow type will be a separate instance of that process flow.
- Each task executer that is attached to a Task Executer process flow type will be a separate instance of that process flow.

- Sub Flow process flow types have a special property called Instance Creation that allow you to define when the FlexSim system will create an instance of a sub flow. This can affect the way local shared assets are used. See General Properties - Instance Creation for more information.

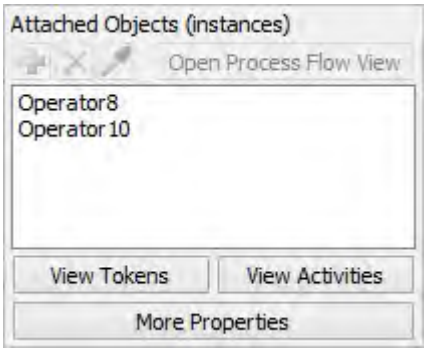
See Process Flow Types for more information.

Additionally, you can use the Type menu on Shared Assets objects (Resource, List, and Zone) to determine whether these assets are global or local to an instance.

Viewing Instances

To view an instance of a process flow:

1. Click in a blank spot in the process flow to ensure nothing is selected.
2. In Quick Properties, find the Attached Objects (instances) group. This lists the objects that are attached to the process flow, as shown in the following image:



3. Click one of the objects in the list to select it. Then click the Open Process Flow View button to open the process flow view for that specific instance. This view will only show the tokens associated with that instance.

Process Flow Variables

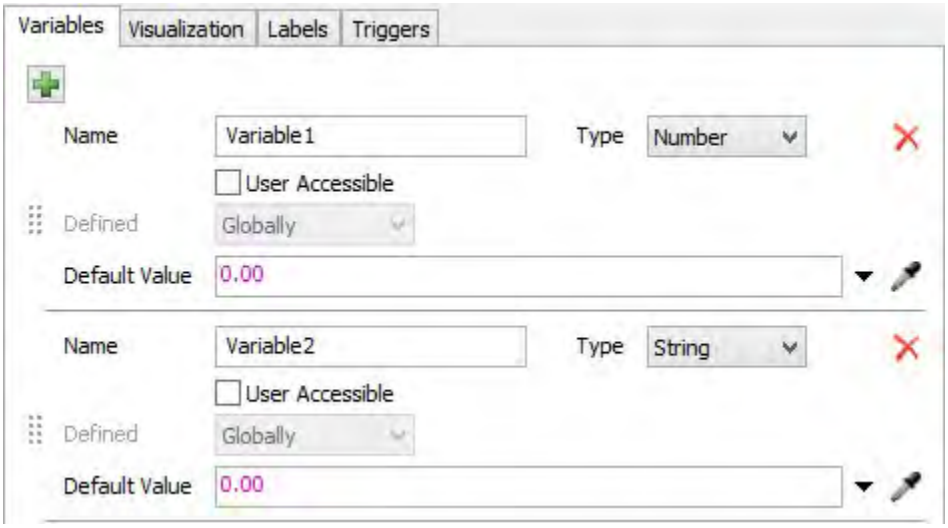
Process Flow objects can have variables defined on them that can be referenced from any activity through the Universal Edit fields or through the `getprocessflowvar()` command.

Why Use Process Flow Variables?

Process Flow Variables allow you to present certain inputs in the Process Flow in an easy-to-use front-end. Let's say there is a certain input that may change across different instances, or perhaps across different scenarios that you want to test. Instead of requiring you or another person to dive into the internal workings of the Process Flow, you can place it in a Process Flow variable, then turn off Edit Mode so that when users want to change it, they can change it through an easy front-end interface.

Properties

The following image shows properties for Process Flow Variables:



Each of these properties will be explained in the following sections.

- NameYou can set the name of the variable here.
- Type This specifies the return value of this variable, which can be Any, Number, Object, Node, or String.
- User Accessible If checked, this variable will be visible and editable to anyone. If unchecked, this variable will remain internal to this Process Flow.
- DefinedThis defines the instancing behavior for the Process Flow Variable. If set to Global, there will be one variable for every instance of this Process Flow. Since there is only ever one instance of a General Process Flow, this option is always set to Global for General Process Flows. If set to Per Instance, each instance will have its own version of this variable. Refer to the Types of Process Flows documentation for more information on instancing.
- Default ValueThis is where you define the code to calculate this variable.

User Libraries

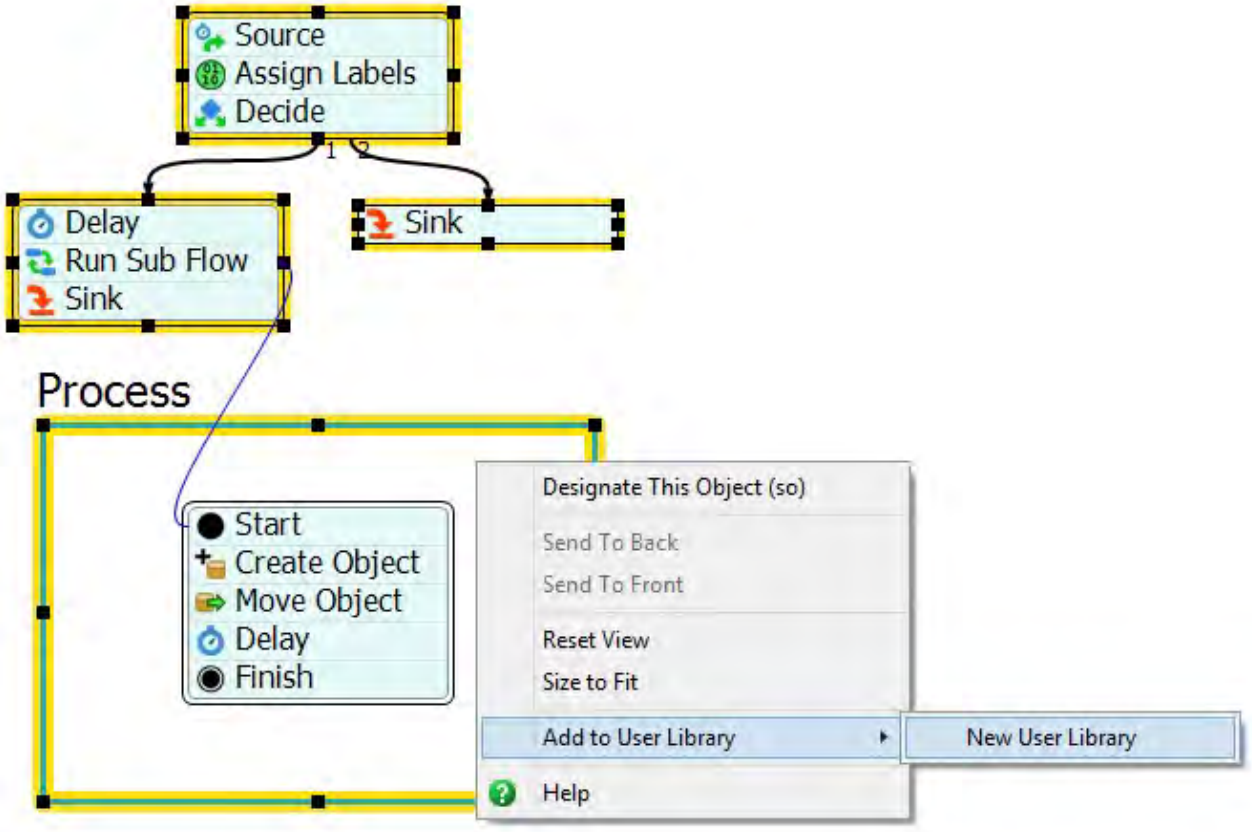
Activities and display objects, Process Flow objects and 3D objects attached to Process Flow objects can be added to user libraries. This topic will cover how to add these objects to a library. For more information on user libraries and how to modify them, see the Custom User Libraries topic.

External References

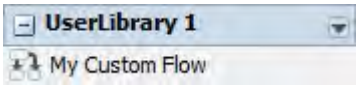
When you add a Process Flow object or a set of activities to a user library, keep in mind that references that are external to your object or activities will not be saved in your library object. For example, if you have a user library object that includes a Create Object activity that references Processor4 in your model as its destination, when the library is distributed to other users, that reference will be null and be displayed as None. One way to solve this issue is to reference the external object by name using flexscript. For example, node("Processor4", model()). If you are creating a Process Flow user library object you can set the destination property to a Process Flow Variable. When the library is distributed to other users, they only need to set up the user accessible Process Flow Variables for the object to function.

Add Activities and Display Objects to a User Library

- Select all of the activities and display objects you want to add as a single user library object by control clicking each object or by holding the shift key down, clicking on white space and dragging the mouse around all objects.
- Right click and select Add To User Library > New User Library.



A User Library will appear in the Library window. This library can now be saved and distributed to other users.



If you need to modify a library object once it has been added to a user library, drag and drop the object from the library into a process flow view, make the desired modifications and then add it to the user library. You can then delete the old object from the library by clicking on the down arrow to the left of the user library name and clicking Explore Tree.

Add a Process Flow Object to a User Library

- In the quick properties for the Process Flow Properties, click the button next to the Process Flow Name field.
- Select the user library to add to, or create a new user library.
- Select As Draggable Icon or As Auto-Install Component.

When a Process Flow object is added to a user library, any instances, or attached objects, will be removed. The exception is for the General Process Flow which has one instance, itself.

Once added, a Process Flow object can be updated by adding it to the user library again. This will overwrite the previous Process Flow with the new one.

Adding Attached Objects (Instances) to a User Library

When you add a 3D object like a Task Executer or Fixed Resource object to a user library that is attached to a Process Flow, the Process Flow object will automatically be added to the user library. By default, the Process Flow object is added as a draggable icon along side the 3D object that was added. The Process Flow object can be moved into an auto-install component if desired.

When the 3D object is dragged into the model, the library will check to see if attached Process Flow objects are currently in the model. If the Process Flow isn't in the model, the Process Flow object will be added. If it does exist, the object will be connected to the preexisting Process Flow object, but the Process Flow will not be updated. To updated the Process Flow, manually drag the Process Flow object from the user library into the model. Doing this will maintain all of the Process Flow instances.

User Libraries Compare Object Names

When saving to a user library and when creating an object from a user library, the Process Flow object's name is used to compare with what already exists in the model or user library. If a Process Flow object is saved to a user library with another Process Flow that has the same name, that Process Flow will be overwritten. When a 3D object is dragged into the model, it will check for a Process Flow object in the model with the same name as the attached Process Flow.

Process Flow Coordination

This topic will discuss some of the key concepts related to coordination in Process Flow. It contains the following sections:

- What are Coordination Activities?
- Process Flow Activities Used in Coordination
- What Makes Coordination Activities Different
- Key Concepts About Connectors
- Waves and Partitions

What are Coordination Activities?

The Coordination activities provide an easy way to both split a token into multiple tokens and coordinate timing between multiple tokens. They also improve the visibility of your logic by using connectors to visually layout when tokens are created, when they wait for each other, and when they come back together. Coordination activities have several possible applications, such as:

- Coordinated Task Sequences - You may need multiple operators to perform a series of tasks together. You could use Coordination activities to make a token for each operator, synchronize when they finish certain tasks, and then destroy the extra tokens when the task is finished.

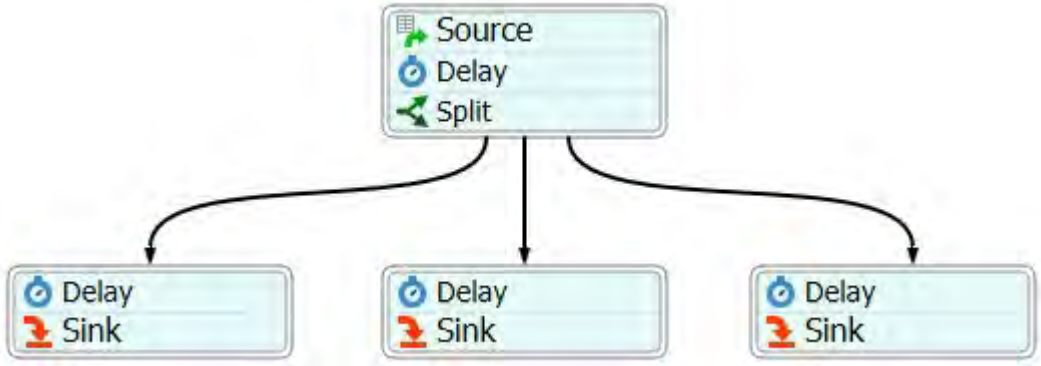
- Concurrent Processes - Perhaps you need to start several processes at once, have them happen concurrently and then wait for all of them to finish. For example, a doctor orders a series of tests and the samples get sent off to separate areas where they are processed in parallel. The doctor then needs to be notified when the results of each test are done. Coordination activities could be used to make a token representing each test that would carry out the test logic concurrently and then have all the tokens rejoin before having the doctor look at the results.

Process Flow Activities Used in Coordination

There are three Coordination activities, as described in the following sections:

Split

When a token enters a Split activity, it splits that token into multiple tokens and releases them through its outgoing connectors. The number of tokens it releases is based on how many outgoing connectors it has.



Important considerations when using the Split activity:

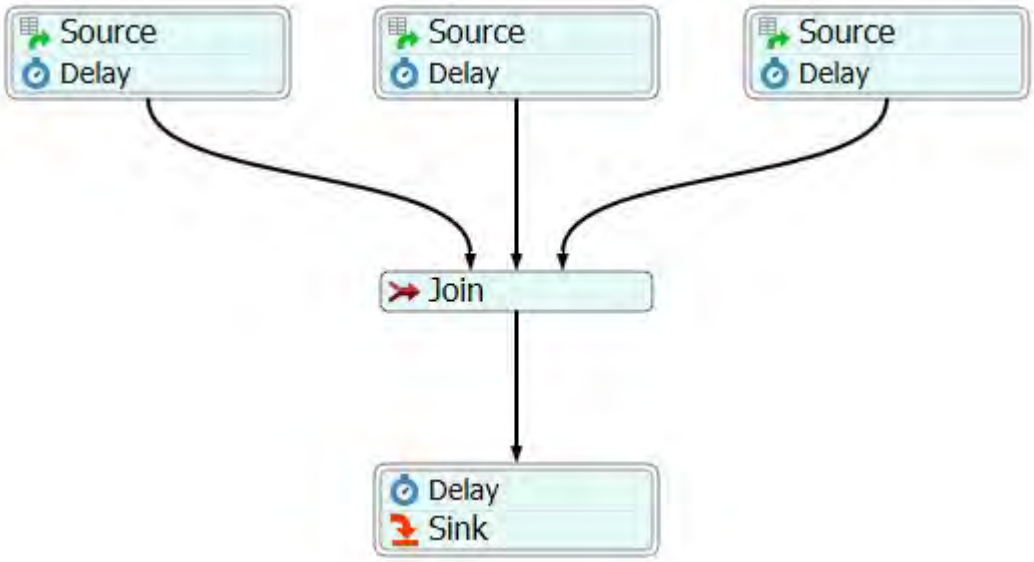
- Number of Connectors - The Split activity can have as many incoming or outgoing connectors as needed. The number of incoming connectors will not affect its behavior, only the number of outgoing connectors.

- Labels - The Split activity adds a label to each newly created token that contains a reference to the original token. By default, this label is named *SplitID*, but you can customize that label if needed. You can possibly use this label with the Join or Synchronize activities to make the tokens wait for each other downstream in the process flow. See Split ID for more information.

See Split for more information about this activity's properties.

Join

The Join activity will wait until it has a token from each of its incoming connectors before releasing a single token.



Important considerations when using the Join activity:

- Number of Connectors - The Join activity can have as many incoming connectors as needed, but it can only have one outgoing connector. You can use a Join activity in combination with a Split activity if you need to have a different number of incoming vs. outgoing connectors.

- Partitions - If needed, you can set the Join activity so that it will only join tokens that have matching partition IDs. In other words, you can join tokens that have matching labels if you want to sort tokens into groups based on a common attribute. (See the following section about Partitions and Waves for more information about partitions.)

- Max Idle and Max Wait Timers - The Join activity also has the option to use the Max Wait and Max Idle Timers if you want the activity to release the tokens after a maximum amount of time has elapsed, even if it does not yet have a token from each incoming connector.

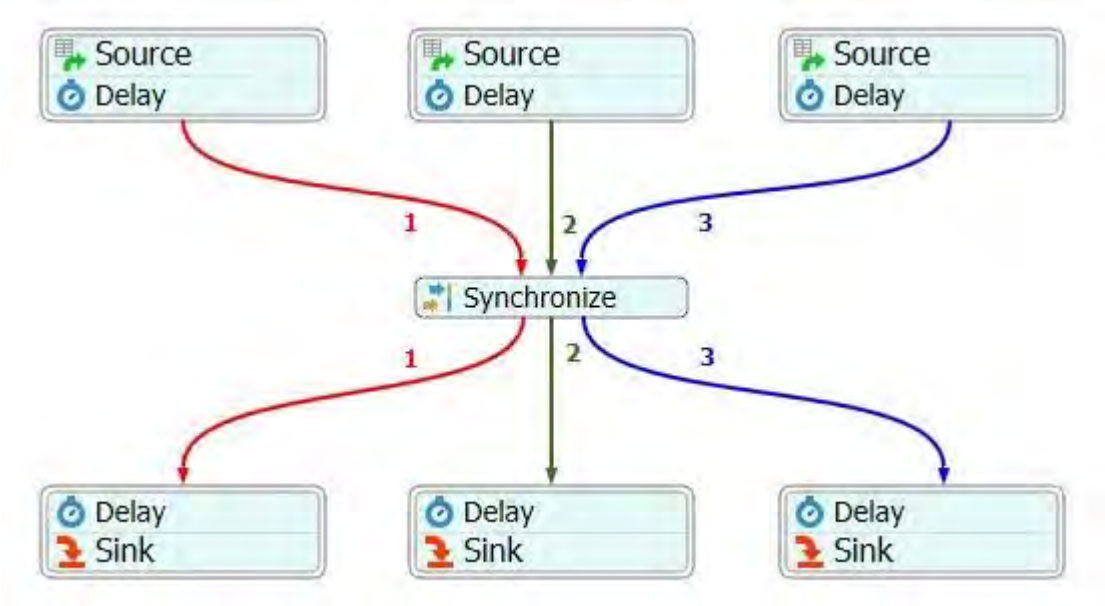
Labels on the Outgoing Token
The Join activity will only release the token that entered through the first connector, meaning the connector with the rank of 1. The Join activity destroys the other tokens.

The outgoing token will not retain any of the information from labels that are on the destroyed tokens; it will only retain the data on the labels that were on the released token before it entered the Join activity.

> If you need an activity that combines the data from the labels of all the tokens and changes the data in the labels on the released token, consider using a Batch activity instead. See Join for more information about this activity's properties.

Synchronize

The Synchronize activity waits until it has a token from each of its incoming connectors, then it releases all the tokens at the same time. The Synchronize activity is similar to the Join activity, but rather than releasing only one token, it releases all the tokens simultaneously.



Important considerations when using the Synchronize activity:

- Number of Connectors - When using a Synchronize activity, it is important to have the same number of outgoing connectors as there are incoming connectors.
- Connector Rankings - It's also important to check the ranks of your incoming and outgoing connectors. The Synchronize activity will release the token that came in through the first incoming connector (rank 1) through the first outgoing connector (also ranked 1). The token that came in the second incoming connector will be released through the second outgoing connector, etc.
- Partitions - If needed, you can set the Synchronize activity so that it will collect and release tokens with matching partition IDs. In other words, you can synchronize tokens that have matching labels if you want to sort tokens into groups based on a common attribute. (See the following section about Partitions and Waves for more information about partitions.)

See Synchronize for more information about this activity's properties.

What Makes Coordination Activities Different

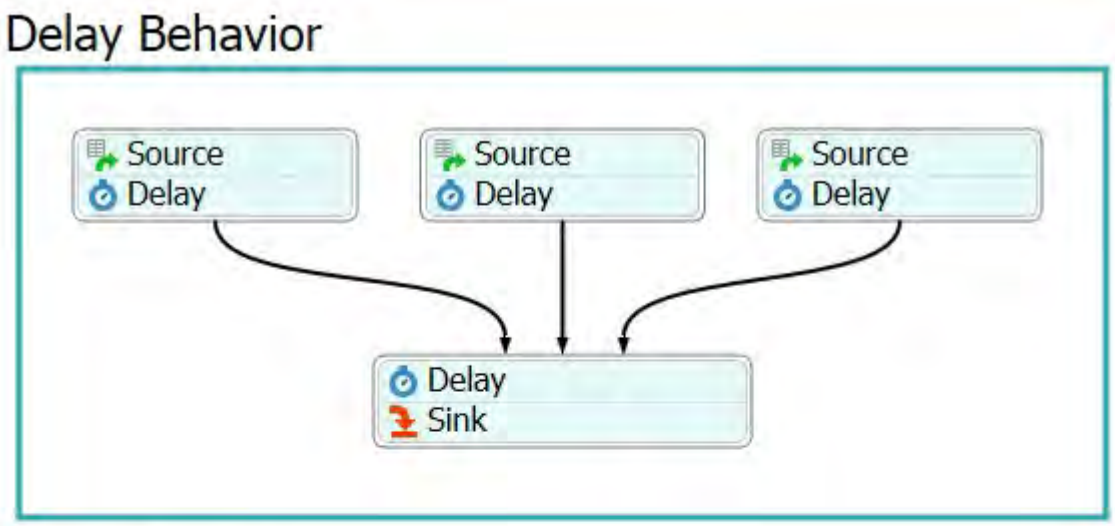
If you've been using Process Flow for a while, you're used to controlling the logic and behavior of the activities by editing the activity's properties (settings). This isn't the case with the Coordination activities. While the Coordination do have a few properties that you can customize, the behavior of Coordination activities is mostly determined by their incoming and outgoing connectors.

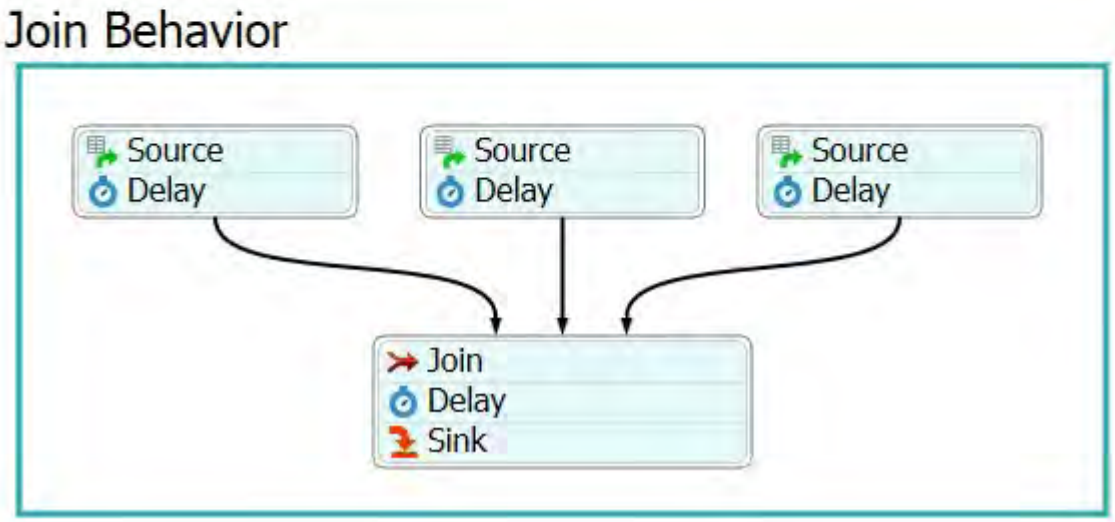
When a token enters most process flow activities, it doesn't matter at all which incoming connector it used to enter the activity. For example, a Delay activity will treat all incoming tokens the same. No matter which connector the token came from, the Delay activity will apply the same Delay time or statistical distribution for each entering token. In contrast, the Join activity will wait until a token has entered from each incoming connector before it releases a token. In other words, the amount of time an arriving token waits at the Join activity is determined by:

1. Which connector it entered through

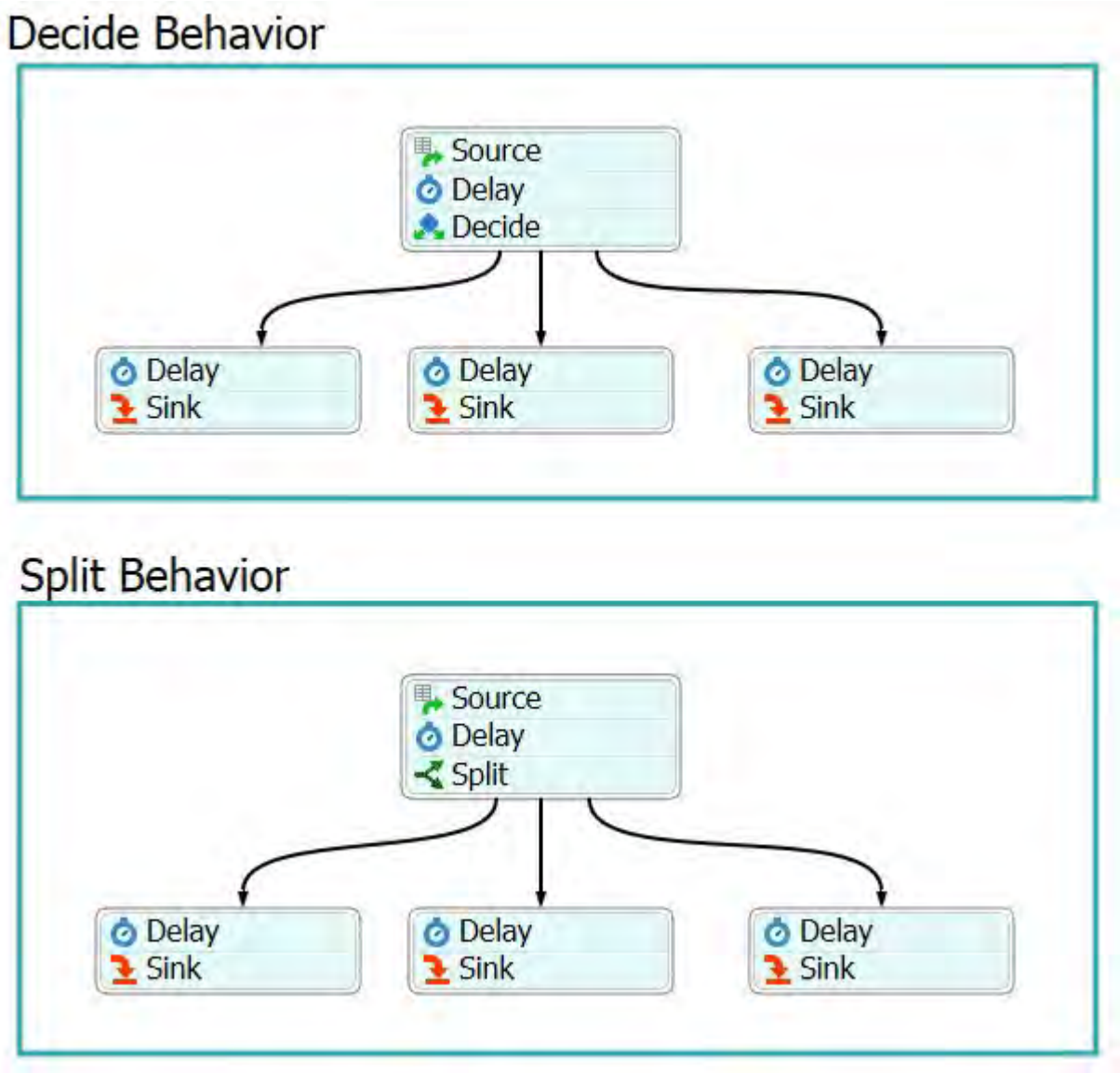
2. Whether other waiting tokens have entered through the other connectors

The number of incoming connectors is also important since that will determine the number of tokens these activities will be waiting for.





The number of outgoing connectors will affect the behavior of the Split and Synchronize activities. These activities allow multiple outgoing connectors. Most process flow activities don't allow multiple outgoing connectors, but the ones that do can only release a token through one of its outgoing connectors at a time. For example, a Decide activity might have three outgoing connectors and it will use its activity properties to send a token into one of those three outgoing connectors. In contrast, the Split and Synchronize activities will release tokens through all of its outgoing connectors at the same time. For that reason, the number of outgoing connectors is important because it affects how many tokens are released from the activity at the same time.



Key Concepts About Connectors

The most important thing to pay attention to when using the three coordination activities is the number of incoming and outgoing connectors you have. Depending on which activity you use, the connectors will affect:

- How many tokens the activity needs to collect before they can be released
- How many tokens the activity releases
- Which connector the tokens will be released through

The following table summarizes how connectors affect the behavior of the coordination activities:

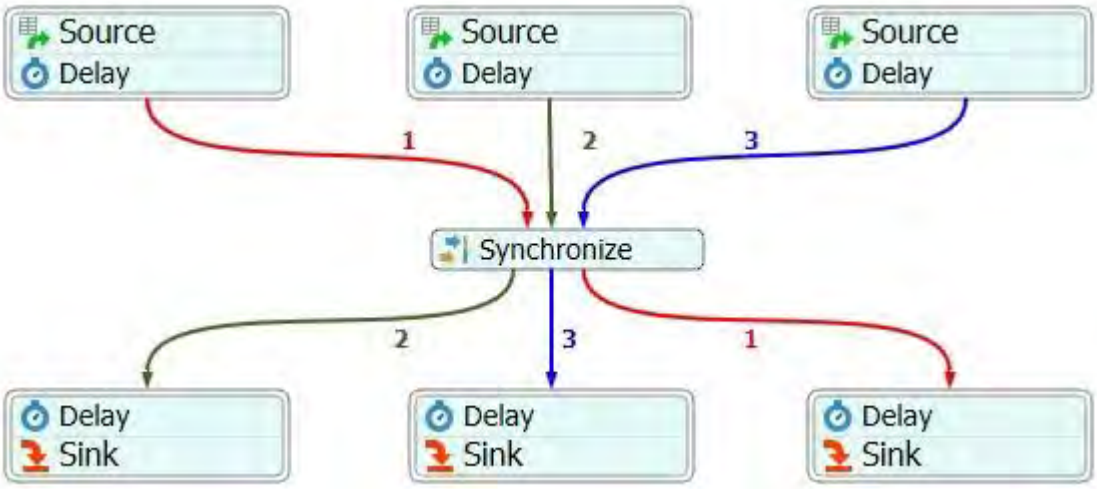
Activity	Incoming Connectors	Outgoing Connectors
Split	Can have as many incoming connectors as needed	Can have as many outgoing connectors as needed Will create and release one token through each outgoing connector
Join	Can have as many incoming connectors as needed Will not release the tokens until there is one from each incoming connector Can sort incoming tokens into groups based on their partition IDs (see the following section about Partitions and Waves for more information)	Can only have one outgoing connector Will release one token through the outgoing connector The released token will inherit the labels of the token that entered through the first connector Will only release the token that entered through the first connector, meaning the connector with the rank of 1; all other tokens are destroyed (see the following section about Connector Rankings for more information)
Synchronize	Can have as many incoming connectors as needed, but needs to have the same number of incoming connectors as outgoing connectors Can sort incoming tokens into groups based on their partition IDs (see the following section about Partitions and Waves for more information)	Will release one token through each outgoing connector Will display a warning in the System Console if the number of incoming connectors does not match the number of outgoing connectors Will use connector rankings to determine which connectors it should send tokens to, as discussed in the following section about Connector Rankings

Connector Rankings

When using the Join and Synchronize activities, it's important to check the ranks of your incoming and outgoing connectors. The Join activity will only release the token that came in through the first incoming connector (the connector that is ranked 1) and will destroy the other tokens. For that reason, you should check that the token you want to release is the one that will come in through the first connector on the Join activity.

The Synchronize activity will release the token that came in through the first incoming connector (the connector that is ranked 1) through the first outgoing connector (also ranked 1). The token that came in the second incoming connector will be released through the second outgoing connector, etc.

For example, the following image shows a Synchronize activity with three incoming connectors ranked 1-3 from left the right. The outgoing connectors have a different ranking pattern. Notice that the red token comes in through the connector ranked 1 on the left and exits through the connector ranked 1 on the right:

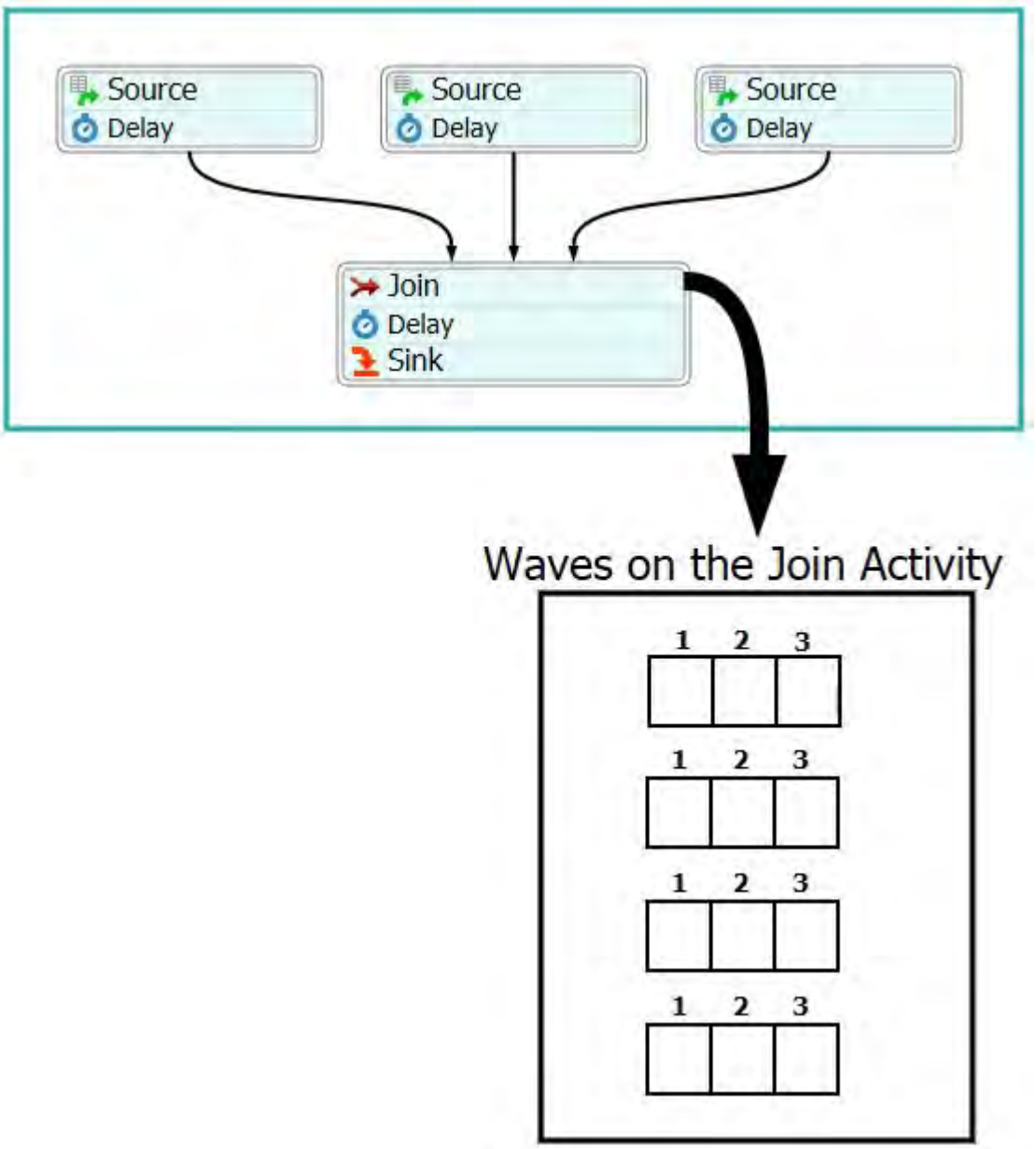


To edit a connector's rank number, click the connector to select it and look at the Rank box in Quick Properties.

Waves and Partitions

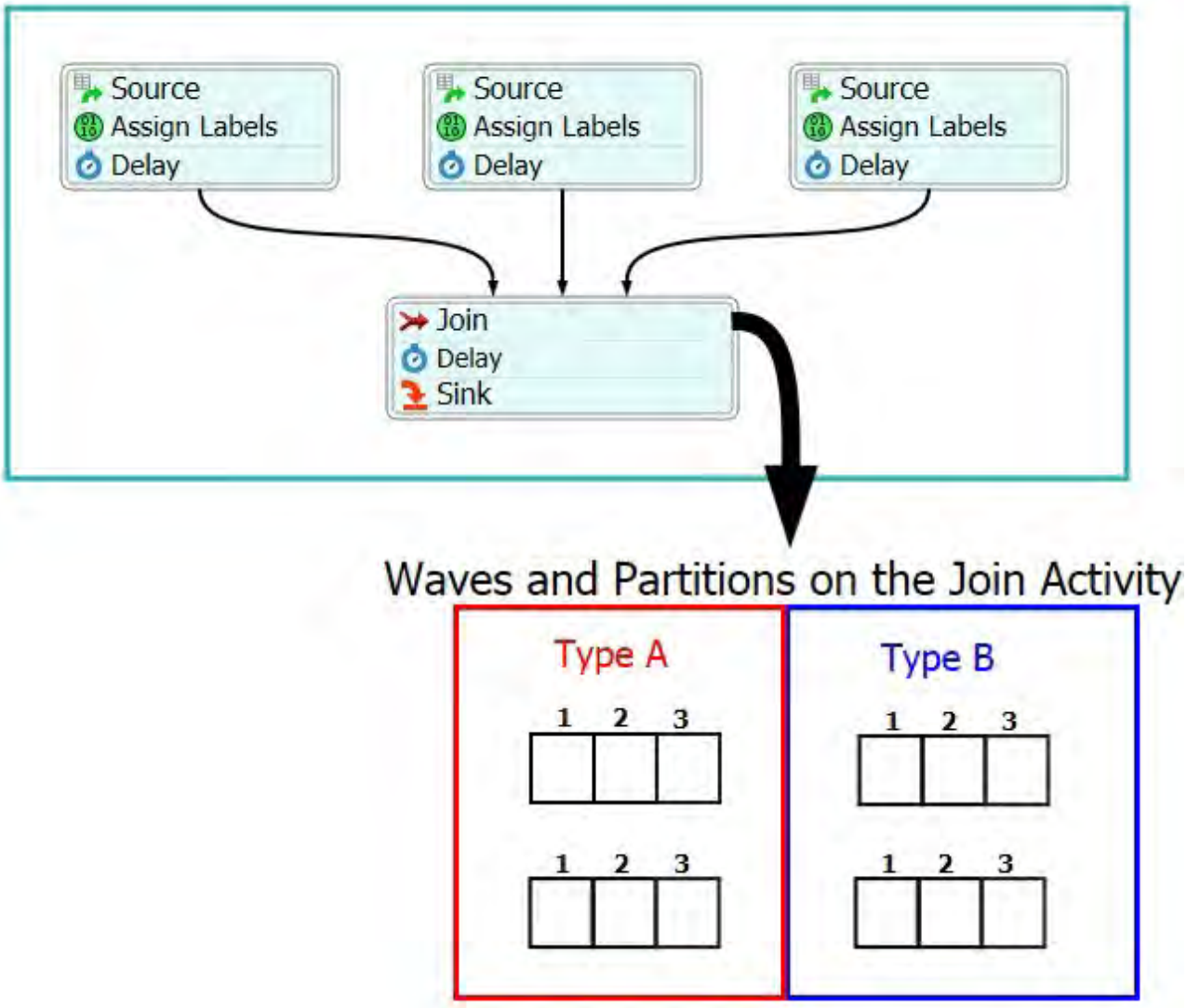
As was discussed in the previous sections, the Join and Synchronize activities can have any number of incoming connectors. However, the number of connectors these activities have will influence their behavior. Both of these activities will wait until it has a token in each incoming connector. When it has one token in each incoming connector, it will release these tokens in a *wave*. A wave is a group of tokens that are released at the same time.

If Join or Synchronize activities receive more than one token through one incoming connector, it will create additional waves. The following image visualizes what happens inside a Join activity as it receives tokens and creates waves:



The Join and Synchronize activities can also create waves with tokens that share a common attribute using a *partition ID*. A partition ID is defined by a label on the incoming tokens and it can be a number, string, or reference to an object or node. The Join and Synchronize activities can sort incoming tokens into waves based on data from a label on the tokens.

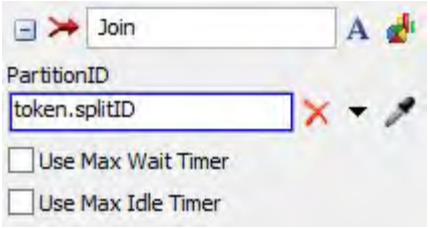
For example, the following image simulates a system with two different product types: A and B. Each incoming token has a label named *productType*. The Assign Labels activity assigns a value of either "A" or "B". When the tokens enter the Join activity, it assigns them to waves based on 1) which connector it came through, and 2) its partition ID (based on the *productType* label). It will release a wave after it has a token from each connector with matching partition IDs:



If needed, the values on the partition ID label can used mixed datatypes. In other words, it won't matter if some of the values are numbers and some of the values are strings or object references, etc. The Join and Synchronize activities will simply assign any tokens with a unique value to a new partition. For example, the Join and Synchronize activities will put tokens with a number value of 5 into one partition and it will put tokens with a string value of "five" in a different partition.

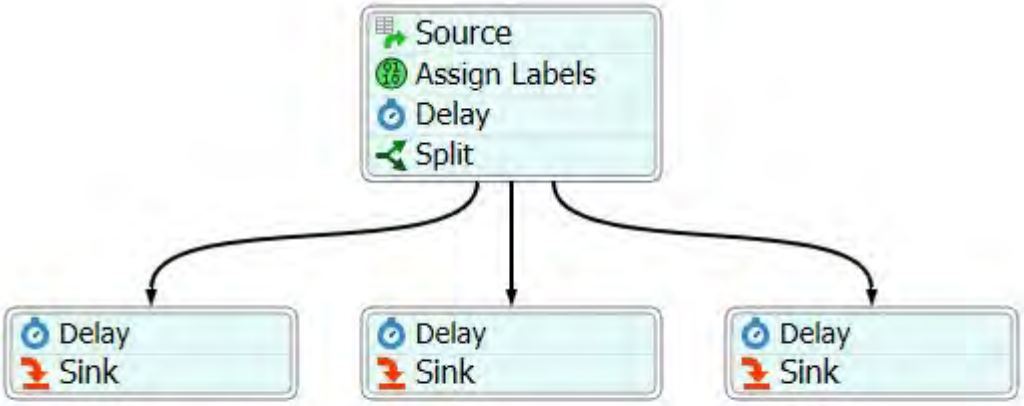
Defining a Partition ID

You can define which label the Join and Synchronize activities will use to create partitions using the PartitionID property. For example, the Join activity used in the previous image was set to use a the *productType* label in the PartitionID box:

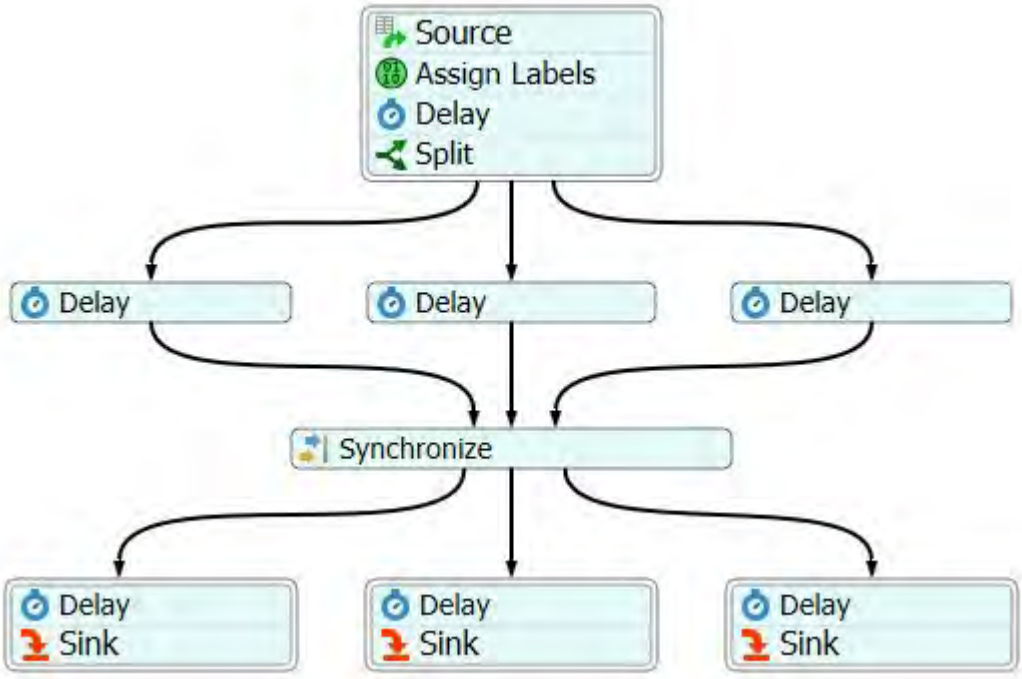


Split ID

The Split activity creates a Split ID on all tokens it creates. The Split ID is a label that contains a reference to the original token that was split. By default, this label is named *SplitID*, but you can customize that label if needed.



You can possibly use the Split ID label with the Join or Synchronize activities to make the tokens that were split wait for each other downstream in the process flow. For example, in the following process flow, a Split activity assigns tokens a Split ID. Then, the Synchronize activity puts incoming tokens into partitions based on the Split ID label:



Process Flow Preemption

This topic will discuss some of the key concepts related to preemption in Process Flow. It contains the following sections:

- What is Preemption?
- Process Flow Activities Used in Preemption
- How Preemption Works
- Key Concepts About Save Points
- Connecting Preempting Process Flows
- Using Save Token Context Activities as Checkpoints
- Using Preempting Activities with Time Tables

What is Preemption?

In Process Flow, *preemption* is when you stop a token's current activity and redirect it to a sequence of activities that take precedence over other activities. After the token completes the preempting activity sequence, it can possibly resume its previous activities. Preemption has several possible applications, such as:

- Machine Availability or Scheduled Maintenance - It's possible that not every section of your business system is available 24 hours a day, 7 days a week. Some sectors of the business might close down at regularly scheduled times. For example, some sets of machines might only operate during regular business hours. You could use preemption to simulate these scheduled closures. At closing time, the process flow could trigger the preempting sequence of activities. Then when the machines reopen the next day, they could resume their work from the previous day.

- Routine or Periodic Tasks - It's possible that your employees have to perform tasks at regularly scheduled periods of time. For example, perhaps your employees have to attend a weekly staff meeting or take a break every four hours. It's also possible that if you have employees who work in shifts, you might have a certain set of tasks that employees need to do during each shift change. You can use preemption to begin the activities that need to be performed at periodic intervals, then have the employees resume their previous tasks after they finish the preempting activities.

- High Priority Situations - Perhaps your business system receives an expedited order or an urgent event occurs that needs immediate attention. You can use preemption to divert business resources to handle the situation. When the situation has been resolved, the resources that were diverted can resume normal operations.

- Machine Breakdowns - If you wanted to simulate the occasional breakdown of machines in your business system, you could use a random statistical distribution to trigger the preempting activities that would need to occur during a breakdown. After the breakdown has been resolved, the machine can resume normal business operations.

Machine Breakdown Logic
You can use the standard MTBF/MTTR objects in the toolbox for generating the events that simulate machine breakdowns. Then you could set up your Process Flow to either listen for events on the MTBF/MTTR or for the onStop and onResume events on objects. See MTBF/MTTR for more information.

Process Flow Activities Used in Preemption

Process Flow preemption involves three key actions. Each corresponds to a Process Flow activity, which you will likely use in this exact order when building a preemption process flow:

1. Save Token Context - Before preempting, you need to save the token's current context in the process flow so that it can possibly return to this context after it finishes the preempting activities.

2. Release Token - This action aborts the token's current activity and releases it to a new activity, either to do something else, or to just wait.

3. Restore Token Context: After completing the operation that required the preemption, the token can be restored back to its saved context.

Each of these activities has a Token(s) property that defines the target tokens, meaning the tokens that will be preempted. When you build your preempting process flow, the Token(s) property on each of these three activities need to refer to the same tokens. The recommended method is to use an Assign Labels activity to create a common label that can be referenced by the Token(s) property on each preemption activity. The following sections about How Preemption Works and Connecting Preempting Process Flows will explain this in more detail.

How Preemption Works

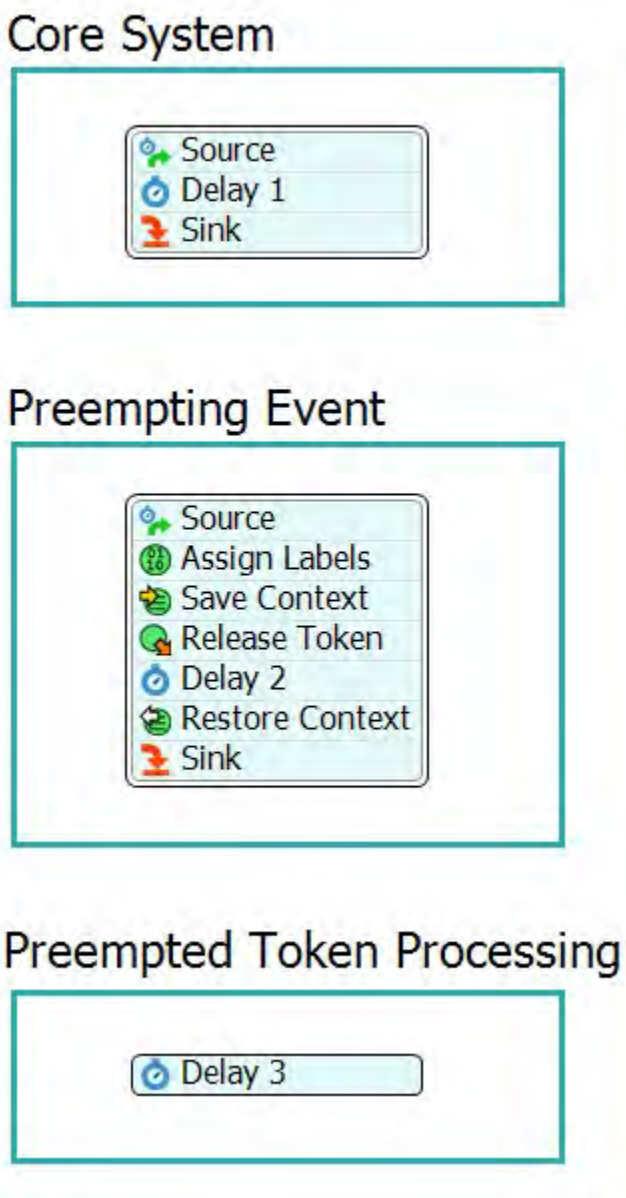
Typically in any process flow that uses preemption, you would have these three elements:

- Core System Process Flow - This process flow represents how your business system operates under normal working conditions.

- Preempting Event Process Flow - This process flow triggers the preempting sequence of activities. The event can be triggered at random, by an event in the simulation model, or it can happen at regular intervals using any of the Token Creation (Source) activities. The Preempting Event Process Flow will run parallel to the Core System Process Flow.

- Preempted Token Processing - You need at least one activity that temporarily stores the tokens that have been preempted until they can be restored to the Core System Process Flow.

The following image shows an example of a very simple preemption system with each of these three key elements:



How the Core System Works

The Core System works like any normal process flow. In the specific example illustrated in the preceding image:

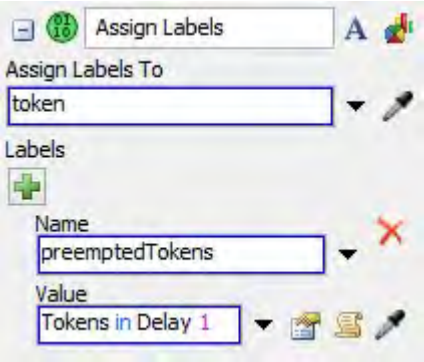
1. An Inter-Arrival Source creates a token every 20 seconds.
2. A Delay activity (Delay 1) then delays the token for 20 seconds, representing the processing time for the token.
3. A Sink activity removes all tokens that have been successfully processed in Delay 1.

How the Preempting Event Works

The preempting process will control when the preempting event occurs. It will also include any activities that are required as part of the preemption process. In the specific example illustrated in the preceding image:

1. An Inter-Arrival Source creates a token randomly using a statistical distribution within a range of 90100 seconds. In other words, a preempting event will occur randomly every 90-100 seconds.

2. An Assign Labels activity assigns all incoming tokens a label called *preemptedTokens* that refers to all tokens in the Delay 1 activity. See Connecting Preempting Process Flows for more information.



3. A Save Token Context activity saves the context of all tokens referenced by the *preemptedTokens* label. In this case, it will save the context of any tokens that are in the Delay 1 activity. The activity creates a label on the token called *savePoint* which will store the token's context. See Key Concepts About Save Points for more information.

4. A Release Token activity sends any tokens referenced by the *preemptedTokens* label to the Preemption Token Processing. In this case, it means that any tokens in the Delay 1 activity will be sent to the Delay 3 activity.

5. A Delay activity delays entering tokens for 40 seconds. This represents the amount of time it will take for the preempting process to finish. You could possibly add additional activities (such as task sequences) instead to control the logic of the simulation when the preempting event occurs.

6. A Restore Context activity restores the preempted tokens in Delay 3 to the exact state at which they were saved in the Delay 1 activity by the Save Token Context activity (in step 3). It uses the savePoint label to restore the context.

7. A Sink activity removes the tokens that triggered the preempting event.

How the Preempted Token Processing Works

The Preempted Token Processing simply stores or holds the tokens that have been preempted from the Core System. In the specific example illustrated in the preceding image:

1. During a simulation run, when a token enters the Release Token activity in the Preempting Event process flow, the token in the Core System process flow will disappear from the Delay 1 activity .

2. That same token will appear in the Delay 3 activity in the Preempted Token Processing process flow.

3. The token will remain in the Delay 3 activity until a token enters the Restore Context activity in the Preempting Event process flow.

4. The token will then return to the Delay 1 activity in the exact state it was in before. For example, if it had been delayed for 10 seconds (50% of the delay time), the delay would resume at 10 seconds.

Be aware that you could possibly make your Preempted Token Processing process flow more elaborate by adding additional activities to it.

The Preemption System in Action

The following image shows this example preemption system in action during a simulation run:

Preventing New Tokens from Entering

Notice that in the preceding example, new tokens can still enter the Core System process flow even while tokens are preempted away from the target activity. If you want to prevent new tokens from entering the target activity during the preempting event, consider using a Zone shared asset to restrict access or a Decide activity that checks for certain conditions before diverting tokens to an activity.

Key Concepts About the SavePoint Label

The Save Token Context and Restore Token Context activities both use a label called *savePoint* on the target tokens:

- The Save Token Context activity creates the *savePoint* label on the preempted token and saves which activity it was in when it was saved. If the activity is time-sensitive (such as a Delay or any activity that uses a Max Waiting timer or Max Idle timer), it will also save how long the token was in that activity.
- The Restore Token Context activity references the *savePoint* label on the preempted token and restores it to the activity that it was in when the label was created. It will also restore it to the specific time it was in the activity if it was a time-sensitive activity. For example, if a token had been in a Delay activity for 10 seconds when its context was saved, it will resume the Delay at 10 seconds when it is restored.

See Using Labels in Process Flow for other key concepts related to labels in the Process Flow module.

The *savePoint* Label Does Not Save Token Relationships

The *savePoint* label does not contain any information about the relationship between preempted tokens. For example, if 5 tokens were all preempted at the same time, the *savePoint* label does not record that these tokens are connected in any way. It doesn't save them to a special group or collection or list of tokens that were preempted together. The *savePoint* label only contains information about each individual token itself.

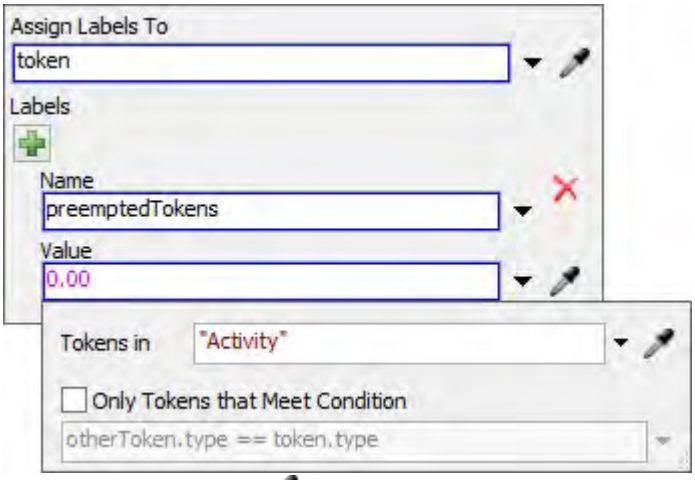
Connecting Preempting Process Flows

The Assign Labels method described in this section is the recommended method for connecting preempting process flows to the core process flow. This section will also briefly mention alternative methods.

Assign Labels (recommended)

The recommended method is to use an Assign Labels activity to create a common label that can be referenced by the Token(s) property on each preemption activity:

1. Add an Assign Labels activity before the preempting activities in the Preempting Event process flow.
2. With the Assign Labels activity selected, in Quick Properties, click the Add button to add a new label.
3. Give the new label a name in the Name box, such as *preemptedTokens*.
4. Click the arrow next to the Value box to open a menu. Point to Token then select Tokens in an Activity to open a picklist, as shown in the following image:



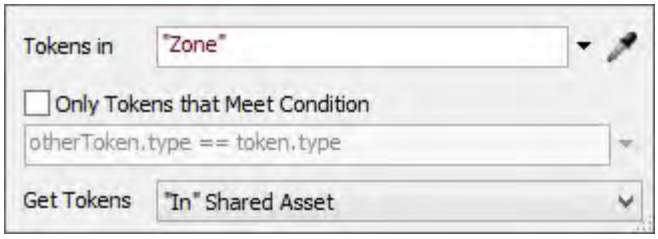
5. Use the Sampler button to select the target activity in the Core System process flow, meaning the activity from which tokens will be preempted away from the Core System.

6. In the Quick Properties for the Save Token Context activity, you'll need to point to the new label you just created. Click the arrow next to the Token(s) box to open a menu. Point to Token Label and select the name of the label you added in step 2. The following image gives an example:

7. Repeat steps 4-6 for the Token(s) property on the Release Token and the Restore Context activities.

You could modify this method to preempt all the tokens in a shared asset, such as a Zone, List, or Resource. This modification is ideal if you have an entire section of activities and you want to save all tokens anywhere in that Zone, List, or Resource. To preempt tokens in a shared asset:

- For step 4, select Tokens in a Shared Asset from the menu instead. This will open a similar picklist, as shown in the following image:



- For step 5, use the Sampler button to select the target shared asset, meaning the Zone, List, or Resource from which tokens will be preempted away. After that point, you can follow the rest of the steps as listed.

See the next section for an additional method for connecting activities.

Alternative Methods

Although the Assign Labels method is the recommended method for connecting preempting process flows, you could possibly connect the preempting activities directly to the specific activity or shared asset from which you want to preempt tokens by sampling it:

1. In the Quick Properties for the Save Token Context activity, click the arrow next to the Token(s) box to open a menu. Point to Tokens in Activity or Tokens in Shared Asset to open a picklist.
2. Use the Sampler button to select the target activity or shared asset in the Core System process flow.
3. Repeat steps 1 and 2 for the Release Token activity.
4. You could possibly repeat steps 1 and 2 for the Restore Token Context activity or you could sample one of the activity or shared assets that is being used in your Preempted Token Processing process flow.

The following image shows an example of the properties for the preempting activities using this method:

One of the reasons this method is not recommended is that you have to be very sure that the target tokens are the same for all activities, or else you could get error messages when you try to restore the tokens to their previous context. These error messages can be caused when new tokens enter the Core System process flow while other tokens are currently preempted. For example, if your Restore Token Context activity points directly to a Zone and it tries to restore all the tokens in that Zone, any new tokens that have entered the Zone will create an error (because they don't have a *savePoint* label).

That being said, you might have valid reasons for using this method. If you do, just keep the following guidelines in mind:

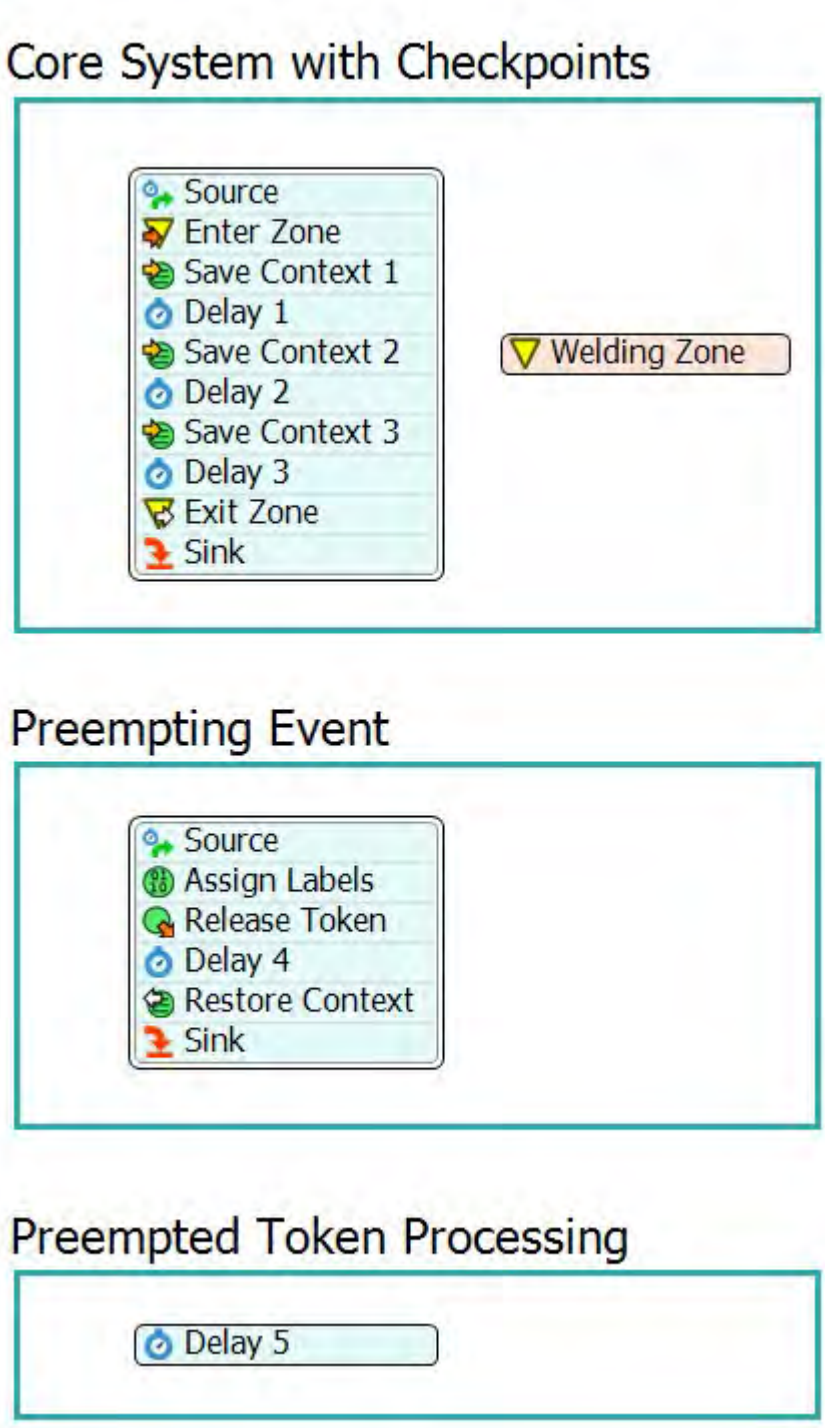
- Make sure you clearly understand what the *savePoint* label does and does not do. In particular, this label does not store any information regarding what groups of tokens were preempted at the same time. See Key Concepts About Save Points for more information.

- Make sure that the set of tokens that you restore is the exact same as the set of tokens that were saved and released.

Using Save Token Context Activities as Checkpoints

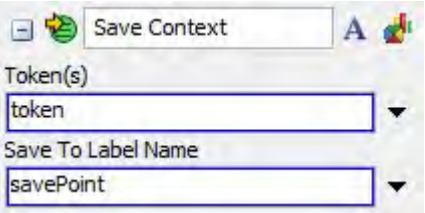
It's possible that when you restore a token to the Core System, you don't want it to resume in the exact place where it was preempted. Instead, you might want it to repeat a series of activities after certain checkpoints (or milestones). You could use Save Token Context activities to create these checkpoints.

The following image shows an example of a preemption system that uses checkpoints:

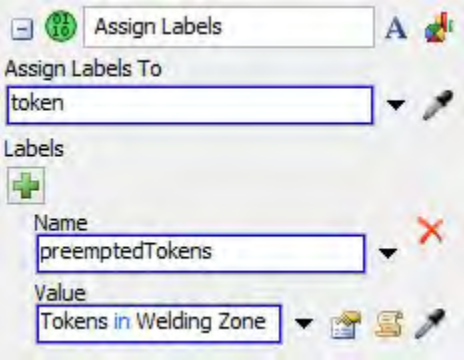


This system is similar to the system illustrated in the How Preemption Works example with a few key differences:

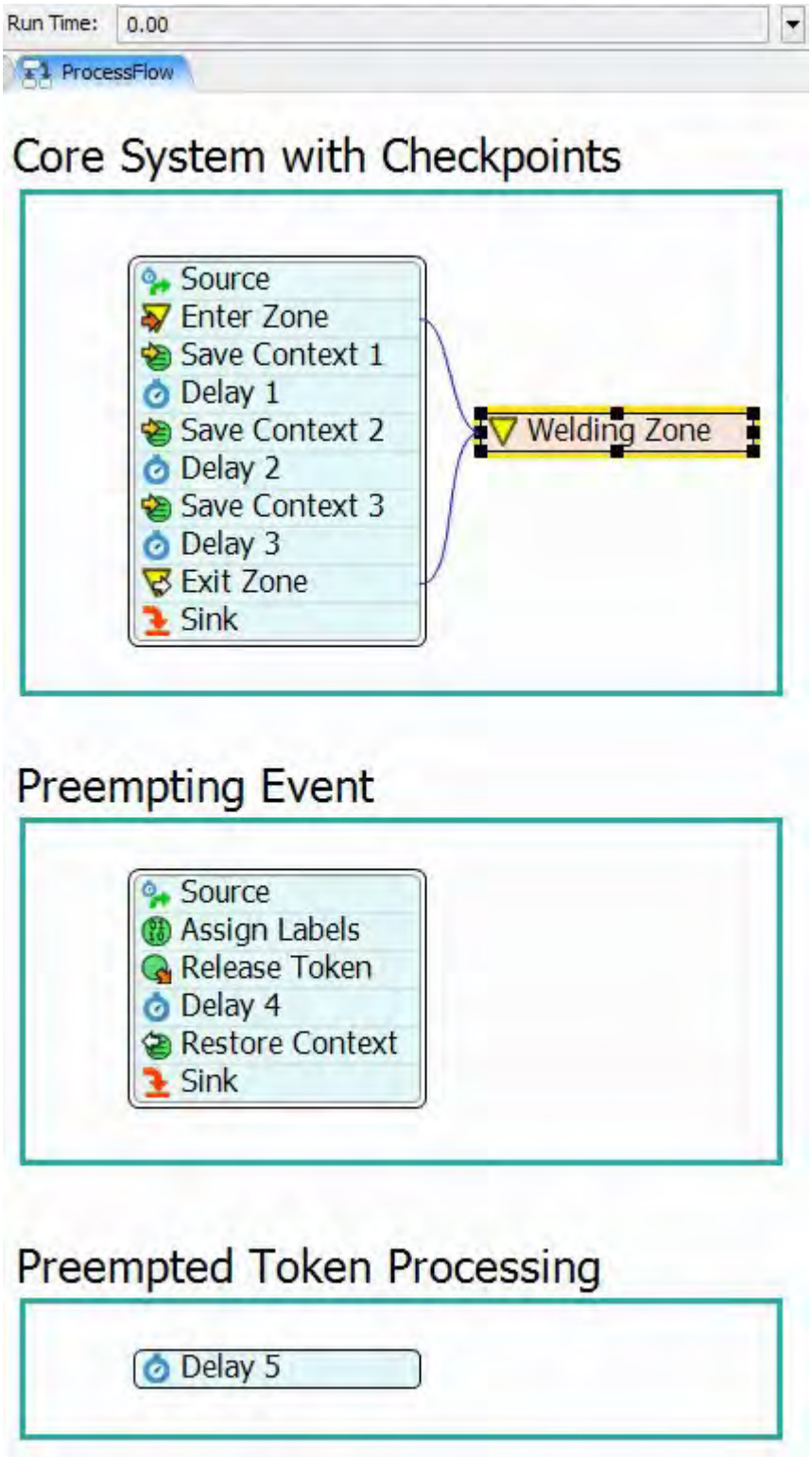
- The Core System process flow uses a Zone shared asset (named *Welding Zone* here), as well as Enter Zone and Exit Zone activities.
- The Core System process flow has three Saved Token Context activities, each one followed by a Delay activity (which could represent different stages at the Welding Station). The Saved Token Context activity will save the context for all entering tokens.



- The Assign Labels activity in the Preempting Event process flow refers to the tokens in the Welding Zone.



The following image shows this example checkpoint preemption system in action during a simulation run (the Zone is highlighted to illustrate the token relationships):



Using Preempting Activities with Time Tables

Preempting activites can be very useful when dealing with Time Tables and MTBF/MTTR objects. These tools give you control of setting an object or set of objects into a down state for maintenance, repairs or operational/non-operational times. By listening to events in these tools and performing the appropriate logic, you can cause your process flow to 'pause' what its doing and even start an additional process during the down time.

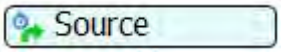
Process Flow preemption involves at least two tokens, but may include more. There are one or more tokens that are performing the core activities of the process; we'll call them the core tokens. Additionally there is a token that is triggering and managing the preemption; we'll call it the preemption token. The preemption token waits for, or is created by, some event that triggers preemption. When this event happens, the preemption token saves the context of all the core tokens. This saved context includes the core token's current activity, as well as any activity-related data. For example, if a core token is in a delay activity, it will save off the total delay time as well as how far into the delay the token is currently. The context is saved on a user-defined label on the core token (not on the preemption token).

Once the preemption token has saved the context on all the core tokens, it releases those tokens to a different activity. If the preemption is meant to just "freeze" the process, core tokens would be released to an activity that does nothing, such as an 0 delay activity with no successor activities. Core tokens will just sit in that activity until the preemption token restores them back. This is the implementation shown in the above image, where the preemption token releases the token to a *Preemption Silo*. On the other hand, if the preemption needs to do an "ordered take down", core tokens could be released to an activity where they perform the ordered take down, such as releasing resources, exiting zones, etc. The following image shows the preempted core token in the *Preemption Silo*.

Once the preempting operation is completed, core tokens can be restored back to their saved contexts. They will continue from the exact point where they were saved.

Inter-Arrival Source

The Inter-Arrival Source activity creates new tokens according to a specific interval of time. Similar to the Inter-Arrival Time arrival style of FlexSim's standard Source, you can use a fixed number to set an exact interval of time between token creations or you can use a statistical distribution to randomly calculate the time between arrivals. Once a token is created it will be released to the next activity.

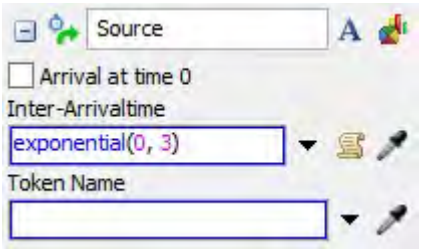


Connectors

The Inter-Arrival Source activity only allows one connector out. See Adding and Connecting Activities for more information.

Properties

The following image shows properties for the Inter-Arrival Source activity:

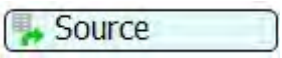


Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button A opens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- Arrival at time 0 If checked, a token will be created at time 0 when you start the model run.
- Inter-Arrival Time This defines the interval of model time between token creations.
- Token Name For debugging purposes, you can assign a name to the tokens created by this source. For example, if you used the name *Token1* in this box, all of the tokens created by this source would be assigned this name. Token names are not required so the box is left empty by default.

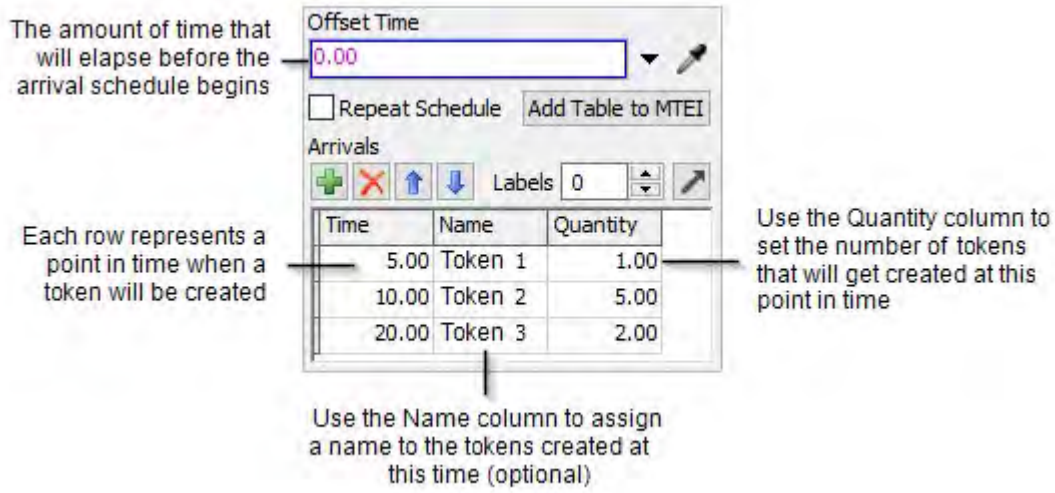
Schedule Source

The Schedule Source activity creates new tokens as specified in its Arrivals table. This table defines the time (in model units) that tokens should be created, the name that will be assigned to the new tokens and the number of tokens to create. These settings can either have a fixed value or the values can be calculated dynamically. Also, you can either enter the data into this table manually or import a table from an Excel spreadsheet using the Excel Interface.



Overview of Arrival Schedules

The following image shows an example of a possible arrival schedule on a Schedule Source activity:



The Offset Time can be used to make a certain amount of time elapse before the Schedule Source activity will begin the arrival schedule. In the example used in the preceding image, the Offset Time is set to 100, which means that schedule won't begin until the simulation clock reaches 100 during a simulation run.

For now, skip over the Repeat Schedule checkbox, which will be discussed in a moment.

Each Schedule Source activity has an Arrivals table, which controls the schedule for creating new tokens. Each row in the table represents a point in time that one or more new tokens will be created. The values for the cells under each column can be changed to customize the arrival schedule:

- Time - The time when one or more tokens will be created, relative to the simulation model clock and the offset time. In this example, if the offset time is set to 0, the first token will be created when the model clock reaches 5, the second set of tokens will be created when the clock reads 10, and so forth. However, since the offset time is currently set to 100 in this example, the first token will actually be created when the clock reads 105, the second set of tokens will be created when the clock reads 110, and so forth.

- Name - For debugging purposes, you can assign a name to the tokens created at this point in the schedule. In this example, the first token that is created will be named *Token1*.

- Quantity - The number of tokens that will be created at this point in the schedule. In this example, only one token will be created when the model clock reaches 105, then five tokens will be created when the model clock reaches 110, and so forth.

With that in mind, let's return to the Repeat Schedule checkbox. If the schedule is set to repeat, it will loop indefinitely. The offset time will NOT be repeated. In the example illustrated in the preceding image, the first token will be created at 105 (because of the offset of 100). Then five tokens will be created at 110 and two tokens at time 120. Then the schedule will repeat itself. One more token will be created at 125, five tokens at 130, and two tokens at time 140, and so forth. This process will last indefinitely until the simulation clock is stopped.

If you want to create a gap between repeats, you can add an arrival at the end of the list with a quantity set to 0. This will cause no tokens to be created at that time. In this example, adding an arrival time of 40 with a quantity 0 will make the arrival times be: 105, 110 and 120 then after the first repeat the arrivals will be 145, 150 and 160.

Dynamic Arrival Values

In the above example, all of the arrival times, token names and quantities were defined as static values. However, each of the values in the arrival table may also be defined dynamically using picklists or code. This gives a greater amount of control on how an arrival schedule is defined. Keep in mind when calculating the time of an arrival, the value should be a number greater than the previous arrival where arrivalTime = time - previousTime + modelTime;

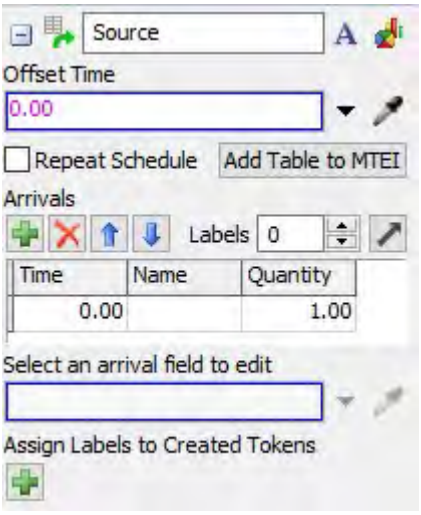
Entering Dynamic Time
If the Time value is defined dynamically, it's important to ensure that the times returned increase for each arrival. Times that are less than or equal to previous arrival time will create items in 0 time.

Connectors

The Schedule Source activity only allows one connector out. See Adding and Connecting Activities for more information.

Properties

The following image shows properties for the Schedule Source activity:



Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button A opens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- Offset Time Can be used to make a certain amount of time elapse before the Schedule Source activity will begin the arrival schedule. For example, if the first entry in the arrivals table has a time of 20, and the offset time is 10, the first token will be created at a simulation time of 30.

- Repeat Schedule If the Repeat Schedule box is checked, the schedule will repeat indefinitely during the simulation run. Note that the Offset Time will not be repeated each cycle, but will only be used in calculating the start of the schedule. For example, if the last entry in the Arrivals table is at time 50 and the offset time is 10, the first cycle will end at simulation time 60. The second cycle will end at simulation time 110.
- Arrivals This is where the arrival schedule is defined. (See Overview of Arrival Schedules for more information.) Rows can be added by clicking the button, deleted by clicking the , and reordered by clicking the and buttons. The table can also be added to the MTEI allowing you to import data from Excel.

  There are three default columns in the table:

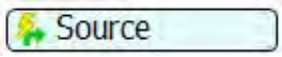
  - Time - The time when one or more tokens will be created, relative to the simulation model clock and the offset time. (See Overview of Arrival Schedules for more information.)
  - Name - For debugging purposes, you can assign a name to the tokens created at this point in the schedule.
  - Quantity - The number of tokens that will be created at this point in the schedule.
  - Labels - Label columns can be added or removed by changing the number in the Labels field. Each token will be given labels with names corresponding to the column headers and values corresponding to the value of the table cell.

  You may define static values through the table or dynamic values using the picklist field directly below the table. See Dynamic Arrival Values for more information.

  The table can be opened in a larger popup window by clicking the button.
- No Selection This field will change based upon which cell is currently selected in the Arrivals table. You can use this box and the pull-down arrow next to it to enter dynamic values for the Time, Name, or Quantity values in the table. You can select a picklist option from the pull-down menu or you can enter in custom code if you are comfortable with FlexScript.
- Assign Labels to Created Tokens Allows you to add labels to the created tokens based upon the arrival entry. For more information on assigning labels, see the Assign Labels activity.

Event-Triggered Source

The Event-Triggered Source activity creates tokens in response to an event during a simulation run. When that event occurs, it will create a token. (See Event-Listening Activities for more information about the types of events that can be listened for.) If the event has data associated with it like a flowitem or port number, you have the option to save that data off as a label on the created token. If the data is an object like a flowitem, a reference to that flowitem will be saved on the token label for later use. You can also assign a name to the created token.



To get a deeper understanding of how the Wait for Event activity works, you might want to consider reading these topics first:

- Linking Process Flows to Simulation Models
- Key Concepts About Event-Listening Activities
- Event Types and Related Properties

As a more advanced feature, the Event-Triggered Source also has the ability to override the return value of the event it is listening to. For more information, see the Will Override Return Value.

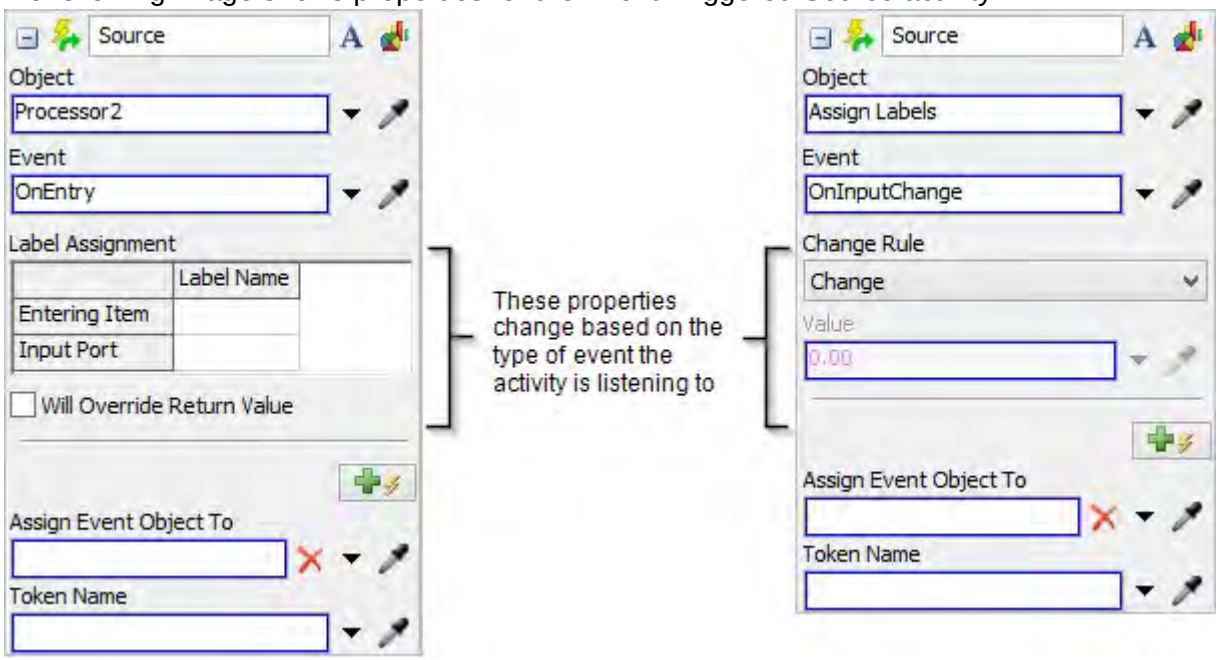
When this activity is first created, a red exclamation mark shows up to the right of the activity notifying you that a link to an event is required for this activity to function. This link may be a direct pointer which can be created by clicking on the exclamation mark and then clicking on an activity or object and selecting an event, or the reference may be set through the quick properties.

Connectors

The Event-Triggered Source activity only allows one connector out. See Adding and Connecting Activities for more information.

Properties

The following image shows properties for the Event-Triggered Source activity:



- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button opens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- ObjectUse the Object box to select the object that the Source activity will listen to. You can listen to events on:
    - An object in the model. o An activity in the Process Flow.
    - The current instance object (See Instances for more information.)

- A group of objects. In this case, the activity will listen to the defined event on all members of the group.

Use the Sampler button to select the object and choose which event to listen to. When you hover over an object with valid events, a list of those events will be shown. Selecting an event this way will change both the Object box and the Event box and will cause the interface to change based upon the event type.

See Event-Listening Activities for more detailed information about listening to objects and events.

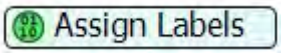
- EventUse the Event box to select the event that the Wait for Event activity will listen to. The Event box is connected to the object you selected in the Object box. When you use the Sampler button to select an object in the Object box, a menu will appear that lists all the different types of events that are available for that object. (This menu is created dynamically based on all the possible events related to that object.) The event that you select from this menu will automatically be listed in the Event box. You can change the event type by clicking the drop down arrow next to the Event box and selecting an event.

  See Event-Listening Activities for more detailed information about listening to objects and events.

- Label Matching/Assignment and Change RuleThe Label Matching/Assignment or Change Rule properties will appear after you've selected an object and event to listen to. See Event Types and Related Properties for more information about these sets of properties.
- Add Events You can listen to more than one event on an Event-Triggered Source. To add an event in addition to the original event, press the Add Event button . Additional events will have a Delete button button if you need to remove them.
- Assign Event Object To Assigns a reference to the event-firing object to the specified label or node. This is useful if you are listening to multiple objects (perhaps from a Group) and need to know which object fired the event. This box is not required and can be left empty.
- Token NameUse the Token Name box to determine the name that the newly created token will be given. Tokens are not required to have a name so this box is empty by default. You may give the token a static name like *Token* or it may be dynamic by using a picklist or through FlexScript. These names do not change the behavior of your Process Flow but can be useful for debugging purposes.

The Assign Labels activity creates or modifies labels on various objects. Labels can be used to store important data about various objects. You can assign labels to any object that owns labels which includes, but is not limited to:

- An incoming (entering) token
- A parent token
- Flowitems
- 3D objects such as an Operator or Processor
- A process flow



To define the labels to create/modify click the button under the Labels section. Labels may be numbers, strings, treenodes (objects) or arrays. If a label is set that does not currently exist on the Assign To object, the label will created. If the label already exists, then the value will be overwritten with the new value. treenode and array

When setting a label to a treenode or array (for example a Processor or Operator in the model), the value stored on the label is just a reference to the treenode or treenodes. The treenode is not physically being stored on the label.

Non-Label Assignment

The Assign Labels activity can also be used to add data to any node in the form of subnodes. If the Assign To is a label node or some other node in the model that is not an object with labels, then sub nodes will be added for each label entry. To reference a label node rather than the value, use the command label(token, "labelName"). This can be useful when storing large amounts of data that is not used to determine model logic. This functionality can be used to create a table on a token's label. To explore the table, right click on the label in Quick Properties and select Explore as Table.

Tracked Variables as Labels

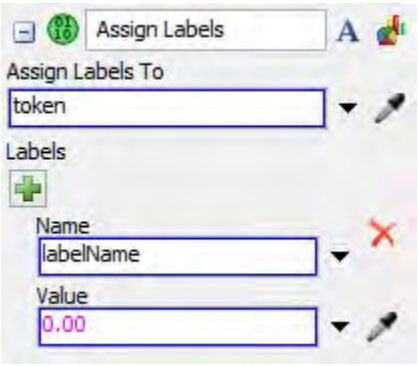
It is often useful to be able to track the history of a value as it changes throughout the model run. Or perhaps you want to wait until a value gets above or below a certain point. One way of doing this is by creating and setting a global Tracked Variable. Alternatively, if you select the pick option Add Tracked Variable from the Value boxes, a tracked variable will be created on the assigned label. This label will have the apperance of a label with numeric data but you have both the ability to store a history of all of the values as well as listen to whenever that value changes through a Wait For Event activity. Once a tracked variable is added to a label, the value can be updated using standard picklist options or the setlabel() command.

Connectors

The Assign Labels activity only allows one connector out. See Adding and Connecting Activities for more information.

Properties

The following image shows properties for the Assign Labels activity:



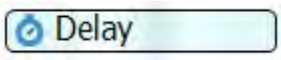
Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button opens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- Assign ToThe Assign To box determines which token or object will be assigned a label. By default this activity assigns labels to the entering token.
- LabelsWhen you click the Add button, a new set of properties will appear:

  Name - The name of the label that will be created or modified on the Assign To object.

  Value - The new value of the label. You can enter a static number or string into this box or choose one of the picklist options from the drop down for a more dynamic value. Use the eye dropper to get references to objects, object labels or other data. Values entered here may be a number, string, treenode (object), array, or process flow variable. The value could also be defined using code.

The Delay activity will hold the token for a certain length of time. You can use a fixed time or you can create the delay time dynamically using a label value on a token, a statistical distribution, etc. NOTE:



Breathe

A *breathe*, which is a delay time of 0, is similar to sending a delayed message in 0 time. Doing so will create an event in 0 time before releasing the token. Breathes can allow you to breakup certain operations

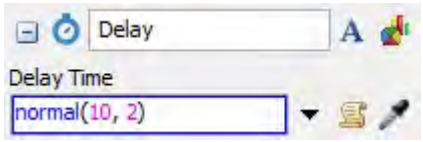
to ensure that things are happening in an appropriate order. This is like taking a breath before moving on. It may be difficult to know when this is needed, but if something in your model is not behaving properly, adding a 'Breathe' can potentially solve issues.

Connectors

The Delay activity only allows one connector out. See Adding and Connecting Activities for more information.

Properties

The following image shows properties for the Delay activity:



Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button opens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- Delay TimeThe Delay Time defines the amount of time that the token will be delayed.

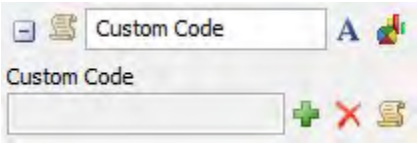
You can use the Custom Code activity to create custom behavior in the process flow module. You can select pre-defined picklist options or write your own code in FlexScript. When a token enters the Custom Code activity, it will evaluate the user-defined code and then immediately be released to the next activity without any model time passing.

Connectors

The Custom Code activity only allows one connector out. See Adding and Connecting Activities for more information.

Properties

The following image shows the properties that are available for the Custom Code activity:



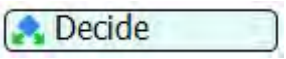
Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button A opens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- Custom Code Use the Custom Code box to choose the custom code that should be executed when a token enters this activity. You can use pre-defined code templates (picklists) or write your own custom code if you are comfortable with Flexscript.

To use a picklist, click the Add button next to the Custom Code box to open a menu listing different picklists that are currently available for this object. Try clicking on this menu and exploring some of the different options to see what's available.

Alternatively, if you are comfortable with Flexscript and want to enter in your own custom code, click the Code Editor button next to the Custom Code box to open the code editor. In the code editor, you would enter in the code you want the activity to execute every time a new token enters the activity.

The Decide activity will send a token to one of two or more possible activities based on conditions that you define. In other words, the Decide activity can determine the next activity that a token should be diverted to.



The Decide activity can use many different methods to determine which activity the token should go to next, such as:

- Sending the tokens to the next activities in round robin order
- Assigning the token to the activity that has the longest or shortest queue (based on the number of tokens that are waiting at that activity)
- Diverting a certain percentage of tokens to one activity while another percentage goes to a different activity
- Sending a token to various activities or outcomes based on a value in a label on the token
- Assigning tokens to one of the next activities at random

You can also use the Decide activity to create looping functionality. After evaluating a condition or a label on a token, the Decide could send the token back to the top of the loop or cause a token to exit a loop.

Connectors

The Decide activity allows any number of connectors out. The Decide does not allow other activities to be snapped to it as doing so would remove all connectors from the Decide. See Adding and Connecting Activities for more information.

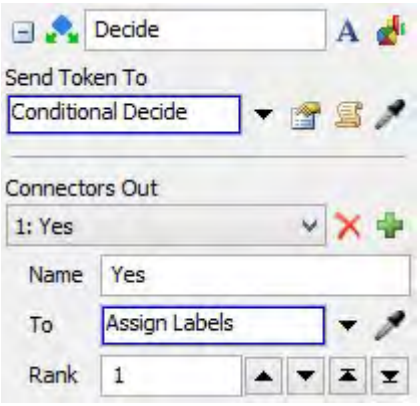
You will need to understand how connectors work in order to use a Decide activity. The following list explains a few important concepts to keep in mind:

- Each outgoing connector has a number (also sometimes called a *rank*) based on the order in which you created them. When you create an outgoing connector from the Decide activity to another activity, this connector will automatically be assigned a number (rank). For example, the first outgoing connector you create will be assigned a rank of 1, the second will have a rank of 2, and so forth. You may have noticed that this is similar to how port connections work between objects in the 3D simulation model.
- You can re-rank outgoing connectors. You can use the properties in the Decide activity or on the connector to change the ranks of the outgoing connectors. You can also disconnect the connector from the Decide activity and then reconnect it to move that connector to the last rank.
- Connectors can have names. You can also assign a name to a connector for a more visual reference if needed. (Double-click a connector to create a name). If the Decision property returns a string (text), the Decide will search it's connectors for a connector with a name that matches the given string.

- Sending to a connector that doesn't exist. If the Decision property returns an invalid rank or a connector name that doesn't exist, the token will remain in the Decide activity and not be released. This can also be done by returning a negative number in the Decision property.
- Connectors are not required, but encouraged. Though visually it is easier to follow the flow of a token when connectors are leaving a Decide activity, they are not required. The Decision property can be given an activity to send to without being connected to it. This could be done by using the getactivity() command or storing a reference to an activity on a token label.

Properties

The following image shows the properties for the Decide activity:



Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button Aopens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- Decision You will use the Decision box to determine the logic that the Decide activity will use to send tokens to the next activities. When you click the arrow next to the Decision box, it will display a menu of many possible options, such as:
  - Conditional Decide - This option will test the token to see if it passes a simple test or not. If it passes, the token will be sent out one connector. If it fails, it will go out another connector. For example, you could test to see if the token has a particular value in one of its labels or not. If the label has that value, it passes. o By Percentage - Sends a certain percentage of the tokens out one connector, a certain percentage out another connector, etc.
  - Connector/Activity by Case - Sends tokens to a connector name/number or an activity based on certain variables, such as the value of a label.
  - Random Connector - Sends the token to any of the outgoing connectors at random.
  - Shortest/Longest Queue - Sends the token to the activity that has the longest or shortest queue

(based on the number of tokens that are waiting at that activity) o Round Robin - Each token will be sent to one of the outgoing connectors sequentially. For example, if there are 3 outgoing connectors, the first token will go to connector 1, the second to connector 2, and the third to connector 3. This will then repeat sending the next token out connector 1 and so forth.

Valid Return Values

Whatever logic you decide to use, keep in mind that the logic needs to have a valid return value. The return

value tells the Decide activity where to send the token. The return value can only be valid if it is the index number for a connector, the name of a connector, or a reference to an activity. Returning a value of 0 is the same as returning a 1 and will go out the first connector. Returning any negative value will cause the token to not be released and remain in the Decide activity.

- Connectors Out You can use the Connectors Out properties to add, edit, or delete the Decide activity's outgoing connectors.

  The Connectors Out menu displays all the outgoing connectors that currently exist for the Decide activity. The connector will be listed by its connector rank and name. (If you haven't assigned a name to the connector, the menu will only display its connector rank.)

  If the Connectors Out menu is blank, that means that you have not created any outgoing connectors yet. Click the Add button  to create a new connector to an activity. The Delete button  will remove the connector that is currently selected in the Connectors Out menu.

  After selecting a connector from the Connectors Out menu, you can edit its following properties:

  - Name - Use the Name box to give a name to the currently-selected connector. Connectors are not required to have names, but can help make your Process Flow chart more readable and allow you to reference connectors by name.
  - To - The To box displays the name of the activity that the currently-selected connector is connected to. NOTE: By default, identical activity types have the same name. For example, all Delay activities are called *Delay* by default. If you need to distinguish between the different activities, you might want to rename them for clarity.
  - Rank - The Rank box lists the rank (connector number) of the currently-selected connector. The ranks are assigned by the order in which you created the connector. You may change the rank by typing a new value into this box or by using the up and down arrows.

Batch

In the Process Flow module, the Batch activity collects incoming tokens and sorts them into groups of tokens (batches). When a batch is ready, the Batch activity will release it to the next activity.

The Batch activity basically functions similarly to Combiners and Separators in the 3D model in that it is capable of both combining and separating tokens.

Using the Batch activity for simple grouping and releasing is relatively straightforward. However, the Batch activity has some additional features that allow you to collect, sort, and release batches in fairly complex ways. If you want to learn more about what the Batch activity is really capable of, consider reading the following topics:

- Organizing Batches - Explains the various ways you can collect and group incoming tokens into batches.
- Releasing Batches - Discusses the options for releasing batches and how to assign labels to an outgoing batch. This topic will also discuss options for handling batches that collect more than a batch can hold.
- Batch Statistics - Explains how to use the unique batch statistics to get useful data from the Batch activity during a process flow simulation.

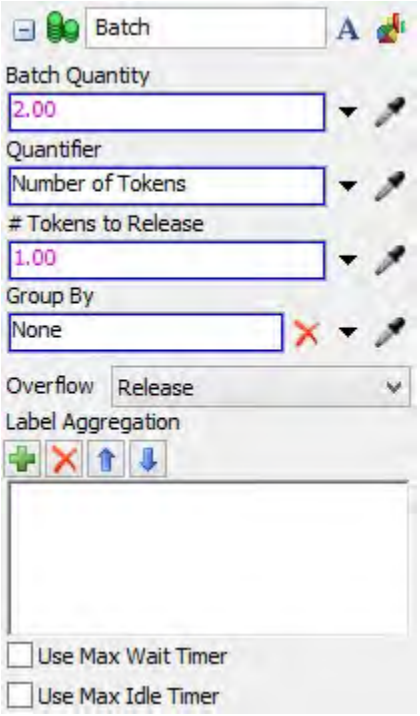
The rest of this topic is a reference page for the Batch activity and its properties.

Connectors

The Batch activity allows for any number of connectors out. However, batches that complete will always exit out the first connector. Only manually released batches have the opportunity to exit out a different connector. See Adding and Connecting Activities for more information.

Properties

The following image shows properties for the Batch activity:



Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button opens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- Batch Quantity The Batch Quantity box sets the target value for the batch. The batch will continue collecting tokens until this value is reached, after which the batch will be released. See Organizing Batches for more information.
- Quantifier The Quantifier box sets the criteria that will determine how the batch quantity is calculated. You can either collect by *Number of Tokens* or by a label on the incoming tokens. See Organizing Batches for more information.

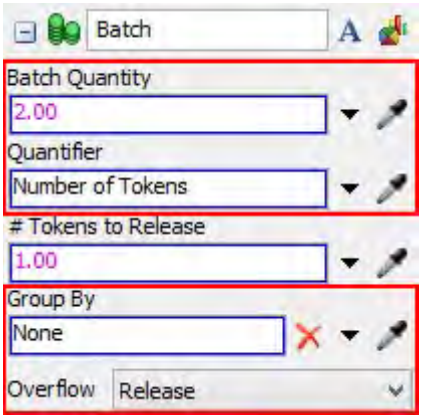
  Valid Quantifier Values
  The Quantifier property must be either *Number of Tokens*, a string which will be interpreted as the label name on the entering tokens, or a pointer to the token label (using token.aLabel).
- # Tokens to Release Specifies the quantity of tokens that the batch will release once it has finished collecting. See Releasing Batches for more information.
- Group By The Group By box sets the criteria by which incoming tokens will be grouped into batches. See Organizing Batches for more information.
- Overflow The Overflow setting controls what happens to values that exceed the Batch Quantity while collecting. This option is not available when collecting by the *Number of Tokens* See Organizing Batches - Batch Overflow Options for more information.
- Label Aggregation Using this set of properties, you can create and modify labels on tokens when a batch is released. The operation performed, as set in the Aggregation column, will be done on every token that was added to the batch, even if the token is destroyed when the batch is released. Once the operation is

performed, the resulting value will be set on every token being released by the batch. See Releasing Batches - Assigning Labels to Outgoing Batches for more information.

- Use Max Wait Timer The Max Wait Timer will be evaluated when a batch is first created. If a timer is not created because the Start Criteria evaluates to false, the max wait timer will be evaluated each time a token is added to the batch until the Start Criteria does evaluate to true. See Releasing Batches - Releasing a Batch Early" for more information.
- Use Max Idle Timer The Max Idle Timer will be evaluated each time a token is added to a batch. See Releasing Batches - Releasing a Batch Early" for more information.

Organizing Batches

The Batch activity can collect entering tokens and organize them into batches based on the criteria you specify. The following properties on the Batch activity are directly involved in how batches organize tokens:



- Batch Quantity - This property sets the target value for the batch. The batch will continue collecting tokens until this value is reached, after which the batch will be released.
- Quantifier - This property sets the criteria that will determine how the batch quantity is calculated.
- Group By - This property sets the criteria by which entering tokens will be organized and grouped into batches.
- Overflow - This property determines what will happen if an entering token will cause the batch to exceed its Batch Quantity.

With that in mind, the Quantifier has two different methods for collecting and organizing tokens:

- Batch by number of tokens - If you select this option (which is the default), the Batch Quantity will be based upon the number of tokens that arrive at the Batch activity. Each entering token will increase the current quantity of the batch by one. When the number of tokens reaches the value specified in the Batch Quantity property, the batch will be released.
- Batch by a label on the tokens - If you select this option, the Batch Quantity will be based upon the value of a label on the entering tokens. In this case, a token can increase the batched quantity by any numeric value stroed on the label.

Each of these methods will be explained in the following sections. This topic will also cover:

- Batch Overflow Options - Explains how to control what happens if an entering token's label exceeds the specified Batch Quantity.
- Grouping - Describes how to use the Group By property to group entering tokens into different batches allowing you to collect multiple batches simultaneously.

Batch by the Number of Tokens

The simplest (and default) method of batching is to collect by the number of tokens. This means that entering tokens each take up one space in their batch with respect to the Batch Quantity. When the number of tokens reaches the value specified in the Batch Quantity property, the batch will be released.
To set the Batch activity to collect by number of tokens:

1. Click the Batch activity to select it.
2. In Quick Properties, check that the Quantifier box says *Number of Tokens*.
3. Click in the Batch Quantity box and type the number of tokens that should be collected in each batch.

See Releasing Batches for more information about what happens when a batch is ready for release.

Batch by a Label on the Tokens

If you select this option, the Batch Quantity will be based upon the value of a label on the entering tokens. In this case, a token can increase the batched quantity by any numeric value listed in the label. For example, if the Quantifier is set to `token.Weight` and the entering token has a label named *Weight* with a quantity of 200, the token will increase the current batch's quantity by 200 when it is added to the batch.
To set the Batch to collect by label values:

1. Make sure an Assign Labels activity or some other activity earlier in the process flow will create the label that will be used for batching.
2. Click on the Batch activity in your process flow to select it.
3. In Quick Properties, click the arrow next to the Quantifier box to open a menu. Point to Token Label, which will open up a submenu listing all the labels that currently exist in the process flow. Select the desired label from the menu by double-clicking it or using the arrow and Enter keys.
4. The Quantifier box will now show the command `token.YourLabelName`. For example, if the label is named *Weight*, the box would read: `token.Weight`. Alternatively, you could type `token.Weight` into the box. You can also enter a string into this box using quotes. ie *"Weight"*
5. Click in the Batch Quantity box and type the target batch quantity. For example, if you wanted the Batch activity to collect tokens until it reaches a collective weight of 300 grams, you would type `300` here.

Now each time a token enters the batch, the value of the specified label on each token will be added to the current batch quantity. When the batch quantity reaches the specified limit, the batch will be released. For example, imagine a Batch activity is using a label named *Weight* to collect tokens and it has a specified Batch Quantity of 300, meaning it needs to collect a total weight of 300 in order to get released. If the first token that enters the Batch activity has a *Weight* label with a value of 200 and the second token's *Weight* label has a value of 100, the batch would get released after the second token is added to the batch. That's because the total sum of the *Weight* label for both tokens would be 300, which was the target weight for each batch.
See Releasing Batches for more information about what happens when a batch is ready for release.
What if the Label Value Exceeds the Batch Quantity?

When batching by a label, it is possible that the batched quantity may be greater than the specified Batch Quantity, which will cause the batch to overflow. See Batch Overflow Options in the next section for more information about how to control overflows.

Batch Overflow Options

When batching by a label, it is possible that the batched quantity may be greater than the specified Batch Quantity, which will cause the batch to overflow. When that happens, it will trigger the Batch activity's overflow logic.

Batching by Number of Tokens Won't Cause Overflows

If you are batch by number of tokens, your batch will never exceed the specified Batch Quantity and therefore won't trigger the overflow logic.

To illustrate what happens when a batch is going to overflow, let's return to the same example that was used in the previous section. As a reminder, in this example, the Batch activity is set to release the batch when the total sum of the *Weight* label is 300. If the first token's *Weight* label has a value of 200 and the second token's *Weight* label has a value of 150, the total sum of the *Weight* label for both tokens would be 350. 350 would exceed the target weight for that batch and it would therefore trigger the Batch's overflow logic.

When overflow occurs the Batch activity's behavior is governed by the Overflow property. This property has a menu with the following options:

- Do Not Divide, Release Excess - The batch is released without modification. If this option is selected, the batch would be released including its excess quantity. For example, if the Batch Quantity is set to 300 and an entering token would cause the batch to overflow by 50, the batch would be released with a quantity of 350. No token labels will be modified by the batch when using this option. Use the Label Aggregation method for modifying labels.

- Do Not Divide, Release Partial - Rather than release an overflowed batch, the Batch activity will release a batch that is only partially full. The entering token that would have caused the batch to overflow is put in the next batch. This logic can be illustrated using the same example in which the Batch Quantity is set to 300 and the current batch has a quantity of 200. An entering token has a value of 150, which would cause the batch to overflow by 50. When this happens, the current batch would be released with a quantity of 200 and the entering token would be placed in a new batch with a quantity of 150. No token labels will be modified by the batch when using this option. Use the Label Aggregation method for modifying labels.

- Hold Excess Quantity - The current batch will be released with the exact required quantity. A new token is then created and placed in a new batch with a label containing the excess quantity. All labels and label values are copied from the last token (the one that caused the overflow) to the new token. For example, if the Batch Quantity is set to 300 and an entering token would cause the batch to overflow by 50, the batch would be released with a quantity of 300. Then, a new batch would be started with a new, nearly identical token containing a label with a quantity of 50. In this option, the batch will modify the quantifier label on tokens to move excess quantity into additional batches. Tokens that were already waiting in the first batch will not have any of their labels modified. Use the Label Aggregation method for modifying labels.

- Destroy Excess Quantity - The label value of the last token is reduced until the batch contains the exact amount specified by the Batch Quantity property. This essentially destroys the overflowed quantity. In this option, the batch will modify the quantifier label on the last token only. No other labels will be modified. Use the Label Aggregation method for modifying labels.

New Batches Created from Excess Quantities

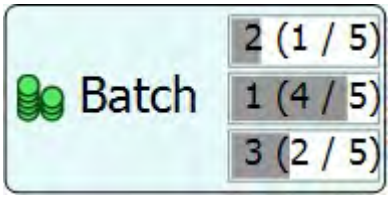
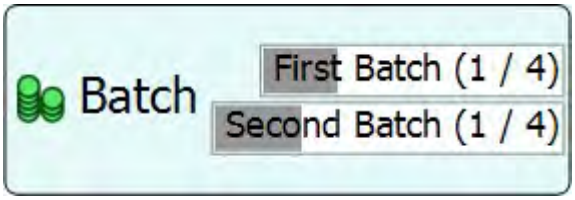
If you choose the Do Not Divide, Release Partial or Hold Excess Quantity options, you have the possibility of creating new batches with the entering token. Be aware that if the next batch created from the entering token meets the batch quantity, it will be immediately released. If the new batch exceeds the batch quantity, it will trigger the overflow logic again. This will continue until the quantity is less than the Batch Quantity.

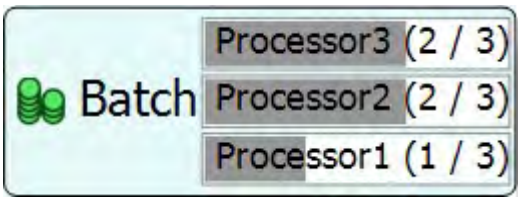
The overflow logic only affects the label value that is specified by the Quantifier property. Other labels on the tokens will not be affected. Label Aggregation can also be affected if of a token is held over to be released in a new batch. See Assigning Labels to Outgoing Batches for more information.

Grouping

When a new token arrives at the Batch activity, it will evaluate the Group By property and place the token into the matching batch. If no matching batch is found, a new batch will be created. By default, Group By is set to *None* and only one batch will be created at a time. But you can use the Group By property if you want to sort your tokens based upon criteria you specify. For example, you could sort tokens into batches based on the token's label values, names, or the object to which the token is linked. Generally this will be a label on the entering token, but it may reference a global table or other value.

The Group By can be numbers, strings, or objects, as explained in the following table:

Type	Description	Example
Number	The tokens can be grouped into batches based on a number, such as a numerical value in a label on the token. The number can be an integer or floating point number and it may be negative or positive.	
Text	The tokens can be grouped into batches based on a string (which is another word for text). For example, you could have an Assign Labels activity earlier in the process flow that assigns a new label to each token called *Batch* and it could assign the strings "First Batch" or "Second Batch" to a certain percentage of the tokens.	

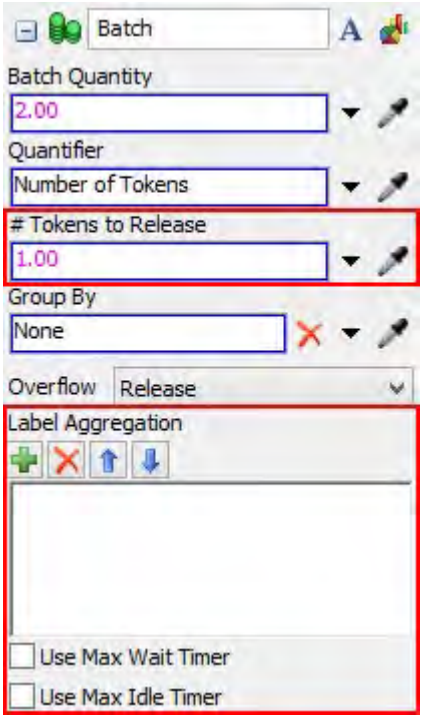
Object	The tokens can be grouped into batches based on an object. For example, the tokens might have a label called *ProcessBy* that references a processor in the 3D model that needs to process that particular batch of tokens	
	(Processor1, Processor2, or Processor3).	

Batches displayed on the Batch activity will be displayed oldest to newest from top to bottom and from left to right.

Releasing Batches

This topic will talk about the two ways that a batch may be released from the Batch activity. This can happen either when the batch is full (see Organizing Batches for information about how batches are collected and sorted), or when a batch is manually released. In both cases at least one token will leave the Batch activity.

The Batch activity has a variety of different options you can use to control how batches are released. The following properties on the Batch activity are used when a batch is released:



This topic will explain how you can use these properties to control what happens when a Batch activity is ready to release a batch. It will cover the following topics:

- Releasing More than One Token and Destroying Tokens
- Assigning Labels to Outgoing Batches
- Releasing a Batch Early

Releasing More than One Token and Destroying Tokens

The Batch activity allows any number of tokens to be released when a batch is finished. This can be useful to simulate a process that collects materials, combines them together, and then divides the combined materials equally into a certain number of portions. For example, a Batch activity could be set to collect 5 tokens and release 3 tokens, almost as though it had mixed the original tokens together and divided them up into different portions.

You can use the # Tokens to Release property to change how many tokens get released. This number can be the same number of tokens collected in the batch, greater than the number of tokens collected, or less than the number of tokens collected. It's important to understand what is happening inside the batch for each of these three scenarios.

Release Order

There are a number of references below to the order in which tokens are released from the batch. Though all of the tokens are released at the same model time (in 0 model time), the order in which they are released may impact the flow of your process flow. For instance, the first token released from the batch will enter an Acquire activity first and may acquire a resource before any other tokens from the same batch arrive.

If # Tokens to Release is Equal to the # of Collected Tokens

This is the simplest of the three scenarios. If three tokens are collected in the batch and the # Tokens to Release is set to three, then all three of the original tokens added to the batch will be released in the same order in which they arrived. This can be useful as a method of syncing a set of tokens.

If the Quantifier is set to a label value, the number of tokens in each batch may vary. In order to release all of the tokens that were collected in the batch, you can select the option *Number of Tokens Batched* or return any number <= 0 in the # Tokens to Release box.

If # Tokens to Release is Greater than the # of Collected Tokens

If the number specified in the # Tokens to Release box is greater than the number of tokens collected in the batch, the Batch activity will create and release new tokens. These new tokens will have no initial labels. Labels may be created on these new tokens by using the Label Aggregation. See Assigning Labels to Outgoing Batches for more information. These new tokens will be the last to leave the Batch activity.

If # Tokens to Release is Less than the # of Collected Tokens

If the # Tokens to Release is less than the number of tokens collected in the batch, tokens will be destroyed until the correct number of tokens remains. In deciding which tokens to destroy and which to preserve, the Batch activity prioritizes tokens using the following rules, in order:

1. Tokens that have children in the same batch should be preserved over their children. (See Sub Process Flows for more information about tokens with parent-child relationships.)
2. Tokens with more children should be preserved over tokens with fewer children.
3. Tokens with more allocated shared assets should be preserved over tokens with fewer allocated shared assets.

Once the Batch activity has prioritized the tokens, it will destroy tokens with the lowest resulting priority.

Be aware that when a Batch activity destroys tokens, it functions in a way that is similar to Finish activity:

- If the Batch activity destroys a token that has a parent that is in a Run Sub Flow activity, the parent activity will be notified that the child finished.
- Any shared assets that have been allocated by the token will be deallocated.
- If the token has children, the token will not be destroyed until all its children are destroyed (it will remain in the activity but will not be part of any subsequent batch).

Extra tokens will be destroyed before any tokens are released from the Batch activity.

Assigning Labels to Outgoing Batches

By default, tokens that enter and leave the Batch activity will retain their labels and no new labels will be added to them. Newly created tokens will have no labels assigned to them. However, utilizing the Label Aggregation, the Batch activity can assign new labels or aggregate the labels of the tokens that were included in the batch to all of the outgoing tokens. Label aggregation does not occur until the batch has collected all of its tokens as has reached the Batch Quantity value.

One possible use of the label aggregation would be to calculate the total weight of all of the parts that are combined into a single part. If each token represented a single part and they each had a label named *Weight*, the label aggregation can be used to sum all of the weights of every token added to the batch and create a new label like *BatchWeight* that will be assigned to all of the outgoing tokens.

To assign labels to outgoing batches:

1. Click the Batch activity to select it.

2. In Quick Properties, under the Label Aggregation group, click the Add button to add a new label. A new row will appear in the table.

3. In the From Label column, double-click on the cell with the text that says *labelName*. Type the name of the label on the tokens in the batch that will be used as the source of the data. For example, if the incoming tokens all have a label named *Weight*, you would type that label name here.

4. In the To Label column, double-click on the cell with the text that says *labelName*. Type the new name of the label that will be assigned to outgoing tokens. For example, the new name could be *BatchWeight*.

5. In the Aggregation column, you'll select how the value of the new label will be calculated or assigned. Click on the cell with the text that says *Batch ID* to open a menu. Select the appropriate option from the menu. For example, if you wanted to calculate the total sum of all the batched token's *Weight* labels, you could select SUM from this menu.

For your reference:

From Label - The name of a label on the batched tokens that will be used as a source of data. The value of this label from each token will be used to calculate the aggregated value. If the label does not exist on a token, the returned value will be 0. NOTE: This entry is not used for the *Batch ID* and *Token Count* aggregation types.

To Label - The name of the label that the new value will be written to. Each token in the outgoing batch will get a label with this name, and with the assigned value.

Aggregation - The value that will be assigned to the To Label on all tokens released for the batch. The possible values are as follows:

- Batch ID - Starting at 1, each batch that is released from the Batch activity will increment the Batch ID by 1. The Batch ID is only guaranteed to be unique among batches produced by a given Batch activity.
- Token Count - The number of tokens collected in the batch. If the Quantifier property is set to *Number of Tokens*, this value will be equal to the Batch Quantity value.
- Last Value - This will get the label value of the From Label on the last token to enter the batch.
- First Value - This will get the label value of the From Label on the first token to enter the batch.

- Make Array - This will get the label value of every token collected in the batch and create an array of those values. If the label does not exist on a token, a NULL value will be set on the array.
- SUM - Sums the From Label values on all tokens in the batch.
- AVG - Gets the average of the From Label values on all tokens in the batch.
- MIN - Gets the minimum From Label value on all tokens in the batch.
- MAX - Gets the maximum From Label value on all tokens in the batch.
- STD - Gets the standard deviation of the From Label values on all tokens in the batch.
- VAR - Gets the variance of the From Label values on all tokens in the batch.
- COUNT - Gets the number of tokens in the batch that have a label named From Label.

Batch Modifying Labels

In general, token labels are not modified by the Batch. There are some exceptions that can occur based upon the overflow type. See Batch Overflow Options for specific information.

Aggregating the Quantifier Label when using Overflow

If you are batching by a label on the tokens and your Overflow is set to *Hold Excess Quantity* or *Destroy Excess Quantity* then the quantifier label may be modified prior to performing the aggregation. For example, if you collect by the label *Weight* and if the Batch Quantity is set to 300 and the final batched quantity is 350, the SUM of the *Weight* label will be equal to 300, not 350.

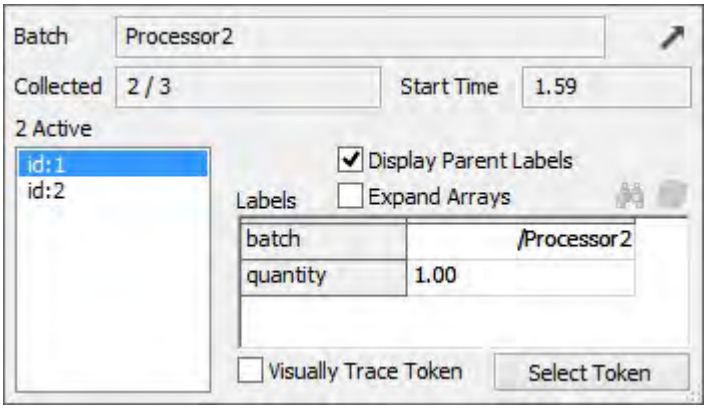
Releasing a Batch Early

The Batch activity has both a Max Wait Time and Max Idle Time property. These properties allow you to define start criteria and the amount of time to wait before firing a trigger. If the batch reaches one of these times and calls either the OnWaitTimerFired or OnIdleTimerFired trigger, you have the option to release the batch early. However, this is not a requirement. If the batch is not released, the batch will continue to collect tokens until it reaches its Batch Quantity, after which the batch will be released as normal. The Max Wait Time and Max Idle Time properties give you a reference to the batch. The releasebatch() command can be called to release a batch prematurely from within the OnWaitTimerFired or OnIdleTimerFired triggers or from anywhere else in the model. If failed is 1, the batch will immediately release all tokens in the batch and not perform any label aggregation or evaluate the # of Tokens to Release property. If failed is 0, the batch will release normally and evaluate the # of Tokens to Release and perform all of the label aggregations on the tokens in the batch. The releasebatch() command also gives you the ability to send the released tokens out any connector of the Batch activity. The connector can be referenced by index or name. The releasebatch() command even allows you to release tokens to an activity that is not directly connected to the Batch activity by referencing the activity using getactivity(). See Max Wait Time and Max Idle Time for more information about these properties.

Batch Statistics

You can use statistics to get useful data from your Batch activities if needed. This topic will discuss how the statistics behave differently on Batch activities (as opposed to the statistics on other process flow objects).

Batch Dialog Box



The Batch dialog box can be opened by clicking on the batch displayed on the Batch activity. This dialog box gives information on the selected batch and lets you see which tokens are currently in the batch.

- BatchThis is the Group By value of the batch. If the Group By was set to None, this field will display 0.
- CollectedDisplays the progress of the batch either as the # of tokens collected or the summed label value collected out of the total Batch Quantity.
- Start TimeThe time the batch was created in model time units.
- Active TokensDisplays all the tokens currently in the batch.
- Token InfoDisplays the labels of the selected token and allows you to select and visually trace the token.

Standard Statistics

The standard statistics kept by the Batch activity (see Statistics) are unique because the Batch can create or destroy tokens. This makes the statistics a little less intuitive. The following list discusses the meaning of each statistic:

- Input All tokens that arrive at a Batch activity increment the input. Tokens created by the Batch do not affect input.
- Output All tokens that move on to another activity from the Batch increment the output. Tokens destroyed by the Batch do not affect output.
- Content All tokens that arrive at the Batch activity increment the content. All tokens that leave the Batch activity decrement the content, including tokens that are destroyed by the Batch. Tokens created by the Batch do not affect content, unless they were created as part of overflow logic. The content is always equal to the number of tokens currently being collected.
- Staytime All tokens that arrive at the Batch activity, or are created as part of overflow logic, will affect the staytime when they leave or when they are destroyed. The average staytime is the average time that tokens spend collecting on the Batch activity.

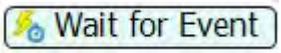
Batch Statistics

The Batch activity can also record statistics dealing with batches, rather than tokens. The following list describes these statistics; type of each statistic (Incremental, Level, or Stream) is listed with the title.

- Number of Batches (Level)The number of batches presently being collected on the batch.
- Batches Created (Incremental)The number of batches created by the Batch activity.
- Batches Released (Incremental)The number of batches released by the Batch activity.
- Batch Staytime (Stream)The amount of time each batch spent being collected. The Batch activity updates this statistic every time a batch is released (including manually releasing a batch early).
- Batch Quantity (Stream)The quantity within a batch. The Batch activity updates this statistic every time the quantity of any batch changes, setting it to the quantity of the batch. This statistic is probably most useful as an event; for example, you could listen for any time a batch quantity exceeds a certain value.
- Tokens per Batch (Stream)The number of tokens in each batch. The Batch activity updates this statistic every time a batch is released, setting it to the number of tokens in the most recently released batch.
- Quantity per Batch (Stream)The quantity in each batch. If the Batch is collecting by Number of Tokens, this statistic will not be updated, since it would be identical to the Tokens per Batch statistic.

Wait for Event

The Wait for Event activity will hold the token until a certain event is triggered. This activity will listen for that event to occur in the simulation model. When that event occurs, it will release the token. (See EventListening Activities for more information about the types of events that can be listened for.)



To get a deeper understanding of how the Wait for Event activity works, you might want to consider reading these topics first:

- Linking Process Flows to Simulation Models
- Key Concepts About Event-Listening Activities
- Event Types and Related Properties

As a more advanced feature, the Wait For Event also has the ability to override the return value of the event it is listening to. For more information, see the Will Override Return Value.

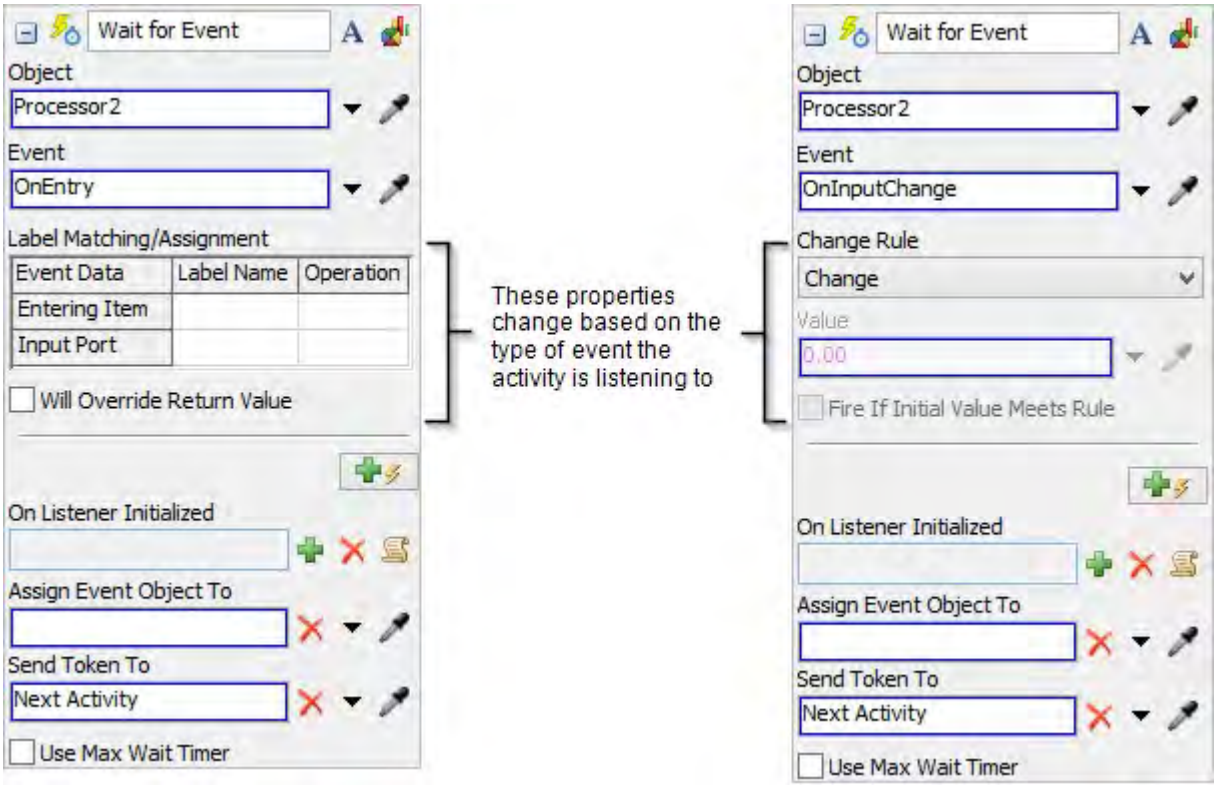
When this activity is first created, a red exclamation mark shows up to the right of the activity notifying you that a link to an event is required for this activity to function. This link may be a direct pointer which can be created by clicking on the exclamation mark and then clicking on an activity or object and selecting an event, or the reference may be set through the quick properties.

Connectors

The Wait for Event activity allows multiple outgoing connectors. However, tokens automatically released from this activity will be released through the first connector. Only manually released tokens can exit out a different connector. See Adding and Connecting Activities for more information.

Properties

The following image shows properties for the Wait for Event activity:



Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button **A** opens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- ObjectUse the Object box to select the object that the Wait for Event activity will listen to. You could possibly listen to:
    - An object in the model.
    - An activity in the Process Flow.
    - The current instance object (See Instances for more information.)
    - A group of objects. In this case, the activity will listen to the defined event on all members of the group.

  You could select the event you want to listen to by either using the pull-down arrow next to the Object box or using the Sampler button to select the object.

  See Event-Listening Activities for more detailed information about listening to objects and events.
- EventUse the Event box to select the event that the Wait for Event activity will listen to. The Event box is connected to the object you selected in the Object box. When you use the Sampler button to select an object for the Object box, a menu will appear that lists all the different types of events that are available for that object. (This menu is created dynamically based on all the possible events related to that object.) The event that you select from this menu will automatically be listed in the Event box.

You can listen to more than one event on an object. To add an event in addition to the original event, press the Add Event button  . Additional events will have a Delete button  button if you need to remove them.

See Event-Listening Activities for more detailed information about listening to objects and events.

- Label Matching/Assignment and Change RuleThe Label Matching/Assignment or Change Rule properties will appear after you've selected an object and event to listen to. See Event Types and Related Properties for more information about these sets of properties.
- OnListenerInitialized This is a field that is evaluated after the event listening mechanism has been set up. You may want to use this field if, for example, in a Fixed Resource Process Flow you want to both release an item and wait for the item to leave the object. Here you would want to first set up the exit listening mechanism, and then release the item (since if you do it in the opposite order, the item may leave before you can set up the exit listening). You can use the Wait for Event to listen for the exit, and then once the listening is set up, release the item.
- Assign Event Object To Assigns a reference to the event-firing object to the specified label or node. This is useful if you are listening to multiple objects (perhaps from a Group) and need to know which object fired the event. This box is not required and can be left empty.
- Decision By default, the token is released to the next activity when the triggering event occurs in the simulation model. However, you can use the Decision box to build some more complex logic into the Wait for Event activity so that it acts more like a Decide activity. See the Decide activity for more information about the Decision property.
- Use Max Wait Timer If you check the Use Max Wait Timer to release the token after a certain amount of time has passed, regardless of whether the event it was waiting for occurred. You can also determine what should happen to this token if it is released because the wait timer expired. See Max Wait Timer for more information about this property.

Create Tokens

When the Create Tokens activity receives an incoming token, it will create one or more new tokens and automatically send them to a different activity or sub flow (referred to here as the *destination activity*). This activity is similar to the Run Sub Flow activity, except this activity does not hold on to the entering token once the created tokens are released. This activity is also not restricted to only sending tokens to a sub flow or Start activity. Tokens can be created in any activity to include activities in other General Process Flows.

Consider reading Sub Process Flows for more in-depth information about:

- How Sub Flows Work
- Creating Internal Sub Flows
- Creating External Sub Flows
- Linking to Sub Flows
- Multiple Start or Finish Activities

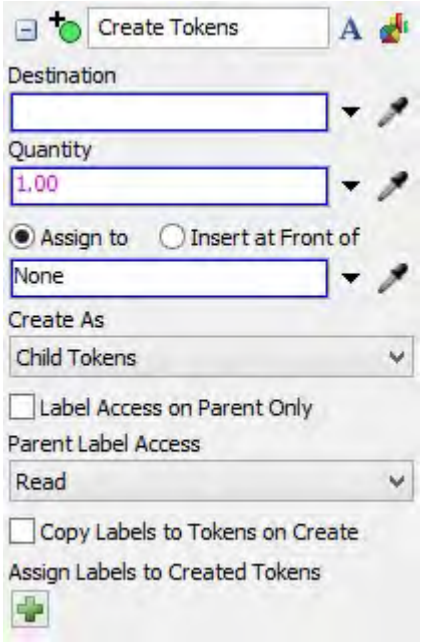
When this activity is first created, a red exclamation mark shows up to the right of the activity notifying you that a link to another activity or Process Flow Sub Flow object is required for this activity to function. This link may be a direct pointer which can be created by clicking on the exclamation mark and then clicking on an activity or Sub Flow object, or the reference may be dynamic by setting the value for the Destination property.

Connectors

The Create Tokens activity can have any number of connectors out. Created tokens will exit this activity based upon the Destination. The entering token will always exit through the first connector out (or the next activity if part of a stacked block). See Adding and Connecting Activities for more information.

Properties

The following image shows the available properties for the Create Tokens activity:



Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button Aopens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- Destination Indicates the destination activity that the newly created tokens will be automatically sent to. This can be the index of a connector, a connector name, or an activity. Unlike the Run Sub Flow activity, the destination is not restricted to being a Start activity or a Sub Flow. This activity can send the newly created tokens to any activity, whether it be in a sub flow, a General process flow, or any activity in the same process flow. See Linking to Sub Flows for information about linking to a sub flow from this property.
- Quantity The number of tokens that will be created every time the Create Tokens activity receives an entering token.
- Assign To Assigns a reference on the specified label/node to the created tokens. See the Assign To section of Common Properties for more information.
- Create AsDefines the relationship of the original token to the created token(s). o Independent Tokens - Created tokens will have no association with the original token. o Child Tokens - Created tokens will be child tokens of the original token.
    - o Sibling Tokens - Created tokens will be sibling tokens of the original token. In other words, they will have the same parent as the original token.
- Label AccessFor information on Label Access, see the Label Access section of the Sub Process Flows page.

Sink

The Sink activity destroys tokens, removing all data stored on those tokens. If a token has children, it will remain inside the Sink activity until all of its children have been destroyed, thus maintaining the links and data on the children. If Deallocated Shared Assets is unchecked and the token has allocated shared

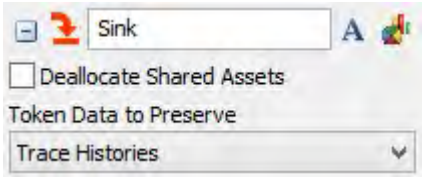
assets, the token will also remain inside the Sink activity until all shared assets have been deallocated. The Sink activity and the Finish activity are valid ways to destroy tokens. Do not use the destroyobject() command to destroy tokens.

Connectors

The Sink activity marks the end of a set of activities. No activity may be connected to a Sink. See Adding and Connecting Activities for more information.

Properties

The following image shows properties for the Sink activity:



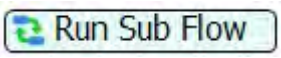
Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button **A** opens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- Deallocate Shared Assets If the Deallocate Shared Assets box is checked, any Shared Assets that are allocated by the token will be deallocated when the Sink destroys the token. In other words, when a token is destroyed by the Sink, any resources that have been previously acquired by the token will be released and any entries or backorders on lists associated with the token will be deleted. If the token is in a zone, the zone will only update its content and not update any of its other statistics. If unchecked, the token will remain in the Sink activity until its shared assets have been deallocated.

  Exiting a Zone
  In order to properly remove a token from a zone and have the zone's output, staytime and custom statistics updated, send the token through an Exit Zone activity.

Run Sub Flow

The Run Sub Flow activity can be used to initiate a sub process flow. The entering token will remain in the Run Sub Flow activity until all of its child tokens have completed their sub flow (by exiting a Finish activity).



Consider reading Sub Process Flows for more in-depth information about:

- How Sub Flows Work
- Creating Internal Sub Flows
- Creating External Sub Flows
- Linking to Sub Flows
- Multiple Start or Finish Activities

When this activity is first created, a red exclamation mark shows up to the right of the activity. This is notifying you that a link to the Run Sub Flow activity is required. This can be either to a Process Flow Sub Flow object that has a Start activity in it or to a Start activity in the same process flow. This link may be a direct pointer which can be created by clicking on the exclamation mark and then clicking on a Sub Flow object or Start activity, or the reference may be dynamic by setting the value for the Destination property.

Starting and Finishing a Sub Flow

The Run Sub Flow activity requires a Start activity as the first activity in the sub flow and a Finish activity as the final activity. As child tokens enter the Finish activity, they notify the parent token (in the Run Sub Flow) that the child is complete. Once all children have completed the sub flow, the parent token will be released from the Run Sub Flow.

If the Run Sub Flow references a Process Flow Sub Flow object, it will search the Sub Flow for the first Start activity and use that as the start of the sub flow. If your Sub Flow has multiple Start activities, you can use the Sampler to a specific Start activity or enter `SubFlowName: StartName` in the Destination box.

Connectors

The Run Sub Flow activity only allows one connector out. See Adding and Connecting Activities for more information.

Properties

The following image shows properties for the Run Sub Flow activity:

Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button A opens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- Destination Use the Destination box to link to a sub flow. This can be a Process Flow Sub Flow object, or it can be a Start activity. See Linking to Sub Flows for more information.
- Quantity The number of child tokens that will be created in the sub flow.
- Label Access For information on the Label Access property, see the Label Access section.

Start

The Start activity is usually the first activity in a sub flow. However, the Start activity only marks the beginning of a sub flow. It does not perform any other logic. After a token enters the Start activity, it will immediately move to the next connected activity in the sub flow.

Consider reading Sub Process Flows for more in-depth information about:

- How Sub Flows Work
- Creating Internal Sub Flows
- Creating External Sub Flows
- Linking to Sub Flows
- Multiple Start or Finish Activities

Connectors

The Start activity only allows one connector out. See Adding and Connecting Activities for more information.

Properties

The following image shows properties for the Start activity:

Each of these properties will be explained in the following section.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button opens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.

Finish

The Finish activity marks the end of a sub flow. The Finish activity essentially acts the same as a Sink in that it destroys tokens that enter it.

The Finish activity has a few additional functions depending on what kind of activity or event that triggered the sub flow to run:

- Run Sub Flow - If the sub flow is being run through a Run Sub Flow activity, each token entering the Finish will notify its parent token that is waiting in the Run Sub Flow that it has finished its sub flow. Once all tokens have finished for that parent, the parent will be released from the Run Sub Flow activity. In this case, the Return Value property is not used.

- Will Override Return Value - The Wait For Event and Event Triggered Source activities have a Will Override Return Value property. This property allows the token to give a new return value to the event that was triggered. For example, overriding the *Process Time* event of a processor would allow you to perform steps using process flow to define the process time of that processor. The processor will ignore whatever value was originally returned in the *Process Time* box. When overriding a return value, the token will move through the defined set of activities until it hits a Finish activity where it will evaluate the return value for the event. (A return value might be data from a calculation, a new label value, etc.)

- Execute Sub Flow command - If you are using any kind of code that uses the `executesubflow()` command to run a sub flow (some picklist options use this command), the Finish activity can return a value when the sub flow is finished being run such that `double processTime = executesubflow(processFlow);` will give you the value evaluated from the Return Value box.

About Sub Flows

Consider reading Sub Process Flows for more in-depth information about:

- How Sub Flows Work
- Creating Internal Sub Flows
- Creating External Sub Flows
- Linking to Sub Flows
- Multiple Start or Finish Activities

Overriding a Return Value

Though Finish activities are generally thought of as the end of a sub flow, they are also used to mark the end of a Will Override Return Value option from the Wait For Event or Event Triggered Source activity or the end of an executesubflow() command. In both of these cases, one token will be used to execute a series of activities ending with a Finish activity. Upon arriving at the Finish activity, the Return Value box will be evaluated for the token and the result will be returned to the calling function. There are picklists available through standard FlexSim objects that execute sub flows in order to get return values for properties such as a Processor's Processing Time.

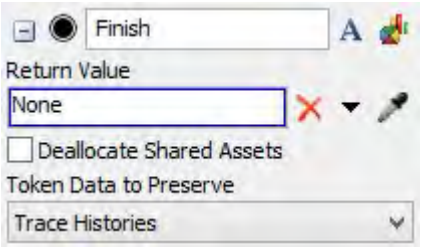
If the Finish marks the end of a sub flow, the Return Value will not be evaluated.

Connectors

The Finish activity marks the end of a set of activities. No activity may be connected to a Finish. See Adding and Connecting Activities for more information.

Properties

The following image shows properties for the Finish activity:



Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button Aopens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.

- Return Value The Return Value property can be a static or dynamically-generated value. A return value can be anything you need it to be. For example, you could return a calculation based on data in a label on the token or you could return a reference to an object. See Overriding a Return Value for more information.
- Deallocate Shared Assets If checked, when a token is destroyed, any Shared Assets that are allocated by the token will be deallocated. This means, any resources that have been acquired by the token will be released and any entries or backorders on lists associated with the token will be deleted. If the token is in a zone, the zone will only update its content and not update any of its other statistics. If unchecked, the token will remain in the Finish activity until its shared assets have been deallocated.

    Exiting a Zone
    In order to properly remove a token from a zone and have the zone's output, staytime and custom statistics updated, send the token through an Exit Zone activity.

Change Visual

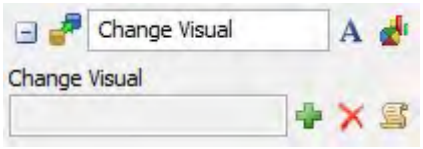
The Change Visual activity changes the appearance of an object in the 3D model. Generally this is used for changing a flowitem's visuals, but it may also be used to adjust any 3D object's visual properties.

Connectors

The Change Visual activity only allows one connector out. See Adding and Connecting Activities for more information.

Properties

The following image shows properties for the Change Visual activity:



Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button Aopens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- Change VisualBy pressing the button, you can add a number of picklist options for changing an object's visual properties. You may also edit the code directly by clicking on the button. Pressing the will remove all code from the trigger, making it blank.

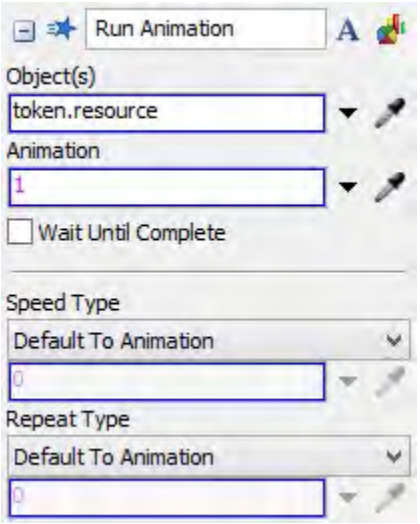
The Run Animation activity can trigger an animation on a 3D object in the simulation model. You can use any of the animations that come pre-programmed with the standard FlexSim objects or you can create your own custom animations. If desired, you can have the token wait in the Run Animation activity until the animation completes. This can be useful for syncing. See the Animation Creator for more information about creating custom animations.

Connectors

The Run Animation activity only allows one connector out. See Adding and Connecting Activities for more information.

Properties

The following image shows properties for the Run Animation activity:



Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button **A** opens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- Object(s) Use the Object(s) property to select the object that will be animated. To run animations on multiple objects at the same time, you can pass an Array of objects into this property. Alternatively, you can create a

group of the objects you want to animate. (See Groups for more information about creating groups of objects.) To reference a group in the Object(s) property, click the pull-down menu next to this property and select Groups to see a list of all the groups that have been created for this simulation model (or type Group: groupName directly into the box).

- Animation Use the Animation property to determine which animation on the object should be run. This should be either a number that is the index of the animation or a string that is the name of the animation on the object.

  To view the animations that are available for any 3D object:

  1. Right-click on the object in the 3D model to open a menu.
  2. Select Edit then Animations to open the Animation Creator.
  3. On the bottom panel, look for the Animation menu. Click the arrow on this menu to open the full list of the currently available animations.

  You can either type the name of the animation or its index number. The index number is the order in which the animation appears in the Animation menu. For example, an Operator object has three animations (listed in this order): Walk, WalkLoaded, and Stand. The Walk animation would be index 1, WalkLoaded would be index 2, and so forth. If entering the name of the animation directly, be sure to put quotes around the name so the Universal Edit will recognize it as a string. For example, "WalkLoaded".

- Wait Until Complete If checked the token will remain in the Run Animation activity until the animation is finished.

  Repeat Indefinitely If you cause the token to wait until the animation is complete, the animation's Repeat Type must be set to *Do Not Repeat* or *Repeat Set Number*. Otherwise the animation will loop continuously and the token will never exit the Run Animation activity.

  To check or change an animation's Repeat Type setting:

  1. Right-click on the object in the 3D model to open a menu.
  2. Select Edit then Animations to open the Animation Creator.
  3. Select the desired animation from the Animations and Components pane.
  4. In Quick Properties, look for the Repeat Type menu.
  5. Click the menu and select Do Not Repeat or Repeat Set Number. If you selected the latter option, make sure you indicate how many times the animation should be repeated in the Repeat Value box.

- Speed Type If set to Default To Animation, the speed at which the animation will run is be defined by the animation. You can override the animation's speed by either specifying a duration or a multiplier.
  - Use Duration Of - Enter the duration of the animation in the box below in model time units.
  - Multiply Speed By - Enter a multiple in the box below to increase or decrease the animation's speed by the specified value.
- Repeat Type If set to Default To Animation, the repeat type of the animation will be defined by the animation. You can override the animation's repeat type with the following options: o No Repeat - The animation will not repeat.
  - Repeat Immediately - The animation will repeat as soon as it finishes. o Repeat # Of Times - Enter the number of times the animation should repeat in the box below.

- Repeat After Time - Specify the time duration the animation should run before repeating. Enter the time in model units in the box below. You may specify a time that is less than the total animation time.
- Repeat After Time Once Ended - Once the animation is finished, specify how much time between animation repeats. Enter the time in model units in the box below.

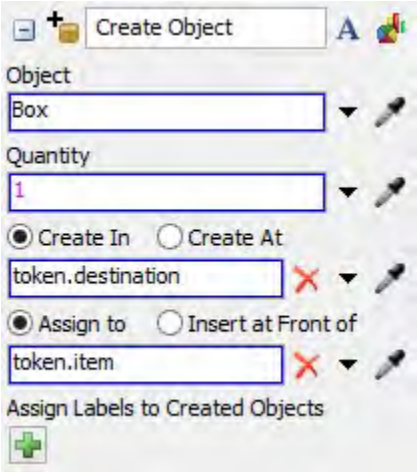
The Create Object activity creates one or more objects in the 3D simulation model. This activity can also create TaskExecuters and TaskExecuter Flowitems and connect them to a travel network using the Destination property. Once created, a reference to the created object(s) will be added to the label specified in the Assign To property.

Connectors

The Create Object activity only allows one connector out. See Adding and Connecting Activities for more information.

Properties

The following image shows properties for the Create Object activity:



Each of these properties will be explained in the following sections.

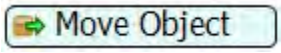
- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button **A** opens a window to edit the activity's background color and font properties. See Font for more information about this property.

- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- Object Use the Object box to specify what type of object should be created. By default, it will create a flowitem with a Box shape, but there are many other possibilities. If you click the arrow next to the Object box, the menu will list a variety of different objects that can be created. If needed, you can dynamically set this property as the rank of a flowitem in the Flowitem Bin.

  You can also use the Sampler button to select an object in the model. When you select an object in the model, the Create Object activity will create a copy of this object.
- Quantity Use the Quantity box to determine how many objects will be created. This should be an integer value (a whole number).
- Destination (Create In or Create At) Use the Destination box to determine where the object(s) will be located when created. By default, the destination is the object referenced by the entering token's *destination* label, but it can be changed to be a different token label, object in the model or other destination. Some of the possible destinations can be:
  - Object in the Model - The object will be created in an object in the model. For example, you could create a flowitem inside of a processor. Use the Sampler button to select an object in the 3D simulation model. Selecting Create At will create the object at the location of the destination object, but inside the destination object's parent.
  - Model - The object will be created inside the model with its X,Y,Z coordinates at 0,0,0. If you are creating a task executer object, it will be connected to the Default Navigator. To use this option, click the arrow next to the Destination box and select *Model* from the menu.
  - Outside the Model - The object can be created outside of the model if you would prefer to move the object into the model manually later. Click the Remove button so that the Destination box will read *None*.
  - Network Node - The object will be created on a network node. (See Network Node for more information.) If you are creating a task executer object, it will be connected to the Default Network Navigator. Use the Sampler button to select the network node in the 3D simulation model. o AGV Control Point - If you have the AGV module installed in FlexSim, you can set the destination to an AGV Control Point. If you are creating a task executer object, it will be created at the control point and will be connected to the AGV Network. See the AGV Control Point for more information.
  - A Star Navigator - If you have the AStar module installed in FlexSim, you can set the destination to an AStarNavigator. If you are creating a task executer object, it will be be created at the AStarNavigator and be connected to it. See the AStarNavigator for more information.

  Additional Activity Needed for the Model and A Star Navigator Options
  If you use either the Model or the AStarNavigator destination options, you might want to change the location to something other than the X,Y,Z coordinates 0,0,0 or the location of the AStarNavigator object. You can use the Change Visual activity next in your process flow to fix this issue. See the Change Visual activity for more information.
- Assign To Assigns a reference on the specified label/node to the created task sequence. See the Assign To section of Common Properties for more information.
- Assign Labels to Created Objects Allows you to add labels to the created objects. For more information on assigning labels, see the Assign Labels activity.

The Move Object activity moves an object or multiple objects to another place in a simulation model. When the object moves it will be instantaneous, meaning that it will instantly appear in its new location. It functions the same as calling the moveobject() command.



Max Content

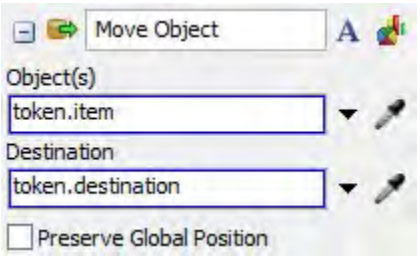
Moving an object using this activity does not take into account Max Content on a task executer or fixed resource object. It will force the object(s) into the destination object regardless of that object's availability.

Connectors

The Move Object activity only allows one connector out. See Adding and Connecting Activities for more information.

Properties

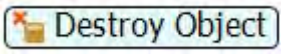
The following image shows properties for the Move Object activity:



Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button Aopens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- Object(s) Use the Object(s) box to specify the object you want to move. To move multiple objects at once, pass an array of object references into this property.
- DestinationUse the Destination box to determine where the object or objects should move to. See Create Object - Destination for a list of possibilities.
- Preserve Global LocationWhen selected, the object's global position will not be affected when moving into the destination. If this option is not selected the object's location will based on the destination object's coordinate space, which will most likely be different than the model's.

The Destroy Object activity removes one or more objects in the 3D simulation model.

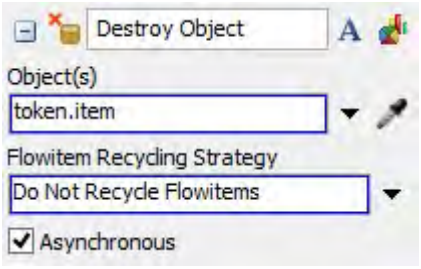


Connectors

The Destroy Object activity only allows one connector out. See Adding and Connecting Activities for more information.

Properties

The following image shows properties for the Destroy Object activity:



Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button Aopens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- Object(s) Use the Object(s) box to specify the object you want to destroy. To destroy multiple objects at once, pass an array of object references into this property.
- Recycling Strategy You can use the Recycling Strategy box to determine the recycling strategy for flowitems that are going to be destroyed. Be aware that flowitems are the only kind of object that can be recycled. If a non-flowitem is destroyed, this property will be ignored. When an flowitem is recycled, FlexSim will place the flowitem in a bucket from which creating flowitems can pull from. This can improve speeds when working with large models that have a large number of flowitems being created and destroyed. NOTE: Recycled flowitems maintain their previous visuals and data from when they were destroyed.
- As ynchronous If checked, the activity will create an event to destroy the object in 0 time. This is important if the activity was triggered within the context of a model event associated with the object you are going to destroy. For example, if you listen for an OnEntry or OnExit of a flowitem in the model and attempt to destroy the flowitem when the event fires, you will need to do the destroy asynchronously as the event may continue to use the item after your activity's logic has fired. The Asynchronous option will allow the triggering model

event to finish what it was doing before the activity destroys the object. This is the same as putting a Delay of 0 seconds before the destroy object activity.

Use the Travel activity to make a task executer (such as an Operator or Transporter) travel to a specific object (such as a fixed resource) in the 3D simulation model. If connected to a Travel Network, the task executer will use that network (such as AStar or Network Nodes) to travel to the specified object.

The Travel activity creates a travel task sequence and dispatches it to a task executer. Optionally, the task may be appended onto an already existing task sequence. See Create Task Sequence activity for more information. Also see Task Sequences for more general information about task sequences.

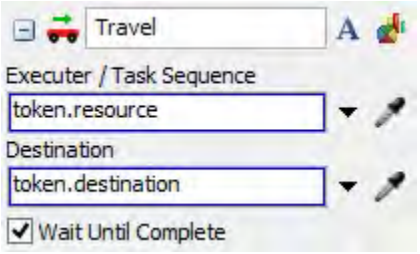
Travel vs. Travel to Loc
Use the Travel task sequence activity if you want a task executer to go to a specific object in the 3D simulation model. Use the Travel to Loc task sequence activity if you want a task executer to go to specific X, Y, and Z coordinates in the 3D simulation model (such as a place where there is no object).

Connectors

The Travel activity only allows one connector out. See Adding and Connecting Activities for more information.

Properties

The following image shows properties for the Travel activity:



Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button **A** opens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- Executer / Task Sequence Use the Executer / Task Sequence box to determine which task executer or task sequence should receive the task. See Executer / Task Sequence for more information about this property.

- DestinationUse the Destination box to set the location to which the task executer will travel. The destination should be an object in the 3D model.
- Wait Until CompleteIf the Wait Until Complete box is checked, the token will be held in the activity until this task has been completed.

Use the Load activity to make a task executer (such as an Operator or Transporter) load an object in the 3D simulation model. For example, you can make a task executer pick up a flowitem to transport it to another destination.

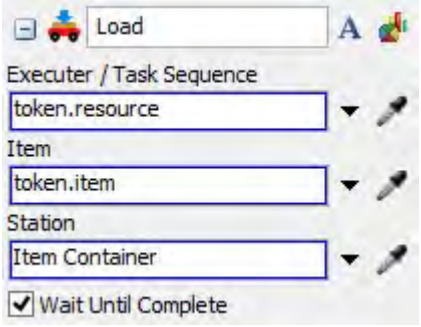
The Load activity creates a load task sequence and dispatches it to a task executer. Optionally, the task may be appended onto an already existing task sequence. See Create Task Sequence activity for more information. Also see Task Sequences for more general information about task sequences.

Connectors

The Load activity only allows one connector out. See Adding and Connecting Activities for more information.

Properties

The following image shows properties for the Load activity:



Each of these properties will be explained in the following sections.

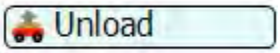
- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button Aopens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.

- Executer / Task Sequence Use the Executer / Task Sequence box to determine which task executer or task sequence should receive the task. See Executer / Task Sequence for more information about this property.
- Item Use the Item box to determine which item or object should be loaded. Typically this is a flowitem, however, other objects can be used. There are many different ways to reference flowitems and objects in the model. Below are just a couple of examples:
    - You can use a Create Object activity to create a flowitem in the 3D model and assign the appropriate label to that object.
    - You can create a Wait For Event or Event Triggered Source activity that listens for flowitems arriving at a fixed resource and assigns a label to those entering flowitems.

- StationBy default the Station box is set to *Item Container* which is the object that the item is currently inside of. For information about why you might choose a station that is not the item container, see Task Type Load.
- Wait Until CompleteIf the Wait Until Complete box is checked, the token will be held in the activity until this task has been completed.

Use the Unload activity to make a task executer (such as an Operator or Transporter) unload an object in the 3D simulation model. For example, you can make a task executer drop off a flowitem at a processor or queue.

The Unload activity creates an unload task sequence and dispatches it to a task executer. Optionally, the task may be appended onto an already existing task sequence. See Create Task Sequence activity for more information. Also see Task Sequences for more general information about task sequences.

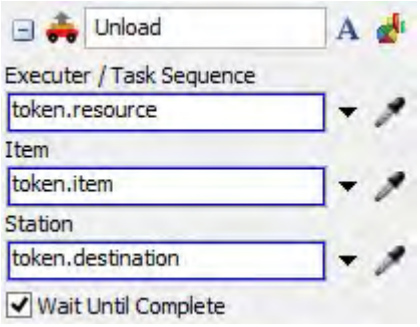


Connectors

The Unload activity only allows one connector out. See Adding and Connecting Activities for more information.

Properties

The following image shows properties for the Unload activity:



Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button Aopens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- Executer / Task Sequence Use the Executer / Task Sequence box to determine which task executer or task sequence should receive the task. See Executer / Task Sequence for more information about this property.
- Item Use the Item box to determine which item or object should be unloaded. Typically this is a flowitem, however, other objects can be used. There are many different ways to reference flowitems and objects in the model. Below are just a couple of examples:

-
    - You can use a Create Object activity to create a flowitem in the 3D model and assign the appropriate label to that object.
    - You can create a Wait For Event or Event Triggered Source activity that listens for flowitems arriving at a fixed resource and assigns a label to those entering flowitems.

  StationUse the Station box to select the object from which the task executer should unload the item to. Usually you will point to a fixed resource (such as a processor or queue) but you can also point it to another task executer.
- Wait Until CompleteIf the Wait Until Complete box is checked, the token will be held in the activity until this task has been completed.

Use the Delay (task sequence) activity to make a task executer (such as an Operator or Transporter) delay for a specific period of time in the 3D simulation model. For example, a delay could represent a task that takes a specific amount of time to complete such as cleaning a machine, assembling a product, etc.

The Delay activity creates a delay task sequence and dispatches it to a task executer. Optionally, the task may be appended onto an already existing task sequence. See Create Task Sequence activity for more information. Also see Task Sequences for more general information about task sequences.

Make Sure You're Using the Correct Delay Activity

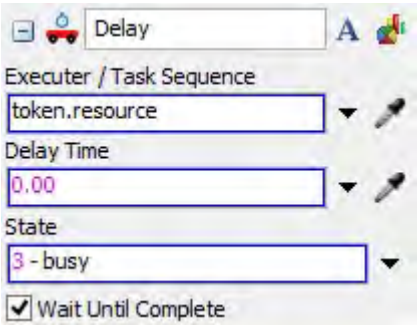
Don't confuse the Delay (task sequence) activity with the the general Delay activity .

Connectors

The Delay activity only allows one connector out. See Adding and Connecting Activities for more information.

Properties

The following image shows properties for the Delay activity:



Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button opens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- Executer / Task Sequence Use the Executer / Task Sequence box to determine which task executer or task sequence should receive the task. See Executer / Task Sequence for more information about this property.
- Delay TimeThe Delay Time defines the amount of time that the task executer will be delayed.

-
- State Use the State box to determine which state the task executer will be in during the delay task (such as idle, busy, on break, etc.). The task executer's state is useful if you want to record statistics about how long the task executer is in a particular state. Click the arrow next to the State box to see a menu of available states. You can also specify user defined states.

  Wait Until CompleteIf the Wait Until Complete box is checked, the token will be held in the activity until this task has been completed.

Use the Travel to Location activity to make a task executer (such as an Operator or Transporter) travel to specific X, Y, and Z coordinates in the 3D simulation model. This will cause the task executer to perform offset travel (ignoring its navigator) to the location.

The Travel to Location activity creates a travel task sequence and dispatches it to a task executer. Optionally, the task may be appended onto an already existing task sequence. See Create Task Sequence activity for more information. Also see Task Sequences for more general information about task sequences.

Travel vs. Travel to Loc

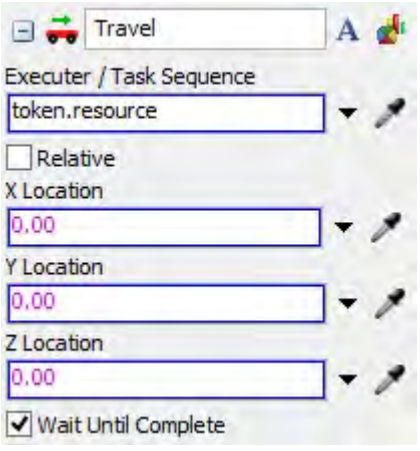
Use the Travel task sequence activity if you want a task executer to go to a specific object in the 3D simulation model. Use the Travel to Loc task sequence activity if you want a task executer to go to specific X, Y, and Z coordinates in the 3D simulation model (such as a place where there is no object).

Connectors

The Travel To Loc activity only allows one connector out. See Adding and Connecting Activities for more information.

Properties

The following image shows properties for the Travel To Loc activity:

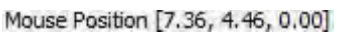


Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button **A** opens a window to edit the activity's background color and font properties. See Font for more information about this property.

-
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- Executer / Task Sequence Use the Executer / Task Sequence box to determine which task executer or task sequence should receive the task. See Executer / Task Sequence for more information about this property. RelativeWhen the Relative checkbox is checked, the X, Y, and Z coordinates will be relative to the current location of the task executer. If the Relative checkbox is cleared, the task executer will travel to the X, Y, and Z coordinates of the object containing the task executer, whether that be the model or a visual tool.
- X LocationThe X location to which the the task executer will move.
- Y LocationThe Y location to which the the task executer will move.
- Z LocationThe Z location to which the the task executer will move.

  Finding the X, Y, and Z Coordinates in the Model
  Position your mouse over the location you want the task executer to travel to. The bottom left corner of the FlexSim window will display the current mouse position, such as in the following image:

  In this example, the X coordinate is 7.36, Y is 4.46, and Z is 0.
- Wait Until CompleteIf the Wait Until Complete box is checked, the token will be held in the activity until this task has been completed.

If you need to give a task executer a task that isn't currently available in the process flow library, you can use the Custom Task activity to build your own custom task. The Custom Task activity has a Task Type menu with a complete list of other tasks that are available in addition to the common ones that are available in the library.

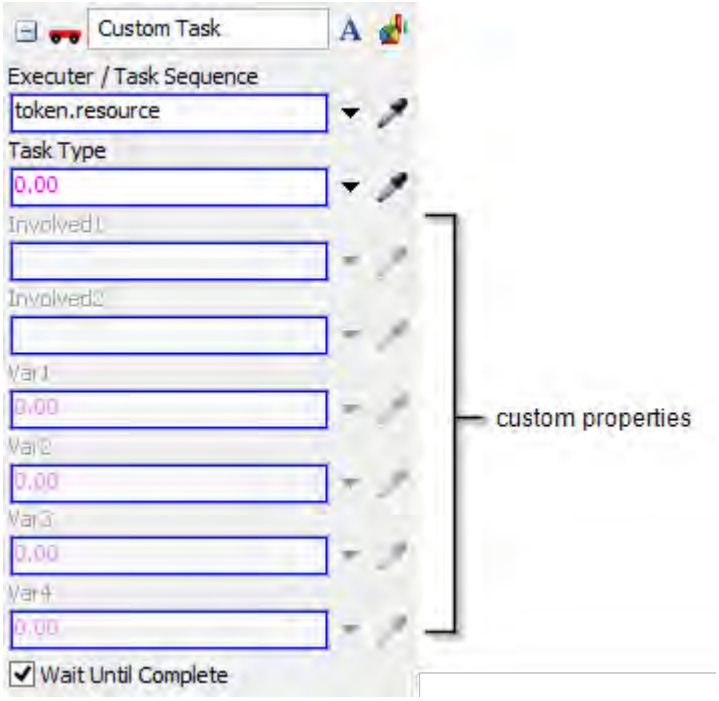
The Custom Task activity creates a task sequence with a single task and dispatches it to a task executer. If the task executer already has a task sequence, the task will be added to the end of a task sequence. Optionally, the task may be appended onto an already existing task sequence. See Create Task Sequence activity for more information. Also see Task Sequences for more general information about task sequences.

Connectors

The Custom Task activity only allows one connector out. See Adding and Connecting Activities for more information.

Properties

The following image shows properties for Custom Task Activities:



Each of these properties will be explained in the following sections.

-
- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button Aopens a window to edit the activity's background color and font properties. See Font for more information about this property.

  Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- Executer / Task Sequence Use the Executer / Task Sequence box to determine which task executer or task sequence should receive the task. See Executer / Task Sequence for more information about this property.
- Task TypeYou can select one of the available task types from the Task Type menu. This menu lists all possible task types. See the Task Sequence Types reference if you would like more information about each of the task types listed in this menu.
- Custom PropertiesThe custom properties are not available until after you select a task from the Task Type menu. The names of the properties will change based on the the task you selected. Each box will represent one of the parameters that are required or optional for that task. See the Task Sequence Types reference for a description of the parameters for each task type.
- Wait Until CompleteIf the Wait Until Complete box is checked, the token will be held in the activity until this task has been completed.

The Create Task Sequence activity will create an empty task sequence and dispatch it to a task executer. A task sequence is a series of actions (known as tasks) that can be assigned to a task executer (such as an Operator or Transporter). See Task Sequences for more general information about task sequences.

Be aware that you don't have to use the Create Task Sequence activity at the beginning of every task sequence. Using any of the task sequence activities in a process flow will automatically create a task sequence any time that task sequence is used in a simulation run, which means the Create Task Sequence isn't technically necessary.

With that in mind, there are a few reasons why you might decide to use a Create Task Sequence activity:

- To give some task sequences higher priorities than other. By default, task executers will complete task sequences and activities in the order they were received (first in, first out). However, you might want a task executer to complete more important tasks first. You can use the Priority property on the Create Task Sequence activity to cause the task executer to work on higher priority task sequences before lower priority task sequences. You can also use the Preemption menu to determine if requests for higher priority tasks should force a task executer to stop its current task sequence and perform the preempting task sequence.

- To prevent a task executer from stopping the current task sequence to work on another task. When using the task sequence activities on their own, a new task sequence is created and dispatched for each task on their own, which leaves room for another task sequence to interrupt the task sequence in between your activity tasks. When you use a Create Task Sequence activity, the task executer will need to finish the entire task sequence before working on a different task sequence.

- To create a task sequence and dispatch it later in the simulation run. You may want to create an entire task sequence but instead of dispatching it immediately, push it to a Global List and have a task executer pull it from the list. Using the Create Task Sequence activity with no assigned Task Executer / Dispatcher specified allows you to build the task sequence without dispatching it.

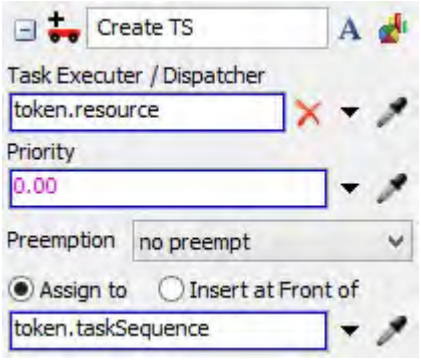
When you use a Create Task Sequence activity, you are essentially grouping a set of tasks together into a single unit, so the task executer will not do anything else until that entire group of tasks is finished. Once the task executer has completed all of the tasks in a task sequence, the task sequence will be deleted. See Deleting Task Sequences for more information.

Connectors

The Create Task Sequence activity only allows one connector out. See Adding and Connecting Activities for more information.

Properties

The following image shows properties for the Create Task Sequence activity:



Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button Aopens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- Task Executer / DispatcherUse the Task Executer / Dispatcher box to select the task executer or dispatcher that you want to assign the task sequence to. If you want to create a task sequence but not dispatch it, click the button so that this box now reads *None*. If you want to dispatch the task sequence at a later time, you should click the remove button and use Dispatch TS (Task Sequence) activity to assign the task sequence to a task executer later in your process flow. See Dispatch Task Sequence for more information about this activity.
- PriorityThe Priority box sets the priority level of the task sequence. By default, task executers will complete task sequences and activities in the order they were received (first in, first out). However, you might want a task executer to complete more important tasks first. You can use the Priority box to cause the task executer to work on higher priority tasks sequences before lower priority task sequences.

By default, all task sequences are assigned a priority of 0, but you can enter in any number in the Priority box. By default, Task executers will prioritize task sequences to perform higher priority task sequences first through their Queue Strategy property. For example, imagine you create a task sequence with a priority number of 1 and another with a priority number of 2. If both task sequences are waiting in a task executer's queue to be performed next, the task executer will work on the task sequence with the priority number 2 first.
Plan Out Your Priority Levels

If you decide to prioritize your task sequences, you might want to plan out what your priority levels will be in advance to save editing time down the road. Or you could reference values in a Global Table to easily define and edit priority levels later.

- PreemptionThe Preemption menu sets the preemption value of the task sequence. *Preemption* is when a task executer stops its current task sequence to work on the preempting task sequence. You can use the Preemption menu to determine if and how preemption occurs.

  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 it is doing and repair the machine, instead of finishing the transport operation. To simulate this behavior in your model, you'll have to make sure that the Operator's machine repair task sequence is set to preempt the task sequence for transporting flowitems. The Preemption menu has the following options:

  - no preempt - The current task sequence will not preempt any other task sequences.

- preempt - The current task sequence will preempt any active task sequences. The active task sequence (the one that is being preempted) will go back into the task executer's queue to be finished later.
- preempt, aborting active - The current task sequence will preempt any active task sequences. The active task sequence (the one that is being preempted) will be deleted and will not be finished later.
- preempt, aborting all - The current task sequence will preempt any active task sequences. The active task sequence and all other task sequences in the task executer's queue will be deleted and will not be finished later.

- Assign To Assigns a reference on the specified label/node to the created task sequence. See the Assign To section of Common Properties for more information.

The Dispatch Task Sequence activity can dispatch a task sequence to a task executer (such as an Operator or Transporter). Generally you won't need to use the Dispatch Task Sequence activity with your task sequences because the Create Task Sequence activity and other task sequence activities will be dispatched immediately by default.

However, you could use a Dispatch Task Sequence if you would prefer to build a series of task sequences first and dispatch them later. There are two different methods you can use to build a task sequence that is dispatched later:

1. You could use a Task Sequence Global List. (See Global Lists for more information.) Using this method, fixed resources can push task sequences to the list or you can manually create the task sequences and push them to the list. When ready, you can pull a task sequence from that list and then dispatch it to the appropriate task executer using this activity.

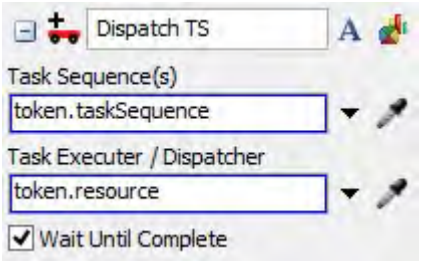
2. You could use a Create Task Sequence activity to create a task sequence but set the Task Executer / Dispatcher property to *None*. (See Create Task Sequence for more information about this property.) You can then build the entire task sequence without dispatching it to a task executer. When ready, you can dispatch the task sequence to the appropriate task executer using this activity.

Connectors

The Dispatch Task Sequence activity only allows one connector out. See Adding and Connecting Activities for more information.

Properties

The following image shows properties for the Dispatch Task Sequence activity:



Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button **A**opens a window to edit the activity's background color and font properties. See Font for mor eiinformation about this property.

- Statistics The Statistics button  opens the activity's statistics window. See Statistics for more information about this property.
- Task Sequence(s)Use the Task Sequence(s) box to select the task sequence or array of task sequences that will be dispatched. Task sequences that have already been dispatched will not redispatch.
- Task Executer / DispatcherUse the Task Executer / Dispatcher box to select the task executer or dispatcher that you want to assign the task sequence to.
- Wait Until CompleteIf the Wait Until Complete box is checked, the token will be held in the activity until this task has been completed.

The Split activity splits the token into multiple tokens and sends one out each outgoing connector. This activity is similar to the Create Tokens activity, except that the quantity of tokens to create and the destination of each token is determined by the number of outgoing connectors.

A Split ID (a reference to the original token) can be added to a user-specified label on each token. This is useful for coordinating with the other split tokens. See Coordination for information on coordinating tokens.

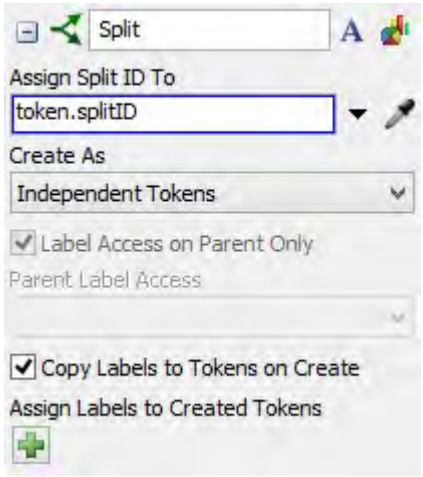
Connectors

The Split activity allows any number of connectors out. The Split does not allow other activities to be snapped to it as doing so would remove all connectors from the Split.

The number of outgoing connectors will affect the behavior of this activity. See Connectors and Coordination Activities for more about the unique behavior of connectors in Coordination activities.

Properties

The following image shows properties for the Split activity:



Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button Aopens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.

- Assign Split ID To A reference to the original token will be stored on the label specified in the Assign Split ID To box on each token.
- Create AsDefines the relationship of the original token to the created token(s). o Independent Tokens - Created tokens will have no association with the original token. o Child Tokens - Created tokens will be child tokens of the original token.
    - o Sibling Tokens - Created tokens will be sibling tokens of the original token. In other words, they will have the same parent as the original token.
- Label AccessFor information on Label Access, see the Label Access section of the Sub Process Flows page.

Tokens wait at the Join activity until one token arrives from each incoming connector. The token from the first connector is then released out the outgoing connector and the other tokens destroyed.

Specifying a Partition ID makes tokens wait only for other tokens in the same partition. This is useful for coordinating with other tokens of the same type. See Coordination for information on coordinating tokens.

Connectors

The Join activity only allows one connector out.

The number of incoming connectors will affect the behavior of this activity. See Connectors and Coordination Activities for more about the unique behavior of connectors in Coordination activities.

Properties

The following image shows properties for the Join activity:

Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button A opens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- Partition ID Defines which partition tokens will use.
- Use Max Wait Timer The max wait timer will be evaluated if the token has not pulled its required amount after the specified time. See Max Wait Timer for more information.
- Use Max Idle Timer The max idle timer will be evaluated each time a value is pulled from the list by the token. See Max Idle Timer for more information.

Tokens wait at the Synchronize activity until one token arrives from each incoming connector. This activity is similar to the Join activity, except that there will be an outgoing connector corresponding to each incoming connector. The token from the first incoming connector is then released out the first outgoing connector, the token from the second incoming connector is released out the second outgoing connector, and so on.

Specifying a Partition ID makes tokens wait only for other tokens in the same partition. This is useful for coordinating with other tokens of the same type. See Coordination for information on coordinating tokens.

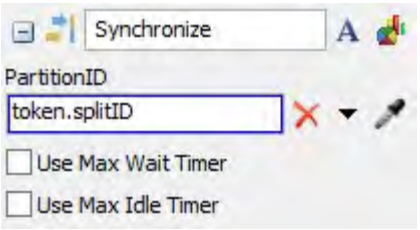
Connectors

The Synchronize activity allows any number of connectors out as long as the number of outgoing connectors matches the number of incoming connectors. The Synchronize does not allow other activities to be snapped to it as doing so would remove all connectors from the Synchronize.

The number of incoming and outgoing connectors will affect the behavior of this activity. See Connectors and Coordination Activities for more about the unique behavior of connectors in Coordination activities.

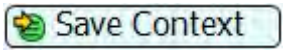
Properties

The following image shows properties for the Synchronize activity:



Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button **A** opens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- Partition ID Defines which partition tokens will use.
- Use Max Wait Timer The max wait timer will be evaluated if the token has not pulled its required amount after the specified time. See Max Wait Timer for more information.
- Use Max Idle Timer The max idle timer will be evaluated each time a value is pulled from the list by the token. See Max Idle Timer for more information.



The Save Token Context activity saves the current activity context of one or more tokens to a user-defined label on those tokens. The saved context includes the token's current activity, as well as any activityrelated data. For example, if a token is in a Delay activity, it will save off the total delay time as well as how far into the delay the token is currently.

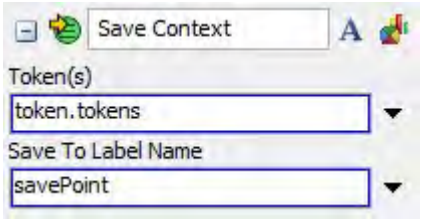
The Save Token Context activity is used for process flow preemption. See Preemption for information on how to implement preemption.

Connectors

The Save Token Context activity only allows one connector out. See Adding and Connecting Activities for more information.

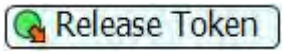
Properties

The following image shows properties for the Save Token Context activity:



Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button Aopens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- Token(s) Define the target token(s) whose context you want saved. See Connecting Preempting Process Flows for more information.
- Save To Label NameHere you enter the name of the label you want to save the context on. This label will be saved on the target token(s) (not necessarily the token executing the Save Token Context activity).



The Release Token activity aborts the current activity of one or more tokens and sends them to a new activity.

This activity is used in process flow preemption. See Preemption for information on how to implement preemption.

Connectors

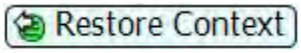
The Release Token activity only allows one connector out. See Adding and Connecting Activities for more information.

Properties

The following image shows properties for the Release Token activity:

Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button A opens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- Token(s) Define the target token(s) who you want to release. This should usually match the Token(s) field of a Save Token Context activity. See Connecting Preempting Process Flows for more information.
- Release To Define the activity you want to release the token(s) to. Usually you will use the sampler button to directly choose the destination activity.



The Restore Token Context activity restores one or more tokens to a saved activity context .

The Restore Token Context activity is used in process flow preemption. See Preemption for information on how to implement preemption.

Connectors

The Restore Token Context activity only allows one connector out. See Adding and Connecting Activities for more information.

Properties

The following image shows properties for the Restore Token Context activity:

Each of these properties is explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button Aopens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- Token(s) Define the target token(s) whose saved context you want restored. This should usually match the Token(s) field of a Save Token Context activity. See Connecting Preempting Process Flows for more information.
- Saved Label NameDefine the name of the label that has context information saved on it. This should match the label name defined in a previous Save Token Context activity.

A shared asset is a finite resource that tokens may claim or release at certain points in the process flow. Shared assets look a lot like standard activities but there are some distinct differences. A shared asset cannot be part of a stacked block or snapped to any other activity. Tokens do not physically move into a shared asset, instead, activities like the Acquire, Push To List or Enter Zone reference a shared asset which will be used in that activity's logic. They are also colored differently than activities to show that they function a little differently, as shown in the following image:

Shared assets can impose constraints on the tokens by making the token wait if the requested asset is unavailable. A real-life example of this would be a certain tool that is shared by three different work stations. If one station needs the tool while it is already claimed by another station, that station must wait. In that same vein, if a token needs an asset in order to move on to the next activity, the token will wait at its current activity until the asset is available. There are three types of shared assets:

- Resource - Represents a limited supply of some resource that can be acquired and released. It can be used to simulate a supply of goods, services, time, materials, employees, etc.
- List - Allows you to add or remove tokens, flowitems, task executers, numbers, strings, etc. to a list. This is a useful asset for syncing multiple tokens within a process flow or as a more dynamic Resource. Process flows can use a list that is local to the process flow itself or could be tied to a Global List in the simulation model.
- Zone - Can collect statistical information not available for standard activities. It can also restrict access to a section of the process flow based on certain statistics or other criteria.

Global and Local Types

All shared assets have the ability to be defined globally or locally within their process flow. In the case of a General Process Flow, the shared asset will always be defined globally. If the shared asset is in a Fixed Resource, Task Executer or Sub Flow Process Flow, there may be more than one instance of the process flow at any given time. The two types behave as follows:

- Global - If the shared asset's Type is set to *Global*, all tokens in all instances of the process flow will compete for the same asset.
- Local - If the shared asset's Type is set to *Local*, then there will be a different asset for each instance of the process flow. Only tokens within the same instance will compete for the same asset.

For example, if a Resource is defined as *Local* and it is referencing an Operator in the 3D model and there are 3 instances of the process flow, then each instance will be associated with a different Operator. This means there will be 3 Operators in the 3D model, one for each instance. When a token attempts to acquire a resource, it will only be able to acquire the Operator associated with that instance. Allocating Global Shared Assets

It is possible to allocate from a shared asset that is not in the same process flow as the requesting activity. However, this is only allowed if the shared asset is in a General Process Flow.

List

The List is a shared asset that can represent a list of tokens, flowitems, task executers, task sequences, numbers, strings, etc. Using the Push To List and Pull From List activities, the contents of a list can be dynamically updated during a simulation run. For example, a list could represent flowitems or tokens that are waiting to go to the next available object or activity. When an object or activity becomes available, the flowitem or token can be pulled from the list and sent to the downstream object or activity.

Lists can be either be internal to a process flow or they can link to a Global List in the 3D simulation model. See Global Lists for more information about how lists work in FlexSim generally.

Why Use Lists in Process Flow

Using Lists, you can:

- Sync tokens or objects
- Organize groups of objects
- Track custom statistics
- Use search concepts like filtering and prioritization for making choices in the model

Lists have a lot of functional overlap with other shared assets. However, one of the unique advantages of using Lists within Process Flow is for token synchronization. Token synchronization means that tokens can wait for each other at defined points by having one activity push a token onto a List, while another activity pulls the token from the List.

To give a real-world example of a business system that would need token synchronization, imagine you want to simulate the loading dock of an order picking warehouse. When the warehouse receives notice that a truck is on its way, the employees need to begin staging the orders that have already been picked for shipment. It's possible that the truck could arrive before the employees are finished staging the orders on pallets, in which case the truck will need to wait until the orders are ready. However, it's also possible that the employees might finish staging the orders before the truck arrives, in which case the orders will need to wait until the truck arrives.

If you represent the truck as a single token and each order as a single token, then the problem is as simple as sending each order token to a Push To List activity that pushes the order token to a list and sending the truck token to a Pull From List activity that will pull all of the order tokens from the list. If the orders arrive first, the order tokens will remain in the Push To List activity until the truck arrives and pulls them from the list. If the truck arrives first, it will wait at its Pull From List activity until it has pulled all of the orders from the list.

The Lists and Resources Tutorial gives a more hands-on example of why and how to use lists in a process flow and simulation model.

Creating, Linking, and Accessing Lists

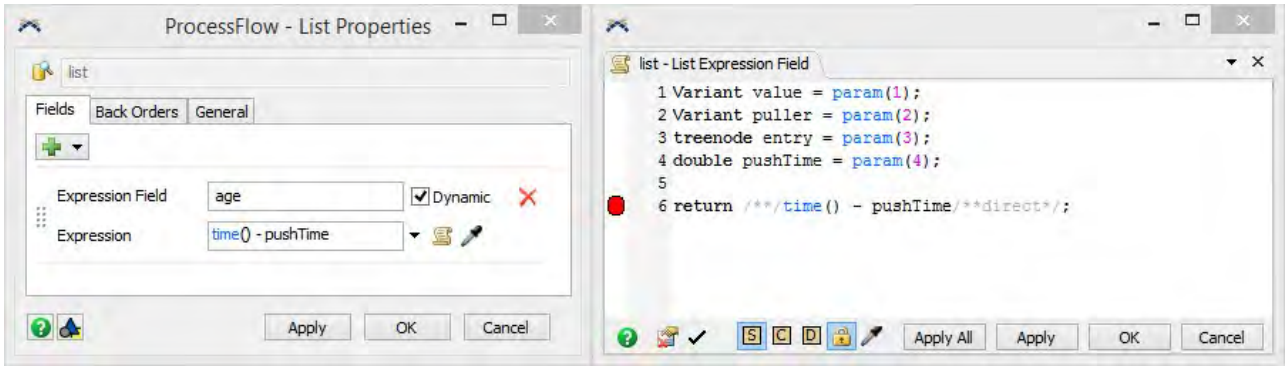
A List shared asset can be used to create either internal lists or link to global lists:

- Global List - Links to a Global List that has already been created in the 3D simulation model. Use a Global List if your list needs to be linked to a list used by objects in the 3D model. See Global Lists for more information about creating Global Lists.
- Internal List - The default method. A list that is created and owned by the List shared asset. Use an internal list if your list does not need to be accessible to objects in the 3D model.

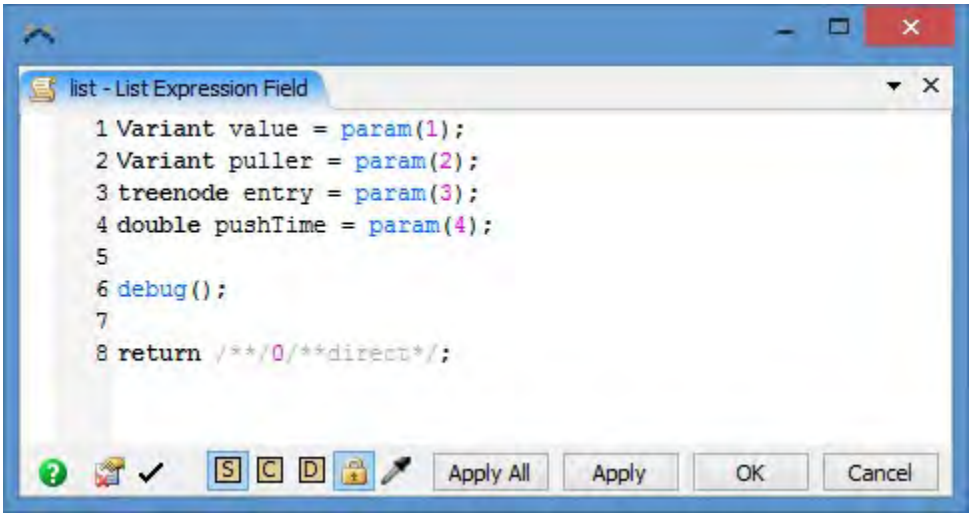
When using a local list, the list can either be defined globally or locally to the process flow. For more information on these types see Global and Local Types.

Debugging Lists in Process Flow

Normally, when you need to debug code in one of your pick options, you would click in the far left column of the code editor to create a breakpoint:



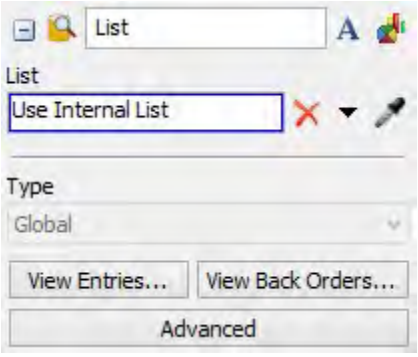
However, if you are using an Internal List in a List Shared Asset and you need to debug a field, you need to use the debug() command:



This also applies to the Back Order Reevaluation Events.

Properties

The following image shows properties for the List shared asset:



Each of these properties will be explained in the following sections.

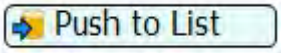
- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button opens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- ListUse the List box to link to a Global list or create an internal list. See Creating, Linking, and Accessing Lists for more information about the difference between a global list and an internal list.

  By default, all lists are internal, which means that this box should say *Use Internal List*. If you want to use an internal list and the box has text other than this in it, click the remove button to revert back to using an internal list.

  To link to a Global List, click the arrow next to this box to open a menu. Point to Global List and select the name of the list from the menu or use the sampler and sample the list from the Toolbox.
- Type The Type menu determines whether the list will be define locally or globally. This menu is only available if you have created an Internal list and the List shared asset is not contained in a General Process Flow. See Creating, Linking, and Accessing Lists for more information.
- View Entries... The View Entries... button will display all of the values currently on the list. This button functions the same as the View Entries... button on the General tab on the Global List Properties.
- View Back Orders... The View Back Orders... button will display all the back orders who are pulling from the list. This button functions the same as the View Back Orders... button on the General tab on the Global List Properties.
- Advanced The Advanced button opens the List properties page. Use this page to edit the list's properties.

Push to List

This activity will add something (tokens, flowitems, task executers, task sequences, numbers, strings, etc.) to a List. For detailed information on how Lists work, see the List shared asset and Global Lists.



When this activity is first added to a process flow, a red exclamation mark shows up to the right of the activity notifying you that a link to a List is required for this activity to function correctly. To link this activity to a List, click the exclamation point and then click on a List object to select it. Alternatively, you can create a dynamic reference by setting the value for the List property (such as using a label value or looking up a value in a global table).

By default, this activity does not finish until the entry is pulled from the list. This means that a token will remain at a Push to List activity until the Push Value has been pulled off the list. The default Push Value is the entering token. Pushing the entering token to a list and waiting for it to be pulled allows for easy token synchronization. (See Why Use Lists in Process Flow for more information about token synchronization.) Both tokens sync with each other through the pushing/pulling mechanism, and the pushing token gets access to the token that pulled it (through the Assign Puller To property), and vice versa (through the Pull From List's Assign To property).

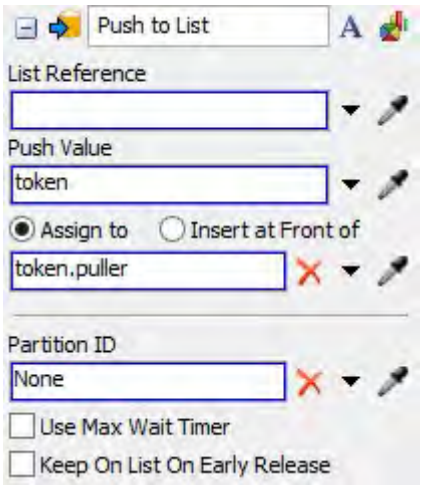
You can use the Max Wait Timer to cause the token to push a value to a list and then move on to additional activities.

Connectors

The Push to List activity allows multiple connectors out. However, tokens automatically released from this activity will be released through the first connector. Only manually released tokens have the opportunity to exit out a different connector. See Adding and Connecting Activities for more information.

Properties

The following image shows properties for the Push to List activity:



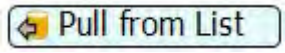
Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button **A** opens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- List The List box displays the name of the List to which this activity will push. If you linked to a List by clicking on the red exclamation mark next to the Push to List activity, then the name of the List you linked to will show up in this box. You can also manually type in the name of the List or create a dynamic reference here in this box.

- Push Value The object or value that is pushed to the list. By default, the entering token will be pushed to the list. But it is also possible to push an object, number, string, node, or array to the list when the token enters the activity instead.
- Assign Puller To Assigns a reference on the specified label/node to the object that pulled the entry from the list. See the Assign To section of Common Properties for more information.
- Partition ID This will push the value to a certain partition within the list. For more information see Partitions.
- Use Max Wait Timer The max wait timer will be evaluated if the token has not pulled its required amount after the specified time. See Max Wait Timer for more information.
- Keep On List On Early ReleaseIf checked and the token is manually released before it's Push Value was pulled from the list, the pushed value will be kept on the list to be pulled at a later time.

Pull From List

This activity will pull one or more values from a List. For detailed information on how Lists work, see Global Lists.



When this activity is first created, a red exclamation mark shows up to the right of the activity notifying you that a link to a List is required for this activity to function. This link may be a direct pointer which can be created by clicking on the exclamation mark and then clicking on a List object, or the reference may be dynamic by setting the value for the List property.

By default, this activity does not finish until the it has successfully pulled everything that it required from the list. This means that a token will remain at a Pull from List activity until the Require Number of values has been pulled off the list.

You can use the Max Wait Timer to cause the token to attempt to pull a value from the list and then move on to additional activities.

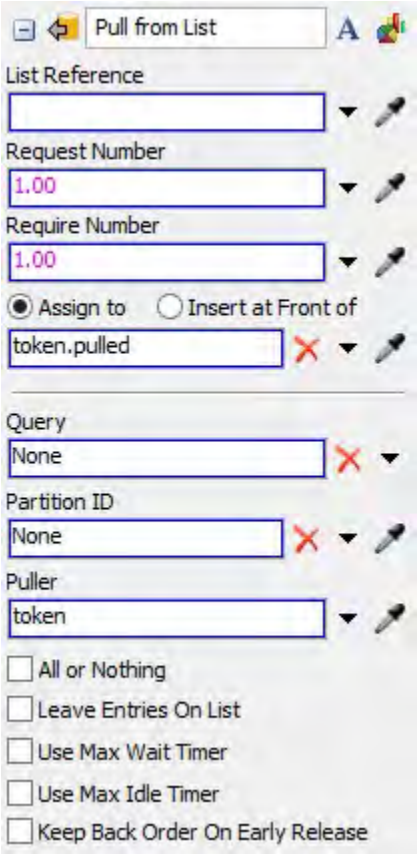
The process of pulling a value from a list does not cause that token or object to be physically moved. Instead, a reference to the pulled value can be stored on the pulling token. This reference can be used later in other activities.

Connectors

The Pull From List activity allows multiple connectors out. However, tokens automatically released from this activity will be released through the first connector. Only manually released tokens have the opportunity to exit out a different connector. See Adding and Connecting Activities for more information.

Properties

The following image shows properties for the Pull From List activity:



Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button **A**opens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- List Reference The reference to the List that this activity is pulling from.
- Request Number This defines how many values you *want* to pull. The request number must be greater than or equal to the Require Number. If more values than the require number are available when the token initially pulls, then the token will pull those additional values, up to the request number, and then immediately release the token.
- Require Number This defines the minimum number of values that you *need* to pull. If the list has insufficient values for the require number, a back order will be created and the token will wait in the activity until the required number of values is available.

    If the require number is zero then the activity will always finish immediately, returning the amount that it successfully pulled. In this case no back order will be created.

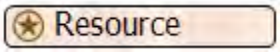
If the request number and the require number are both zero, then the pull operation will become a mere querying operation. The pull will retrieve all objects on the list that meet the query filter, but it will not remove any values from the list.

- Assign To Assigns a reference on the specified label/node to the value(s) were pulled from the list. See the Assign To section of Common Properties for more information.

- Query The conditions which specify which values to pull. This takes the form of an SQL query. You can use the button for help in constructing this query.
- Partition ID Defines which partition in the list to pull from.
- Puller Defines the puller object. This may be important if the query accesses fields that are dynamic and based upon the puller.
- All or NothingIf checked, no values will be pulled from the list until the entire Require Number of values can be pulled at one time. Otherwise, values will be pulled as they become available.
- Leave Entries On ListIf checked, all values that are 'pulled' from the list will remain on the list. The values will still be assigned to the Assign To label/node. This can be useful for querying the list when specific data is added, or for using the list as an informational database. This option behaves like the All or Nothing in that no values will be 'pulled' until all values are available.
- Use Max Wait Timer The max wait timer will be evaluated if the token has not pulled its required amount after the specified time. See Max Wait Timer for more information.
- Use Max Idle Timer The max idle timer will be evaluated each time a value is pulled from the list by the token. See Max Idle Timer for more information.
- Keep Back Order On Early ReleaseIf checked and the token is manually released before it was able to pull its required amount, the back order will be kept on the list to be fulfilled at a later time.

Resource

This shared asset represents an asset with a limited supply that can be acquired and released. For example, three manufacturing stations might all share one tool. The tool is modeled here as a Resource. When a station uses the tool, it would be acquiring the Resource. Once the Resource has been acquired, nothing else can acquire it until the acquiring token has released it. Resources are useful for modeling timing constraints, where multiple tokens need to use a limited pool of assets.



A Resource is acquired by an Acquire Resource activity and released by a Release Resource activity.

Resource Operation Modes

The Resource can operate in two modes, as follows:

- Numeric Mode When the Resource operates in numeric mode, it simply keeps track of a number in determining its availability. This number starts at zero. When activities acquire the Resource, it increments the number, and when activities release the Resource, it decrements the number. If an Acquire activity is requesting a quantity that is greater than the Resource has available as defined by the Count property, then the token will be unable to acquire the resource.

  To put the Resource into numeric mode, define its Reference property as *Numeric*.

  When in numeric mode, the Count property can dynamically change during the simulation. It will be reevaluated every time a token attempts to acquire a resource and whenever a resource is released. This can be useful if you want to simulate a resource that changes based upon the time of day or other model parameters. Even if the total resource's count drops below the number of resources that have been acquired, tokens will keep their acquired value until they release their resource in a Release activity.

  Numeric mode can also be used to simulate non-discrete resource quantities. For example, you can give the Resource a Count of 5.5, and acquire quantities like 1.5, etc.

- Object Mode When the Resource operates in object mode, it links to a discrete set of objects in the model. This may be a TaskExecuter, FixedResource or other 3D object, an array or a Group of objects. When using a TE or FR object, the Count property can duplicate objects when needed. IWhen using arrays or Groups, the Count property defines how many objects in the set can be used. When tokens acquire the Resource, the Resource gives them back a reference to specific acquired object(s), instead of just managing an availability number. In this mode, the Count is established at the start of the simulation and cannot change after that.

Duplicating Objects

If the Resource is in object mode and its Reference property references an object in the model, the Count property will be used to duplicate the object by $_{\text{count - 1}}$, where the referenced object is the first resource available. Once duplicated, the TaskExecuter or FixedResource objects may be moved in the 3D view and their properties may be individually modified. Properties set on the linked resource will NOT be pushed down to the duplicated objects once they have been created. To propagate changes from the referenced object to

the duplicated objects (ie labels), set the Count to 1, reset the model, then set the Count back to the original value. Alternatively, you can select all of the objects and modify values in Quick Properties or use the Edit Selected Objects window.

It may be useful to tie the Count property to a Process Flow Variable so that the count may be changed through the Experimenter over multiple replications. The resource will add new objects to the model and remove unused objects as needed.

Adding and Removing Resources

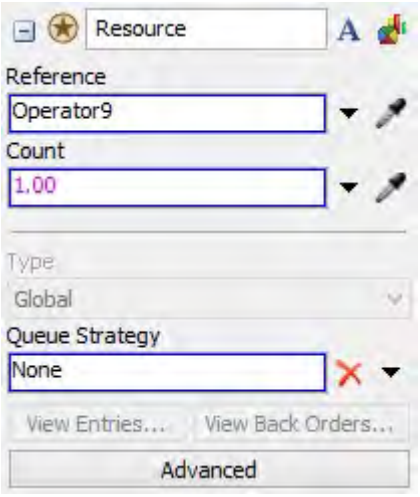
If the Count is greater than the previous Count, the referenced object will be duplicated and the new object will be at the same location as the referenced object. If the Count is less than the previous Count, the last duplicated objects will be deleted. This means if you duplicated objects and then placed them throughout the model, the Experimenter should only remove resources not add them as doing so may cause your objects to be in incorrect locations.

Using A List

By default, acquire requests are processed in FIFO (first in first out) order, and in object mode the resource objects themselves are distributed in FIFO order. If a Queue Strategy is used on the Resource or if any Acquire Resource activity that references this Resource uses a Query, the Resource will switch over to using a List. This List will be internally owned by the Resource. This allows the Resource to utilize the built in SQL functionality of the List to determine which tokens to allocate resources to first and which resources should be allocated. Through the Advanced button, you can set up additional fields for the List that can be used in the Query and Queue Strategy properties.

Properties

The following image shows properties for the Resource shared asset:



Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.

- Font The Font button opens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- Reference This box will only be evaluated on reset. It may be either *Numeric*, a reference to an object in the 3D model, a Group, an array of objects, integers, doubles or strings or any other object or node.
- Count The number of resources that are available to acquire. If the Reference property is set to a Group, a Dispatcher or array of objects, the count should be less than or equal to the number of objects in the Group/Team/array. If a Group/Team/array has 10 objects in it and the count is set to 5, only the first 5 objects in the Group will be used. Reorder the objects to specify which 5 objects will be used. This can be a very useful features when implemented with the Experimenter. The Count can be set to a Process Flow Variable which is then set as an Experimenter variable which changes for each scenario. If Reference is not *Numeric*, this property will only be evaluated on reset. If Reference is a direct reference (ie to a TaskExecuter or FixedResource), the object will be duplicated. See Duplicating Objects for more information.
- Type Resources can be defined globally or locally. If the Resource is in a General Process Flow or connected to a Group or an array, the Resource will be globally available and this property will be grayed out. Otherwise, it can be set to Local so only tokens in the same Process Flow instance will be competing for the same resources. If Type is set to Local and Reference is linked to a Dispatcher, Task Executer or Fixed Resource object, the object will be duplicated for each instance of the Process Flow. For more information see Global and Local Types.
- Queue StrategyIf a number of activities are trying to acquire resources and no resources are available, their acquire requests will stack up in a queue. By default this is FIFO (first in first out or first come first serve). Defining a Queue Strategy gives you control of who is given resources first. The queue then becomes back orders on a List and a SQL command is used to determine the order of the queue.
- View EntriesOnly valid if the Resource is using a list (see Using a List) this will display all of the available resources.
- View Back OrdersOnly valid if the Resource is using a list (see Using a List) this will display all of the tokens attempting to acquire resources.
- AdvancedOnly valid if the Resource is using a list (see Using a List). Opens the List properties page.

Acquire Resource

This activity acquires resources from a Resource shared asset.

When this activity is first created, a red exclamation mark shows up to the right of the activity notifying you that a link to a Resource is required for this activity to function. This link may be a direct pointer which can be created by clicking on the exclamation mark and then clicking on a Resource object, or the reference may be dynamic by setting the value for the Resource Reference field.

When acquiring a resource, the acquired value (number, task executer, fixed resource, etc.) will be assigned to a label on the acquiring token. This value or reference can be used in other activities to affect the logic of the process flow.

Acquiring Multiple Resources

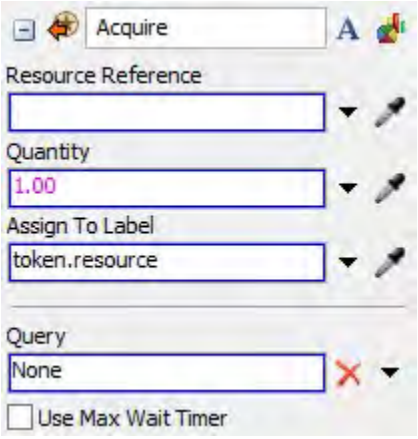
If multiple resources are acquired by a single activity or if multiple Acquire activities share the same Assign To Label, an array will be created on the label with references to all of the acquired resources. The most recent acquired resource will appear as the first index in the array (index 1) and previously acquired resources will be pushed to the end of the array in sequential order.

Connectors

The Acquire Resource activity allows multiple connectors out. However, tokens that successfully acquire a resource and are automatically released from this activity will be released through the first connector. Only manually released tokens have the opportunity to exit out a different connector. See Adding and Connecting Activities for more information.

Properties

The following image shows properties for the Acquire Resource activity:



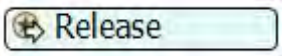
Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button A opens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- Resource Reference The Resource shared asset this activity is linked to.
- Quantity The number of resources to acquire. If the Resource does not have sufficient resources available, the token will wait in this activity until the token is able to acquire the total number of resources returned by this field. If the Resource is in object mode then this field will be read as an integer. If Resource is operating in numeric mode then this field will be evaluated as a floating point or real number.
- Assign To Label Once the token has acquired from the resource, a reference to what was acquired will be added to the label on the token as defined by the Assign To Label box as either a single reference or an array of references. If data already exists on the label, the references will be prepended to the beginning of the array. This must be a token label as the label will be used to later release the resource using a Release Resource activity. However, it does not have to be a label on the entering token and could reference a parent, sibling or other token's label.

  If the Resource is in numeric mode, the label will be assigned the quantity acquired which will always be the same as the Quantity box.
- Query The conditions which specify which resources to acquire. This takes the form of an SQL query, which can be constructed from the drop down menu. Only the WHERE and ORDER BY SQL keywords are available. This is only applicable in object mode. For more information see the SQL Queries page.
- Use Max Wait Timer The max wait timer will be evaluated if the token has not acquired its resources after the specified time. See Max Wait Timer for more information.

Release Resource

This activity releases resources from a token that have been acquired by an Acquire Resource activity. The resources will automatically return to the Resource activity they were acquired from. If the Resource(s) Assigned To does not return a label that has a resource assigned to it, this activity will do nothing and the token will be immediately released to the next activity.



Once a resource is released back to a Resource shared asset, the Resource will go through and check any tokens waiting in an Acquire activity, attempting to acquire a resource, and see if it's able to acquire its requested resource.

Releasing Resources When Multiple Were Acquired

If multiple resources were acquired by a single Acquire activity while in object mode, an Array of references to the acquired resources will be created. When releasing these resources using the Release Resource activity, not all of acquired resources will be released by default. By default, the Resource(s) to Release box is set to *Last Acquired*. If two Operators were acquired by one Acquire activity, the *Last Acquired* option will only release one of the operators. This can allow you to easily begin an operation with multiple resources and release them one at a time.

You can acquire resources from multiple Resource shared assets and assign them to the same label. Doing so will create an array where the last acquired resource will be index 1 of the array. For example, a label with the resources {/Operator3, /Operator2, /Operator1} acquired Operator3 last and Operator1 first. Releasing the *Last Acquired* will release Operator3 leaving you with the array {/Operator2, /Operator1}. When Operator1 and Operator2 are released, the label storing these resources will be deleted.

Releasing Specific Resources

There are times where it may be necessary to release a specific resource from an array of acquired resources. If you know which index the resource is at, then you can pass the index in to the Resource(s) to Release box. However, this may not always be the case. There are two other ways of releasing a resource:

- By Resource Shared Asset Pass in a reference to the Resource Shared Asset that was used to acquire the resource. All resources acquired to the label passed into the Resource(s) Assigned To box from the Resource Shared Asset will be released. This works in both object mode and numeric mode.
- By Object Pass a reference to the acquired object into the Resource(s) to Release box. The Release activity will search through the label passed into the Resource(s) Assigned To box and find the first reference that matches (if any).

For example, three resources are acquired from Cleaning Ops, Maintenance Ops and Technical Ops and each are assigned to the label "operators" in some random order. If you want to release the operator associated with the Maintenance Ops, you could either use a direct reference to the Resource Shared Asset by using the Sampler for the Resource(s) to Release box or by typing `getactivity(processFlow, "Maintenance Ops")`. If the maintenance operator was a 3D object in the model, MODEL:/MaintenanceOperator, you could also pass a reference to that operator. This could be done through the code `model().find("MaintenanceOperator")` or using the Sampler to get a direct reference.

Accessing Resources from an Array

As you acquire multiple resources on the same label, as in the example above, you may need to access those resources to give them tasks to complete. If your resources are assigned to the label *resource* with the value {/Operator3, /Operator2, /Operator1}, using `token.resource` in a property like Executer / Task Sequence of one of the Task Sequence activities will return the reference to the resource at the first index of the array. In this case, Operator3. To access one of the other operators, you can use a *Value/Object From Array* pick option. or enter the code `token.resource[2] //Accesses the 2nd resource`.

Releasing Numeric Resources

Multiple numeric resources can also be acquired and assigned to the same label. Doing so will create an array on the label. For example, if three Acquire activities are used to acquire 1 resource from three different Resource shared assets, and each resource is assigned to the label *resource*, the resource label on the token will display an array of {1.00, 1.00, 1.00}.

In this case where quantities are equal, it is not guaranteed that the resources will be released in the correct order when using pick options like *First Acquired* and *Array/Range of Resources*. To ensure the correct resource is released you can pass a reference to the Resource shared asset into the Resource(s) to Release property.

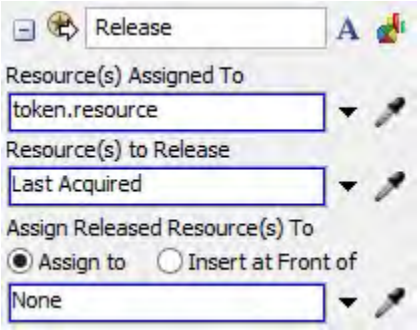
However, if these same Acquire activities acquire different quantities, say 1, 2 and 3, the Release Resource activity will be able to correctly identify which resource is being released using the *First Acquired* and *Array/Range of Resources* pick options.

Connectors

The Release Resource activity only allows one connector out. See Adding and Connecting Activities for more information.

Properties

The following image shows properties for the Release Resource activity:



Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.

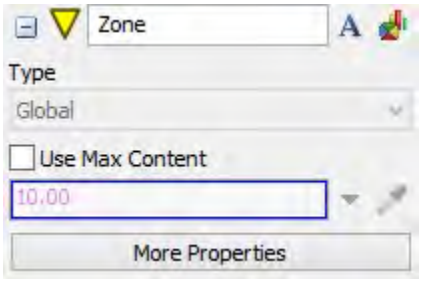
- Font The Font button opens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- Resource(s) Assigned ToWhere the resource(s) to release are located. The return value of this field should be a label on a token where an Acquire Resource activity assigned the acquired resource to.
- Resource(s) to ReleaseThe conditions which specify which resources to release. If multiple resources were acquired and assigned to the same label, the label will display an array of all the acquired resources. To release a specific resource from the array, return the index of the array or an array of multiple indexes. You can also return a reference to the Resource shared asset that was acquired from. Doing so will search the array for resources acquired from the shared asset and release all of them. This can be useful when you are unsure of which index to release, or when releasing Numeric Resources as described above in Releasing Numeric Resources.
- Assign Released Resource(s) To Assigns a reference on the specified label/node to the released resource or resources. This can be helpful if you need to maintain a reference to a resource for use in a later Acquire activity. See the Assign To section of Common Properties for more information.

Zone

The Zone keeps statistics for a group of activities within process flow. Optionally, it can restrict access to those activities, based on the tokens within those activities. See Zone Reference for more information.

Properties

The following image shows the property for the Zone shared asset:



Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button Aopens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- Type Zones can be available globally or locally. The Zone keeps statistics and enforces its Max Content and Constraints per instance. For more information see Global and Local Types.
- Use Max Content The Use Max Content checkbox determines whether or not the Zone will use a maximum content. The Max Content box is enabled when this box is checked, and disabled when it is not.
- Max Content If Use Max Content is checked, the value specified in this box determines the maximum number of tokens that can be in the Zone at any given time.
- More Properties The More Properties button opens the Zone Properties Window.

Zone Reference

This topic describes, in detail, the available functionality provided by the Zone.

The following is the table of contents for this topic:

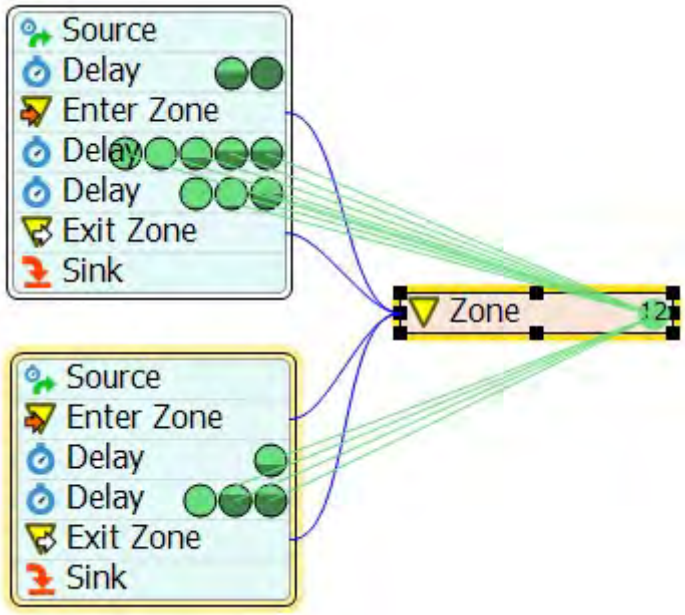
o Subset Statistics o Partition Basics o Partition Statistics o Calculation Basics o Calculation Statistics o Example Subset Calculations o Example Partition Calculations

- Restricting Access o Max Content o Constraints o Subset Constraint Examples o Partition Constraint Example
- Advanced Topics o Custom Constraint Check o Queue Strategy o Enforcing Queue Order o Evaluate Calculations On Exit o Proxy Entry and Exit o Non-Member Exit

Zone Basics

The Zone represents any kind of area in a Process Flow. Tokens enter and exit the Zone through Enter Zone and Exit Zone activities. A token that has entered a Zone is said to be a member of that zone until it exits. The Zone keeps a list of all its members and collects statistics about them.

The following image shows an example Zone. The blue curves indicate that the Enter Zone and Exit Zone activities reference the Zone. The green lines indicate that the tokens belong in the Zone. This Zone has 12 members in 4 activities. Notice that two tokens have not yet passed over an Enter Zone activity, so they are not yet members of the Zone.



The Zone can optionally group its members into categories. When a token enters a Zone, the Zone analyzes the label values of the token and puts it in the appropriate categories. The Zone records many statistics for each category, including the number of tokens in that category, and how long tokens in that category stay in the Zone.

You can limit the number of tokens that are allowed in the Zone. If the Zone is full, tokens wait on the Enter Zone activity until another token exits the Zone.

You can also limit the number of tokens allowed in any given category. If a token belongs to a specific category, and that category is full, then the token must wait until there is room in the category before entering the Zone.

You can also limit the total of a given label value that is allowed in the Zone. For example, if each token has a weight label, you can limit the total weight allowed in the Zone. Each token waits on the Enter Zone activity until there is enough room in the Zone.

In summary, the Zone can collect statistics and restrict access to a set of activities.

Token Labels: A token's labels are not affected by entering or exiting a Zone. The Zone does not add any labels to the token, or change any label values. Back to top

Default Statistics

As a shared asset, the Zone keeps the statistics described in Shared Assets Statistics. These statistics are updated in four steps:

1. A token arrives on an Enter Zone activity.
   The RequestInput is incremented. If the token cannot immediately enter the Zone, the RequestContent is incremented.
2. A token leaves an Enter Zone activity.
   The RequestOutput is incremented, and the RequestStaytime is updated. If the token has waited to enter the Zone, the RequestContent is decremented.
3. A token enters the Zone.
   The Input and Content are incremented.
4. A token exits the Zone.
   The Output is incremented, the Content is decremented, and the Staytime is updated.

It is possible for a token to leave the Enter Zone activity without actually entering the Zone. This occurs if you call releasetoken() on the waiting token (perhaps when a Max Wait Timer fires). In this case, the third and fourth steps do not occur. Back to top

Categories

The Zone can group its members into categories. You can create categories in two ways: by creating one or more Subsets, or by using Partitions. You can specify multiple Calculations for each category, which allow you to generate additional statistics about each category. You can also create constraints on each category, to limit the number of tokens allowed in a category.

Subsets

When you create a Subset, it creates a single category in the Zone. The tokens that belong in that Subset meet specific criteria, based on the token's labels. You can define the criteria, which are called the Token Selection Criteria, for each Subset. For example, if tokens entering the Zone have a weight label, a Subset's criteria might require the value of that label to be greater than a certain value, or within a certain range. The

Zone will then keep statistics on the tokens that meet that criteria. A token can belong to many subsets simultaneously, as long as it meets all the criteria for those subsets.

Partitions

You can also use Partitions to create categories in the Zone. A Zone's Partitions are very similar to a List's partitions. If you use Partitions, each token will belong in a single partition, based on a Partition By expression. The Zone evaluates the expression for each token, and uses the result to place the token in a specific Partition. For example, if tokens entering the Zone have a type label, the Partition By expression might simply return the value of that label. Then the Zone would put each token into a Partition corresponding to its type. You would then have access to statistics about each type of token within the Zone.

Subsets and Partitions: You can use Subsets and Partitions simultaneously. When a token enters the Zone, the Zone will use each Subset's criteria to determine if the token is part of that Subset. Then the Zone will apply the Partition By expression to the token, and place the token in the appropriate partition.

Calculations

Both kinds of categories (Subsets and Partitions) can have Calculations. A Calculation allows you to total a label value, or an expression based on label values, for all the tokens in that category. For example, if tokens entering the Zone have a weight label, you could use a Calculation to get the total weight of all tokens in a particular category. The total is updated as tokens enter and exit the Zone.

Constraints

Both kinds of categories can also have Constraints. A Constraint allows you to limit the capacity of a specific category in the Zone. You can use a constraint to only allow a certain number of tokens in a category. You can also use a constraint to limit the total of a Calculation. In this case, the Zone will ensure that the total of a Calculation never exceeds the limit you have set.

Back to top

Subset Basics

By default, a Zone does not have any Subsets. In order to add a subset, click the More Properties button in the Zone's Quick Properties panel. This opens the Zone Properties window. On the Subsets tab, click the Add button.

When you add a Subset, you can specify a name and define the Token Selection Criteria.

The name is arbitrary. However, the name will appear in other places like the Constraints tab, so make sure that it is unique and helpful.

Token Selection Criteria

The Token Selection Criteria is the test for this Subset. It is this expression that truly defines the Subset. The expression should be a boolean expression consisting of label names, literal value, and arithmetic operators. If there is a space in a label name, surround it with square brackets: [Label Name].

By default, or if the Token Selection Criteria is removed, then all tokens will be included in the Subset. The expression will be replaced with the text "Include all tokens".

When a token arrives on an Enter Zone activity, the Zone will determine which Subsets it belongs to. The Zone assumes that the label values used to categorize the token will not change while it is on the Enter Zone activity, or while the token is in the Zone.

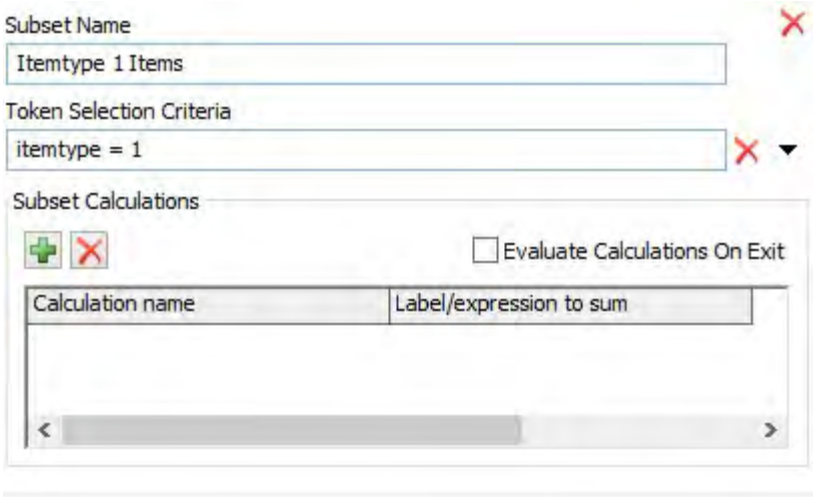
Calculations

You can also define a set of Calculations for each Subset, which will be discussed later. They are not required.

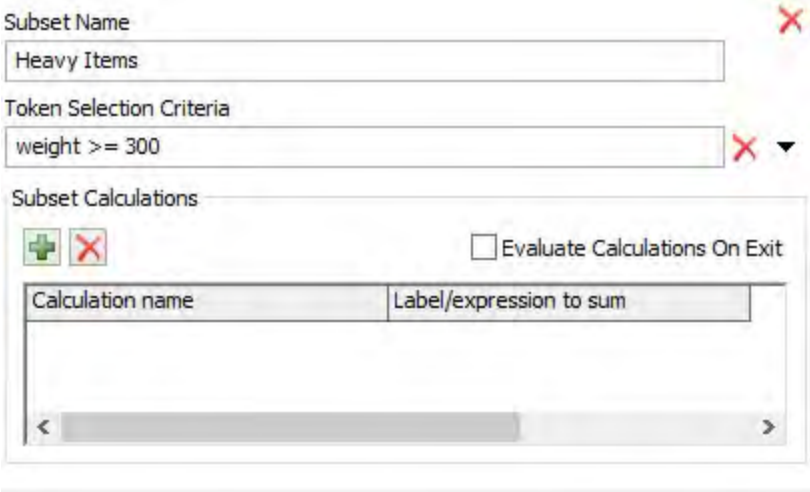
Back to top

Example Subsets

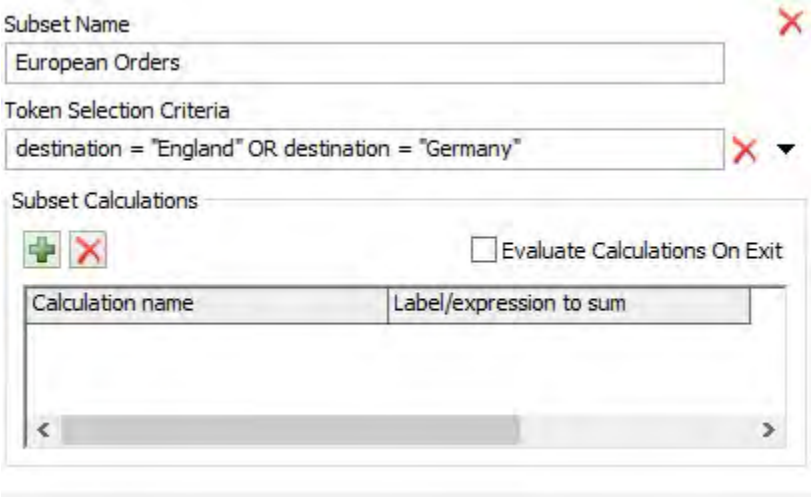
The following are some screenshots of subsets you could add to a Zone. Each image is followed by an explanation about that Subset.



This Subset only contains tokens that have an itemtype label value of 1.



This Subset only contains tokens that have a weight label value greater than 300.



This Subset only contains tokens that have a destination label with text equal to "England" or "Germany".

Back to top

Subset Statistics

Each Subset records the same statistics as the Zone (Input, Output, Content, Staytime, and their Request counterparts), but they are calculated only for the tokens that match the Token Selection Criteria. They are updated at the same time as the Zone statistics.

These statistics are very powerful. You can analyze any area of a Process Flow and answer very complex questions. Using the three examples previous shown, you could answer the following questions:

- How many Itemtype 1 Items are in the Zone?
- How long do Heavy Items stay in the Zone on average?
- How many European Orders are waiting to enter the Zone?

You can create charts and other dashoard widgets that display the values for these statistics.

Back to top

Partition Basics

By default, the Zone does not use Partitions. In order to use Partitions, click the More Properties button in the Zone's Quick Properties panel. This opens the Zone Properties window. On the Partitions tab, you can edit the Partition By field.

If the Partition By field displays "None", than the Zone will not use Partitions.

The Partition By field defines the Partition test. Because this field is a Universal Edit, you can enter in simple expressions like the following:

- token.type - Places tokens in a Partition based on the value of the type label.
- token.name - Places tokens in a Partition based on the name of the token. • duniform(1, 10) - Places tokens in one of 10 random Partitions

Calculations

Like Subsets, Partitions can have Calculations. Unlike Subsets, each individual Partition will have the same set of Calculations as every other Partition. Calculations will be discussed in more detail later. Back to top

Parition Statistics

Each Partition keeps the same statistics as a Subset (Input, Output, Content, Staytime, and their Request counterparts). These statistics are calculated only for the tokens that belong in the Partition. Back to top

Calculation Basics

By default, neither Subsets nor Partitions have any Calculations. In order to add a Calculation to a Subset, click the Add ( ) button in the Subset Calculations panel. This panel found on each Subset panel in the Zone Properties window.

To add a Calculation to a Partition, click the click the Add ( ) button in the Partition Calculations panel. This panel is found on the Partitions tab in the Zone Properties window.

A Calculation is a value that is calculated per token and then totalled for the Subset or Partition. It is also composed of a name and an expression.

The name is arbitrary. For a given Subset, all Calculation names must be unique. For Partition Calculations, all names must be unique. In either case, the name should reflect the meaning of the expression.

The expression can be as simple as a label name, but it can include number values and arithmetic operators. Examples of valid expressions could include:

- weight
- length * width * height
- age / 24

Each Subset can have many Calculations, so there is no reason to have two Subsets with the same Token Selection Criteria. Back to top

Calculation Statistics

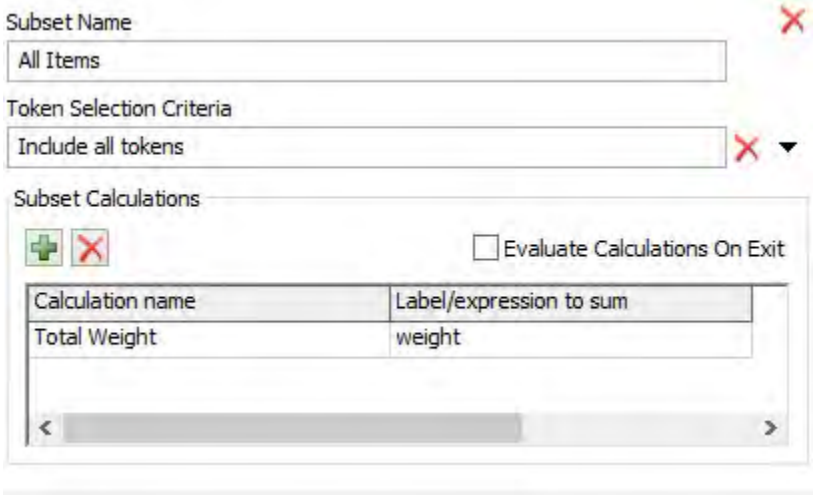
When a token arrives on an Enter Zone activity, the value of each calculation expression is evaluated for that token, and the results are saved. The following statistics are then calculated based on the result:

- Input increases by the result when the token enters the Zone.
- Total increases by the result when the token enter the Zone and decreases by the result when the token exits the Zone.
- Output increases by the result when the token exits the Zone.
- PerToken is set to the result for each token that enters the Zone.
- RequestInput increases by the result when a token arrives on an Enter Zone activity.
- RequestTotal increases by the result when a token arrives on and Enter Zone activity, and decreases by the result when the token leaves the Enter Zone activity.
- RequestOutput increases by the result when a token leaves an Enter Zone activity.

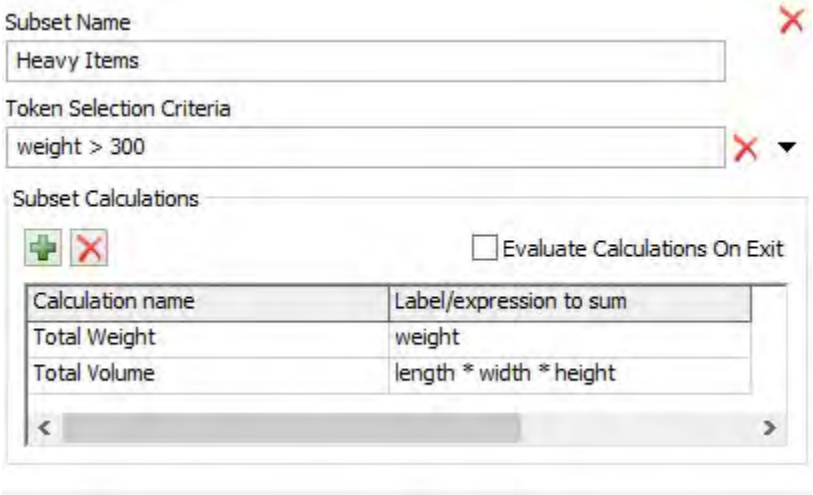
Back to top

Example Subset Calculations

The following screenshots show Subsets with Subset Calculations. Each example followed by a list questions that could be answered by that subset.



- What is the total weight of all the tokens in the Zone?
- What is the average weight per token in the Zone?
- What is the total weight of all tokens waiting to enter the Zone?

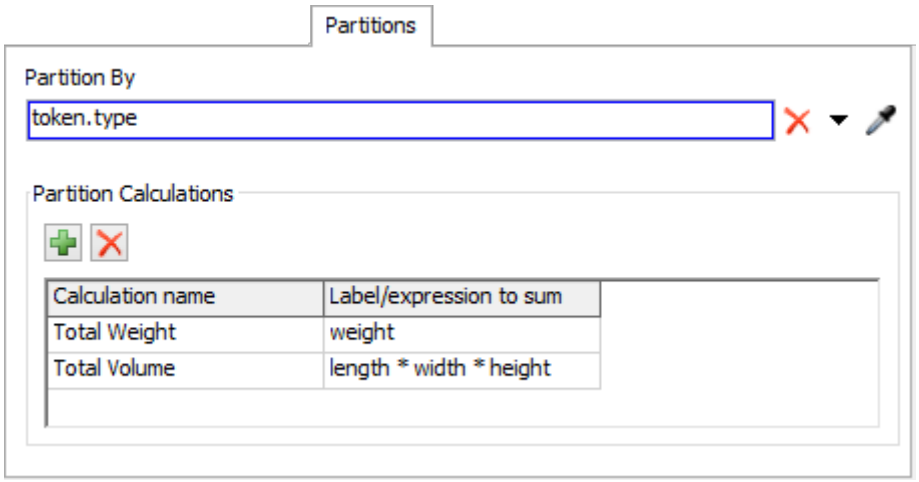


- What is the total weight of all the Heavy Items?
- What is the average staytime of the Heavy Items?
- What is the minimum volume of all the Heavy Items?

Back to top

Example Partition Calculations

Partitions use the same input table as Subsets to specify Calculations. However, if you specify a Partition Calculation, all Partitions will track that calculation. The follwing figure shows two a Zone partitioning by the type label, with two Partition Calculations.



Using this configuration, you could answer the following questions (assume type is a value 1 to 10):

- What is the number of type 1 tokens?
- How much do all the type 3 tokens weigh?
- How long to type 5 tokens stay in the Zone on average?

Back to top

Restricting Access

You can use the Zone to restrict access to a group of activities. If you think of the Zone as an area, then you can limit the size or capacity of that area. If the area is full, tokens wait on the Enter Zone activity until enough tokens exit the Zone. Back to top

Max Content

The easiest way to restrict access to a Zone is by using the Max Content option on the Quick Properties panel for the Zone. When you check this box, it enables the accompanying field. This field is evaluated when a token arrives, or when a token exits the Zone.

The behavior defined by this option is very simple. The Zone guarantees that its content will never exceed the value specified by the Max Content option. Back to top

Constraints

You can also use a Constraint to restrict access to the Zone. By default, Zones do not have any Constraints. There are two kinds of Constraints available in a Zone: Subset Constraints and Partition Constraints. In order to add either kind of Constraint, click the More Properties button to open the Zone Properties window.

Subset Constraints

To add a Subset Constraint, go to the Subset Constraints tab. Click the Add () button. Select a constraint from the menu that appears. If the Add button is disabled, it is because there are no subsets available.

Subset Constraints are always applied to a specific Subset. You cannot change which Subset the Constraint applies to. You must remove the Constraint, and then add a new Constraint for the correct Subset.

Next, you can specify the value that will be constrained using the Calculation dropdown menu. You can always constrain the number of tokens allowed in a subset, or you can constrain any Calculation value. Finally, specify the value for the right hand side of the comparison. This value becomes an upper limit for the Subset.

Constraints can be read in this way:

> The [Calculation] must be less than or equal to [value].

If allowing a token into the Zone would violate any of the constraints (i.e. make the statement untrue), the token is not allowed into the Zone.

For example, if you are calculating the total weight of a Subset, you could restrict the total weight allowed in the Subset. You would create a Constraint for that Subset, set the Calculation to the total weight, and set the Value to something like `1000`. Then, if allowing a token into the Zone would push the total weight over 1000, the token will not be allowed into the Zone.

The Value of all Constraints are evaluated if one or more tokens are about to be tested for entry into the Zone. This can occur when a token arrives at a Zone, or when a token exits a Zone. However, it is not evaluated per token. Therefore, the Value cannot depend on a token.

Partition Constraints

To add a Partition Constraint, go to the Partition Constraints tab. Click the Add () button. This will add a Partition Constraint, which applies to all Partitions.

Next, you can specify the value that will be constrained using the Calculation dropdown menu. You can always constrain the number of tokens allowed in a Partition, or you can constrain any Partition Calculation value.

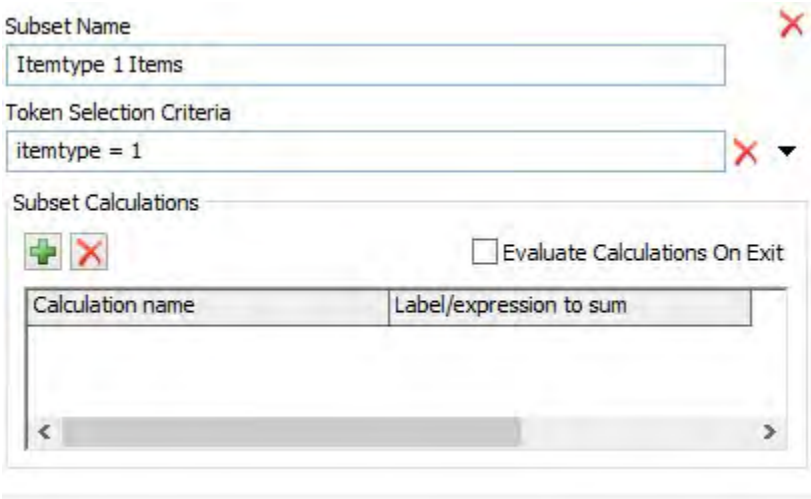
Finally, specify the value for the right hand side of the comparison. This value becomes an upper limit for all Partitions. If some Partitions require a different limit, you can specify those values in the Exceptions table.

Back to top

Subset Constraint Examples

The following screenshots show some example subsets, and constraints that could apply to that subset. Each example also contains a description of the expected behavior for the Zone.

Subset

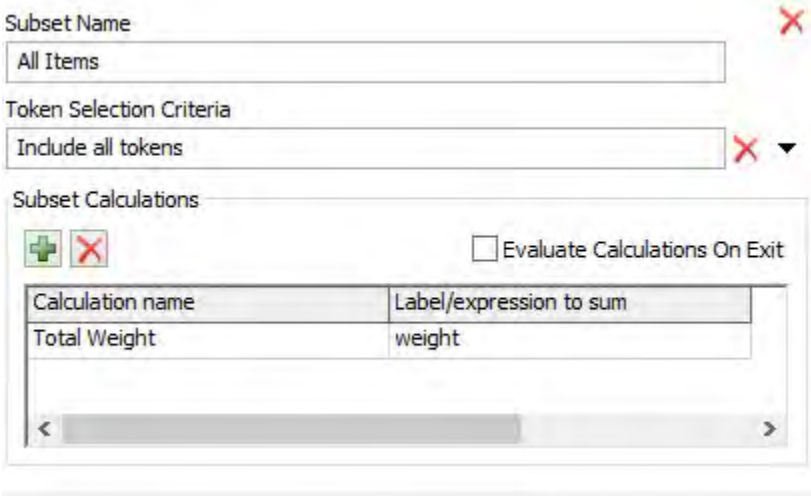


Subset Constraint

Expected Behavior

With this Subset and Subset Constraint, a Zone would only allow 10 tokens with an itemtype label of 1 at any given time.

Subset



Subset Constraints

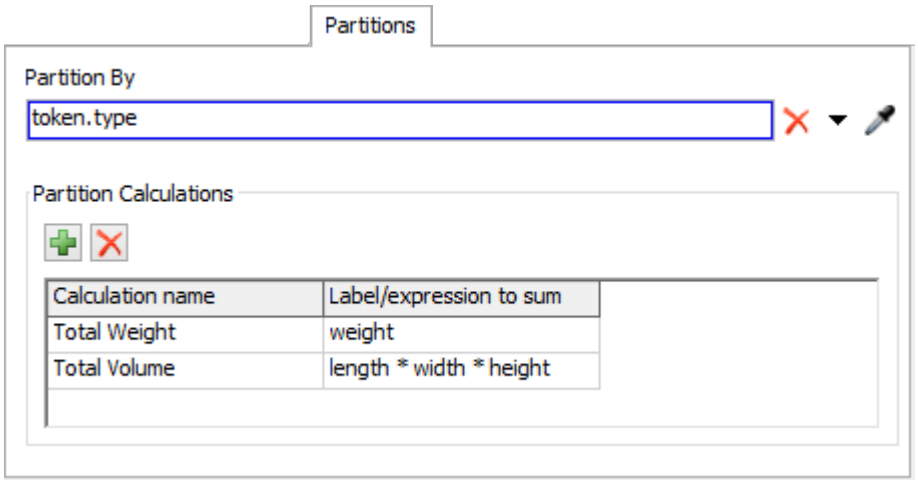
Expected Behavior

Since this Subset includes all tokens, the first Constraint implies that the total weight in the Zone will be no more than 300. The second constraint will ensure that there will be no more than 20 tokens in the Zone. This is the same as applying a Max Content. Normally, you would only constrain the number of tokens of a Subset that has a Token Selection Criteria. Back to top

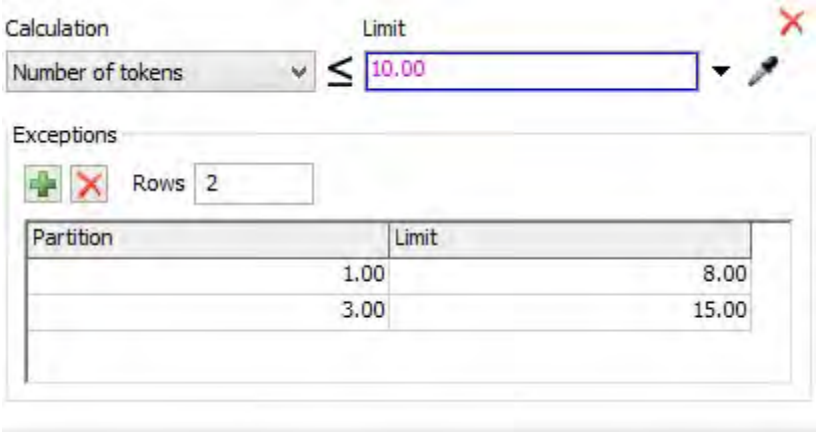
Partition Constraint Example

The following screenshots show an example Partition By expresssion, and constraints that could apply to the resulting Partitions. The example also contains a description of the expected behavior for the Zone.

Partition and Calculations



Partition Constraints



Expected Behavior

The Zone would put all tokens with the same type label together in a corresponding partition. The Zone would calculate the total weight and total volume for each Partition. In addition, the Zone would only allow 10 tokens of any type into the Zone at any given time. However, the Zone would only allow 8 tokens of type 1, and would allow up to 15 tokens of type 3. Back to top

Advanced Topics

The Zone provides some additional control over its behavior. Most of these options are found on the Advanced tab of the Zone Properties window. Proxy Entry and Exit covers options found on the Quick Properties panel for Enter Zone and Exit Zone activities. Non-Member Exit describes the option with the same name on the Quick Properties panel for the Exit Zone activity.

Back to top

Queue Strategy

The Queue Strategy is a SQL statement, with only an ORDER BY component. When a token leaves the Zone, the Zone checks to see if any tokens are waiting to enter the Zone. If there are, the Zone sorts them according to the Queue Strategy. Then it tests each request in that order. You might use this to sort the request tokens by a priority label, or by a weight label, allowing the most important tokens into the Zone first.

If there is no Queue Strategy, the requests are tested the order they arrived. Back to top

Enforcing Queue Order

Normally, the Zone tests all the waiting tokens to see if any of them can enter the Zone. It may be that the second token is small enough to fit, while the first token is not.

If you enforce the queue order (by checking the Enforce Queue Order box), then the Zone will not check any tokens past the first token that cannot enter the Zone. This ensures that tokens enter the Zone in the order they arrived on an Enter Zone, or in the order created by the Queue Strategy. Back to top

Evaluate Calculations On Exit

This option is found in the Subset Calculations area of each Subset's panel on the Subsets tab. If the box is checked, then a token's effect on the calculation is reevaluated. This is only necessary if the token's label values used by the Calculations change while the token is in the Zone. This will keep the Cacluation statistics accurate, but it will invalidate all other statistics kept by the Zone and its Subsets. Back to top

Proxy Entry And Exit

A token can enter for itself, or it can enter for a group of tokens (now deemed a proxy token), such as all of the children of the entering token. This can e defined on the Enter Zone's Enter For property.

When a token attempts to enter a Zone, it does so on behalf of all the tokens it represents. If allowing all the members of that group into the Zone would violate the Max Content or any of the Constraints, the token will not leave the Enter Zone activity, and the members of its group will not become members of the Zone. Otherwise, the token exits the Enter Zone activity and all the tokens it represents become members of the Zone.

The request statistics kept by the Zone are always calculated using the tokens that will become members of the Zone, not by the tokens on the Enter Zone activity. This difference is not important if tokens are entering for themselves.

When the token exits the zone it can exit for itself or as a proxy for other tokens. The Exit Zone activity has an Exit For property that allows you to specify which token(s) the token is exiting the zone for. The Zone's statistics will be updated for all tokens that exit the zone.

If entering or exiting for multiple tokens, these properties should return an array of tokens. This can be created on a label by using the Assign Labels activity.

Back to top

Non-Member Exit

The Exit Zone activity has the Non-Member Exit option on its Quick Properties panel. This option controls the Zone's behavior when the following occurs:

- A token arrives on an Exit Zone activity, and
- Any of the tokens it is exiting for were never added to the Zone.

The default behavior does not allow this to occur; an exception is thrown, appearing in the System Console.

The second option is to ignore these tokens. They do not affect the Zone's statistics, and the simulation continues.

The third option is to update the Zone's statistics. The token is evaluated, so that its affect on any Subset Calculations can be applied as it leaves. You might want to do this if a token enters the Zone with some value

on a label, and that value is split among many tokens that then leave the Zone. The Zone allows you to track the total label value in this case. However, only the Calculation statistics will be valid in this case. Content, Input, Output, and Staytime statistics for the Zone and its subsets will no longer be valid.

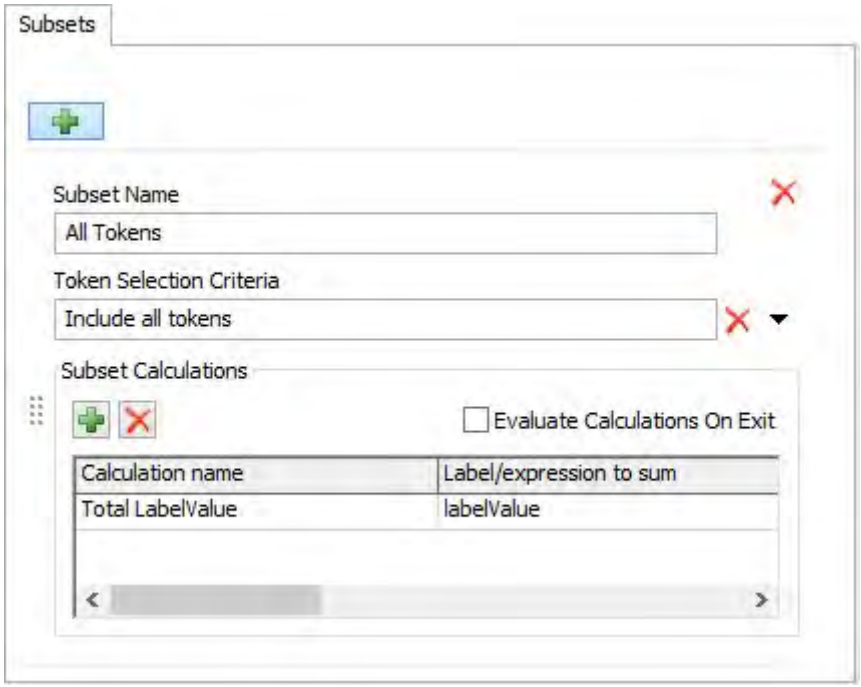
Back to top

Zone Properties Window

The Zone Properties Window can be reached by clicking the More Properties button in the Zone's properties panel. This window allows you to create and edit Subsets, Calculations, and Constraints. It also allows you to manage some of the Zone's advanced features. See Zone Reference for more information.

Subsets

The following image shows the Subsets tab:

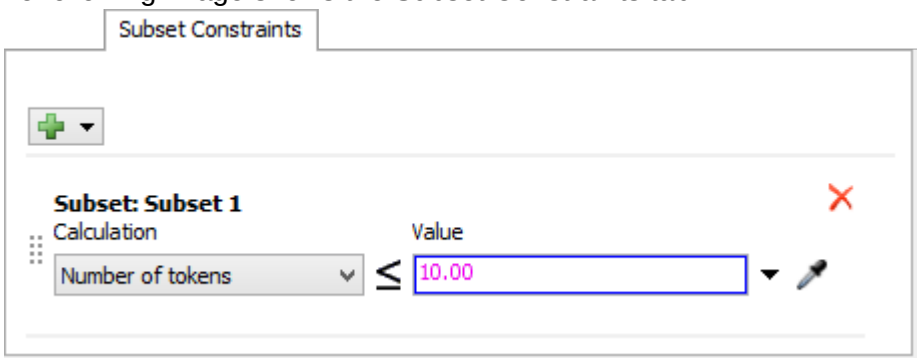


See Zone Reference for more information about using Subsets. Each property is described in the following sections.

- Add Subset The Add Subset button adds a new Subset to the Zone. It is the topmost button on the tab.
- Subset Name The Subset Name box allows you to edit the name of the subset.
- Token Selection Criteria The Token Selection Criteria box allows you to edit the Token Selection Criteria for the Subset. See Zone Reference for more information on possible values for this box.
- Add Calculation The Add Calculation button allows you to add a new calculation to the current Subset. It is found inside the Subset Calculations area.
- Evaluate Calculations On Exit Turns on the Evaluate Calculations On Exit option for the Zone. See Zone Reference for more information on possible values for this box.
- Calculation name You can edit the name of each Calculation by editing the text in this column.
- Label/expression to sum You can edit the expression to sum by edition the text in this column. See Zone Reference for more information on possible expressions.

Subset Constraints

The following image shows the Subset Constraints tab:

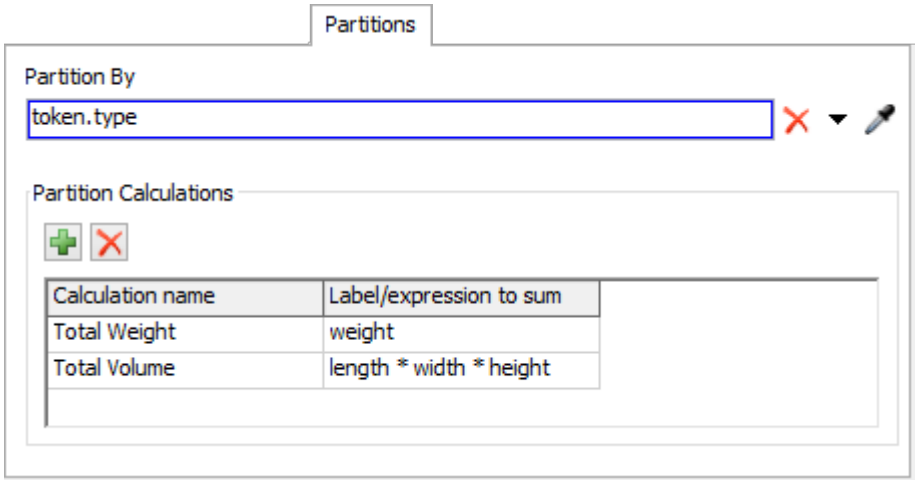


See Zone Reference for more information about using Constraints. Each property is described in the following sections.

- Add Constraint Click the Add Constraint button to add a Constraint to a specific Subset.
- Calculation You can choose which Subset-related value to constrain using the Calculation drop-down list box.
- Value Enter a limit for the value specified in the Calculation drop-down list box.

Partitions

The following image shows the Partitions tab:

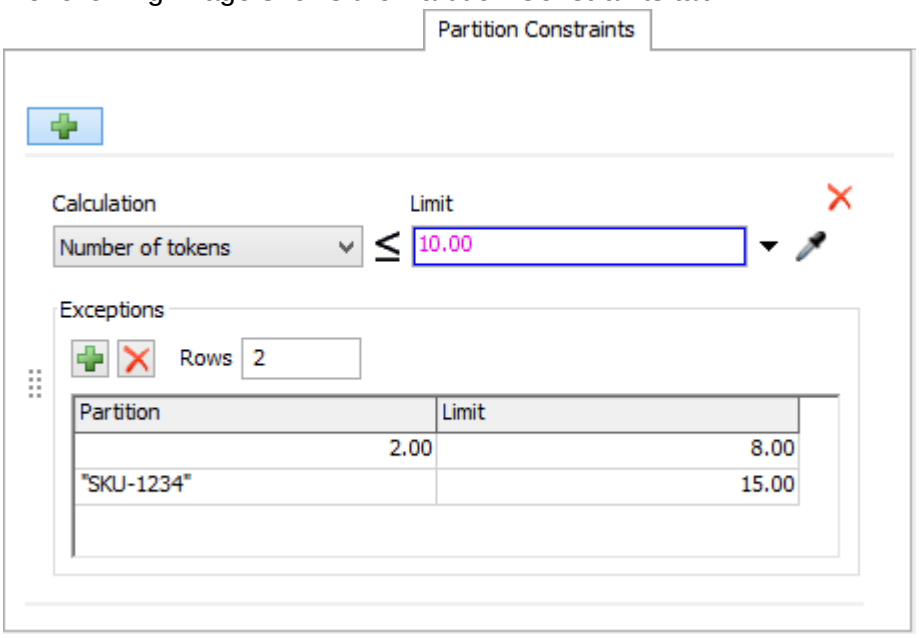


See Zone Reference for more information about using Partitions. Each property is described in the following sections.

- Partition By You can use this field to specify how the Zone should partition its tokens.
- Add Calculation The Add Calculation button allows you to add a new calculation to all Partitions. It is found inside the Partition Calculations area.
- Calculation name You can edit the name of each Calculation by editing the text in this column.
- Label/expression to sum You can edit the expression to sum by edition the text in this column. See Zone Reference for more information on possible expressions.

Partition Constraints

The following image shows the Partition Constraints tab:

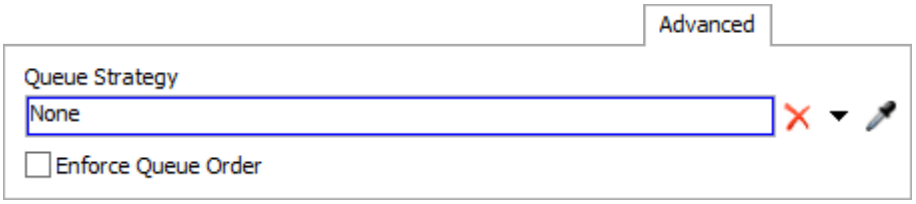


See Zone Reference for more information about using Partition Constraints. Each property is described in the following sections.

- Add Partition Constraint The Add Partition Constraint button adds a new Partition Constraint to the Zone. It is the topmost button on the tab.
- Calculation You can choose which Partition-related value to constrain (for all Partitions) using the Calculation drop-down list box.
- Add Exception You can use the Add Exception button to add an exception to the Partition Constraint.
- Rows You can use this field to specify how many exceptions there are to this Partition Constraint. This is especially useful if you are copying the data from an Excel spreadsheet. Changing this field will add or remove exceptions as necessary.
- Partition This field specifies which particular partition will be affected by the exception. Note that this field accepts text or number data.
- Limit This field the limit for the partition involved in the exception.

Advanced

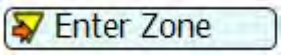
The following image shows the Advanced tab:



See Zone Reference for more information about using each of these properties. Each property is described in the following sections.

- Queue Strategy The Queue Strategy box allows you to edit the expression for the Queue Strategy, which should be an ORDER BY expression.
- Enforce Queue Order Check the Enforce Queue Order box to have the Zone enforce the request queue order.

This activity allows tokens to enter a Zone shared asset. For more information on how the Zone functions see the Zone Reference page. Also see the Enter Zone Behavior page for more advanced functionality.



When this activity is first created, a red exclamation mark shows up to the right of the activity notifying you that a link to a Zone is required for this activity to function. This link may be a direct pointer which can be created by clicking on the exclamation mark and then clicking on a Zone object, or the reference may be dynamic by setting the value for the Zone Reference field.

Connectors

The Enter Zone activity allows multiple connector out. However, tokens automatically released from this activity will be released through the first connector. Only manually released tokens have the opportunity to exit out a different connector. See Adding and Connecting Activities for more information.

Properties

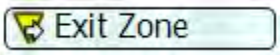
The following image shows properties for the Enter Zone activity:

Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button opens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- Zone Reference The reference to the Zone shared asset that the token is trying to enter.
- Enter For You can control which tokens will enter the Zone when the entering token arrives at this activity using the Enter For box. See Zone Reference for more information.
- Use Max Wait Timer The max wait timer will be evaluated if the token has not entered the zone after the specified time. See Max Wait Timer for more information.

Exit Zone

This activity allows tokens to exit a Zone shared asset. For more information on how the Zone functions see the Zone Reference page. Also see the Exit Zone Behavior page for more advanced functionality.



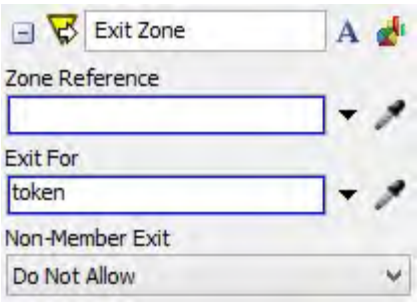
When this activity is first created, a red exclamation mark shows up to the right of the activity notifying you that a link to a Zone is required for this activity to function. This link may be a direct pointer which can be created by clicking on the exclamation mark and then clicking on a Zone object, or the reference may be dynamic by setting the value for the Zone Reference field.

Connectors

The Exit Zone activity only allows one connector out. See Adding and Connecting Activities for more information.

Properties

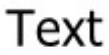
The following image shows properties for the Exit Zone activity:



Each of these properties will be explained in the following sections.

- Name You can change the name of the activity using the Name box. See Name for more information about this property.
- Font The Font button opens a window to edit the activity's background color and font properties. See Font for more information about this property.
- Statistics The Statistics button opens the activity's statistics window. See Statistics for more information about this property.
- Zone Reference The reference to the Zone shared asset that the token is exiting.
- Exit For You can control which tokens will exit the Zone when the entering token arrives at this activity using the Exit For box. See Zone Reference for more information.
- Use Max Wait Timer The max wait timer will be evaluated if the token has not entered the zone after the specified time. See Max Wait Timer for more information.

The Text object can act like a custom text box that can be placed anywhere inside a process flow. The Text object is for display only, meaning that it doesn't have any other function or purpose other than visual display.

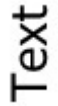


Properties

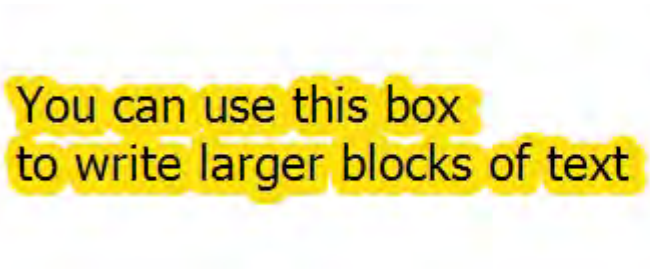
The following image shows properties for Text Display Objects:

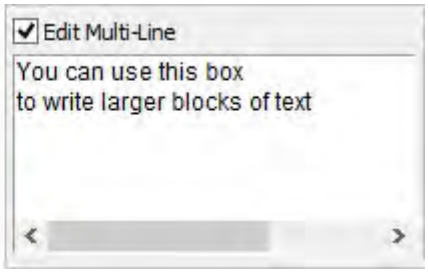
Each of these settings will be explained in the following sections.

- Name You can use the Name box to change the text that will display in the text box. NOTE: Use the Edit Multi-Line checkbox if you intend to create a large block of text. See Edit Multi-Line for more information.
- Lock To You can use the Lock To box to attach the Text object to an activity in your process flow. When the Text is locked to an object, it will move when that activity is moved. Use the Sampler button to select the activity the Text object should be locked with.
- Rotation Using the Rotation box, you can set the angle (in degrees) at which the Text object will display. The base line is horizontal and the rotation is to the right. For example, a rotation value of 90 will result in the following image:



- Edit Multi-Line Check the Edit Multi-Line checkbox if you want to the text box to display a large block of text that might take up multiple lines. When you check the Edit Multi-Line, an edit box will appear where you can enter multiple lines of text, as shown in the following image:





NOTE: You need to enter in a hard return manually if you want to insert a line break.

- Display Name You can use the Display Name to cause the Text Object to display dynamic values during a simulation run. The Text object can be set to display the value of a process flow variable or label. You can possibly use these variables or labels to display relevant data or statistics during a simulation run.

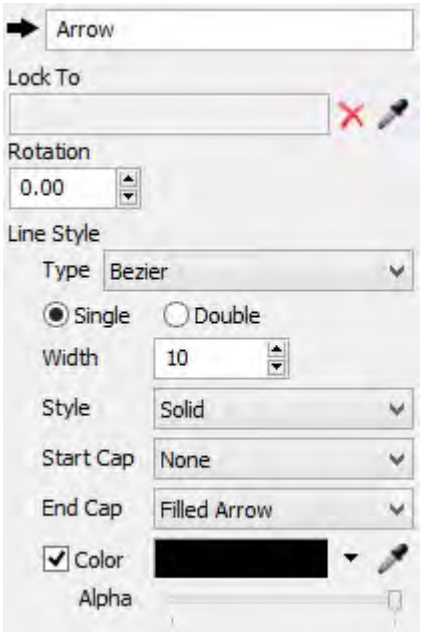
  By default, the Text object will display the name you entered in the Name box. Use the arrow to open a menu and select something else.
- Font There are a variety of font properties you can use to change the text display:
  - The font menu has 13 common fonts available. o You can change the font size. o You can make the text bold, italicized, or underlined.
  - You can change the color of the text by using the color selector or using the Sampler button to select a color from any object in your simulation model or process flow.
- Color AlphaThe Color Alpha slider changes the transparency of the Text object.
- OutlineIf you check the Outline checkbox, the Text object will appear to have a border around it. There are a variety of properties you can use to change the border display:
  - Style - Changes the line style of the border. o Thickness - Changes the thickness of the border.
  - Color Selector - You can change the color of the border by using the color selector or using the Sampler button to select a color from any object in your simulation model or process flow.
  - Color alpha The Color Alpha slider changes the transparency of the border.

The Arrow object is an arrow you can put anywhere in your process flow. The Arrow object is for display only, meaning that it doesn't have any other function or purpose other than visual display.

You can redirect the Arrow object by using the two red control points, and you can resize it by clicking the endpoints.

Properties

The following image shows properties for the Arrow objects:



Each of these settings will be explained in the following sections.

- Name You can change the name of the object using the Name box. See Name for more information about this property.
- Lock To You can use the Lock To box to attach the Arrow object to an activity in your process flow. When the Arrow is locked to an object, it will move when that activity is moved. Use the Sampler button to select the activity the Arrow object should be locked with.
- Rotation The Rotation property actually has no effect on Arrow objects since the rotation is set by the control points. It is only here as a standard property that automatically comes with most display objects.
- Type Use the Type menu to select different arrow styles. In this version of FlexSim, the two available styles are straight and Bezier. Future FlexSim versions might have more styles.
- Single or Double You can use the Single or Double options to switch to from a solid black line or a double line (which looks like it has a white stripe running through it).
- Width Use the Width box to set the width of the line.
- Style Use the Style menu to select the line style (such as a solid, dashed, or dotted line).
- Start Cap Use the Start Cap menu to select the visual style of the start of the line. You can give the start of the line a shape such as a square, a circle, or another arrowhead.
- Color If you check the Color checkbox, it will enable a few properties that can control the arrow's color:
    - You can change the color of the arrow by using the color selector or using the Sampler button to select a color from any object in your simulation model or process flow.
    - The Color Alpha slider changes the transparency of the Arrow object.

You can use the Image object for a number of different purposes:

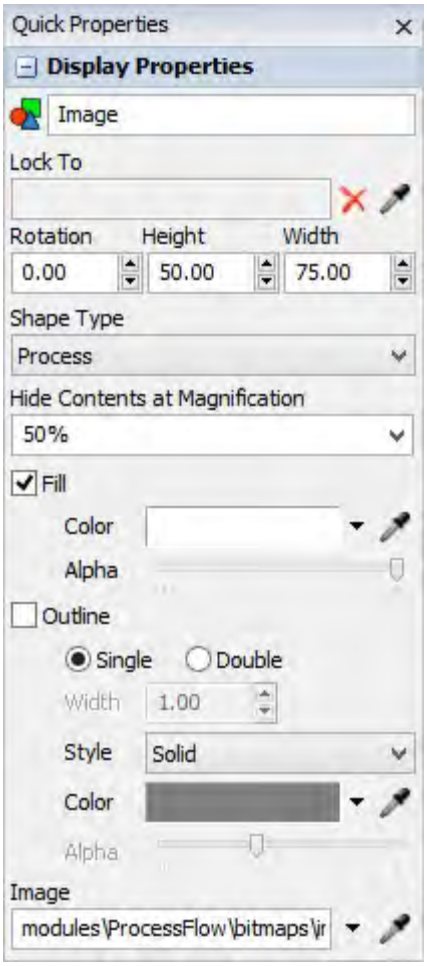
- Adding a custom image somewhere in your process flow.
- Adding a background image to a process flow.
- Adding a background image to Flow Chart object, as shown in the following image:

When you add an image to your process flow, it will initially have a yellow line around it when it is highlighted (as shown in the preceding image). This yellow line will only appear when the image is selected in the process flow. If you want the image to have a border, use the Outline properties. See Outline for more information.

If you want the image to only act as a background image, just clear the Fill checkbox. When this checkbox is cleared, you will only be able to select the image by clicking on its edge. This setting allows you to prevent clicking and selecting the background image on accident. See Fill for more information.

Properties

The following image shows properties for Image Display Objects:



Each of these settings will be explained in the following sections.

- Name You can change the name of the object using the Name box. See Name for more information about this property.
- Lock To You can use the Lock To box to attach the Image object to an activity in your process flow. When the Image is locked to an object, it will move when that activity is moved. Use the Sampler button to select the activity the Image object should be locked with.
- Rotation Using the Rotation box, you can set the angle (in degrees) at which the Image object will display. The base line is horizontal and the rotation is to the right. For example, a rotation value of 90 will result in the following image:

- Height The Height box the Image object's height.

- Width The Width box the Image object's width.
- Shape Type The Shape Type menu allows you to select one of the Flow Chart shapes from a menu. When you select one of the shapes from this menu, it will change the shape of the yellow border that surrounds the image when its selected. This yellow border will disappear when the image is no longer selected in the process flow. However, if you check the Outline checkbox, this shape will be used to create the border for the image. See Outline for more information.
- Hide Contents at Magnification The Hide Contents at Magnification is only relevant if you are using the Image object as Flow Chart object. If you have process flow activities inside the image container, you can use this menu to set these activities to disappear when you zoom out. See Flow Chart - Hide Contents at Magnification for more information.
- Fill When the Fill checkbox is cleared, you will only be able to select the image by clicking on its edge. This setting allows you to prevent clicking and selecting the background image on accident.

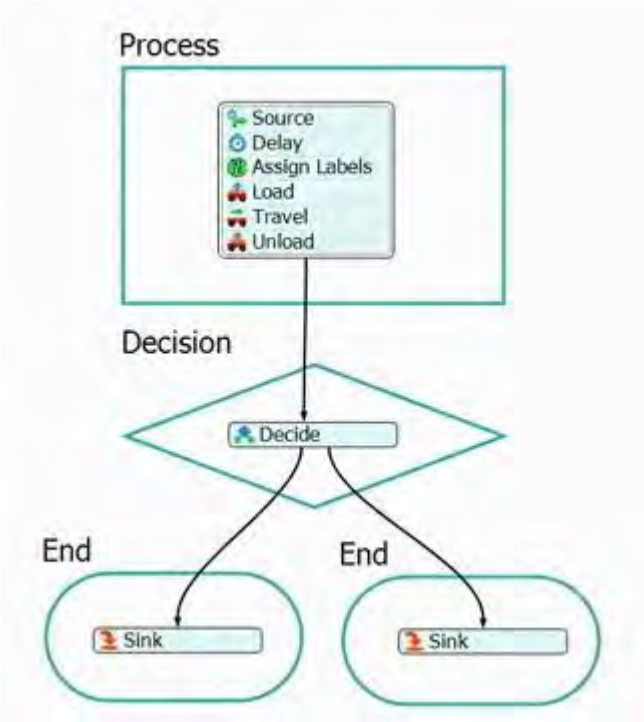
  If you want to be able to click anywhere inside the Image object, check the Fill box. When the box is checked, the following properties will become available:

  - Color - If your image has a transparent background, you can change the color of the background by using the color selector. Alternatively, you can use the Sampler button to select a color from any object in your simulation model or process flow.
  - The Alpha slider changes the transparency of the background color inside the Flow Chart object.
- Outline If you check the Outline checkbox, the Text object will appear to have a border around it. There are a variety of properties you can use to change the border display:
  - Style - Changes the line style of the border. o Thickness - Changes the thickness of the border.
  - Color Selector - You can change the color of the text by using the color selector or using the Sampler button to select a color from any object in your simulation model or process flow.
- Image Use the Image box to choose the image that will display inside the Image box. To browse for an image in your computer, click the arrow next to this box and select Browse from the menu.

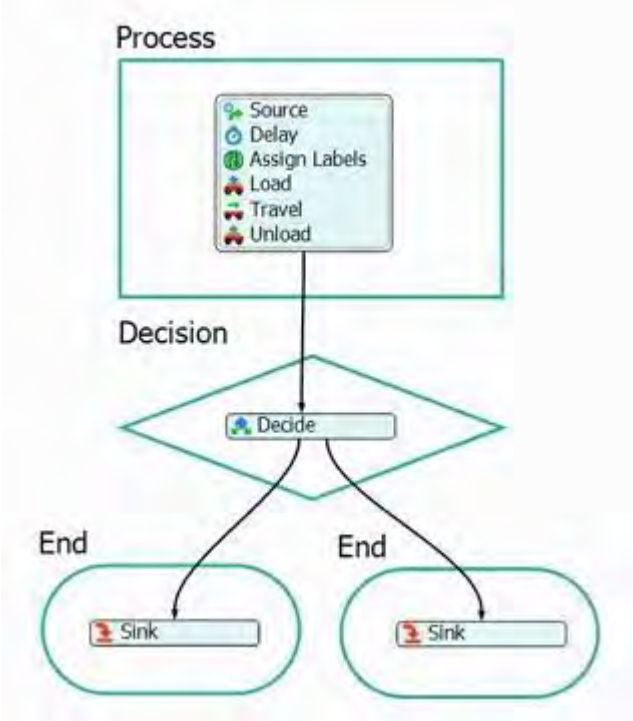
Flow Chart

There are many different Flow Chart objects available in the Process Flow Library. You can use these shapes to build a classic flowchart in your process flow. You could also possibly use them a visual containers that help organize and visualize the activities in your process flow.

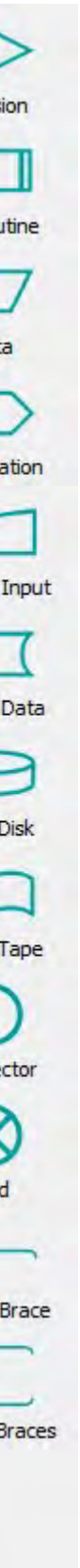
When process flow activities are inside the container, they will move when the container is moved, as shown in the following animated image:



You can also set the Flow Chart object so that the activities possibly disappear when you zoom out from the model. For example, in the following animated image, the Process Flow Chart object is set to hide contents when the process flow is zoomed out by 75%:

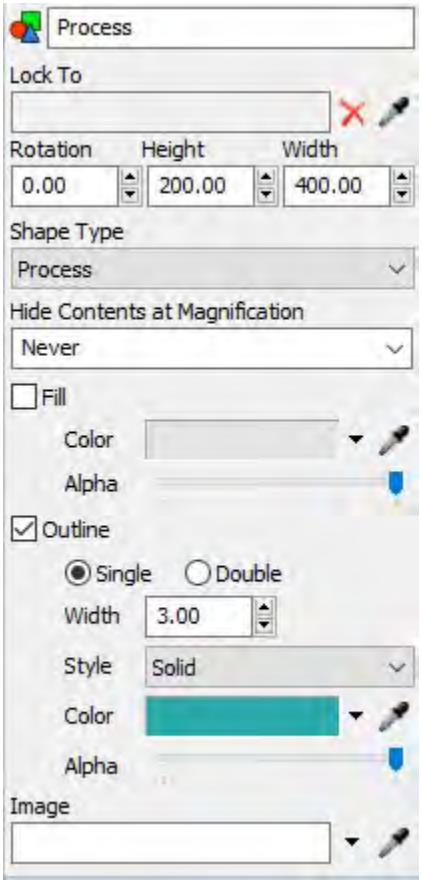


The Process Flow library has many different shapes available, including standard flowchart shapes:



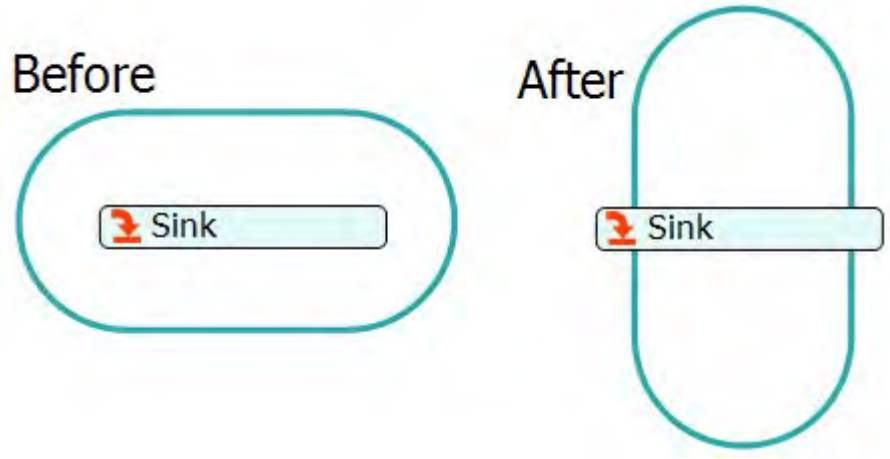
Properties

The following image shows the properties that are available for all Flow Chart objects:



Each of these properties will be explained in the following sections.

- Name You can change the name of the object using the Name box. See Name for more information about this property.
- Lock To You can use the Lock To box to attach the Image object to an activity in your process flow. When the Image is locked to an object, it will move when that activity is moved. Use the Sampler button to select the activity the Image object should be locked with.
- Rotation Using the Rotation box, you can set the angle (in degrees) at which the Flow Chart object will display. The base line is horizontal and the rotation is to the right. For example, a rotation value of 90 will result in the following image, which shows the object before and after rotation:



- Height The Height box the Flow Chart object's height.
- Width The Width box the Flow Chart object's width.
- Shape TypeYou can switch between the different kinds of flowcharting styles here. This will change the shape of the container accordingly.
- Hide Contents at MagnificationYou can specify a zoom level where when you zoom out further than that level the contents of this container will be hidden.
- Fill When the Fill checkbox is cleared, you will only be able to select the image by clicking on its edge. This setting allows you to prevent clicking and selecting the background image on accident.

  If you want to be able to click anywhere inside the Image object, check the Fill box. When the box is checked, the following properties will become available:

  - Color - Change the color of the Flow chart object's background by using the color selector. Alternatively, you can use the Sampler button to select a color from any object in your simulation model or process flow.
  - The Alpha slider changes the transparency of the Image object.
- Outline If you check the Outline checkbox, the border of the Flow Chart object will be visible. There are a variety of properties you can use to change the border display:
  - Single or Double - Use these options to switch to from a solid black line or a double line (which looks like it has a white stripe running through it). o Width - Sets the border's width. o Style - Changes the line style of the border.
  - Thickness - Changes the thickness of the border.
  - Color - You can change the color of the border by using the color selector or using the Sampler button to select a color from any object in your simulation model or process flow.
  - The Alpha slider changes the transparency of the border.
- Image If you want to use a background image for your Flow Chart shape, use the Image box to choose the image. To browse for an image in your computer, click the arrow next to this box and select Browse from the menu.

About Conveyors

The FlexSim Library contains a new set of objects for simulating conveyor systems. Although previous versions of FlexSim had some simple conveyor objects, these new objects are more advanced, featurerich, and user-friendly.

Here are a few of the useful features of the new conveyor system:

- Slug Building and Merge Control - You can now enable slug building on conveyors. This will accumulate a slug of items on a conveyor and then release it once the slug is ready. Using this feature in conjunction with the Merge Controller object, you can easily implement saw tooth merges without writing any custom code.

- Range-Based Transfer Points - Operators can now pick up or drop off items from a range of possible transfer points along the conveyor rather than a single fixed point. This enables easy simulation of picking operations. Also, when picking from a conveyor, operators will automatically predict a proper pick up point on the conveyor based on item speed, operator speed, and distance.

- Improved Photo Eyes - You can now build more sophisticated logic into Photo Eyes. Photo Eyes can be programmed to trigger certain events or behaviors if they are blocked or if they are clear for a specific amount of time. You can also make more precise adjustments to the height and angle of the photo eye on the conveyor.

- Decision Points - You can place Decision Point objects on a conveyor to act as a sensor for control logic, or a communication point for other objects, such as the Merge Controller. Decision Points can be used for building complex logic into a conveyor system. Decision Points are also easier to manipulate and place on a conveyor.

- Improved Movement Controls - FlexSim now has easy controls for simulating item movement along conveyors. For example, there are many possible options for controlling how items transfer between the conveyors, including speed, delay time, and pop-up distance. You can also adjust a roller skew angle on a conveyor, so that items will accumulate on one side of the conveyor if needed. You can also tilt, translate, and rotate items while moving along the conveyor.

- Cross-conveyor accumulation - FlexSim now simulates better accumulation across multiple conveyor sections.

- Item Orientation - You can define any orientation for items on a conveyor. This orientation will automatically persist across multiple sections, and will update its calculations to match the object's orientation when the item transfers through side transfers.

- Power and Free - With the new conveyor system you can simulate power and free systems. Just check a box to enable power and free and then define the chain's dog gap.

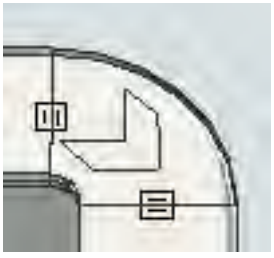
The subsequent chapters of the User Manual will explain in detail how to incorporate these new advanced features into your simulation models.

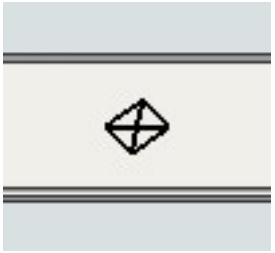
Introduction to Conveyor Objects

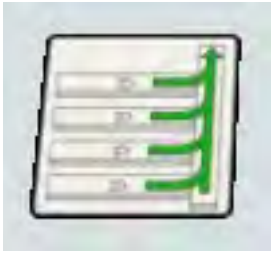
Conveyor objects are now a separate, distinct category in the FlexSim Library. Glancing at the Library under the Conveyors group, you'll see all of the new objects.

The following table will provide a high-level overview of these objects. The categories and objects are presented in the same order as they appear in the Library.

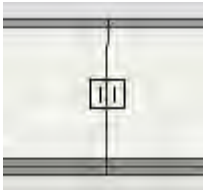
Name and Icon	Description	Appearance

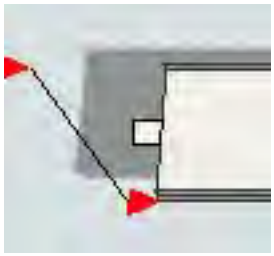
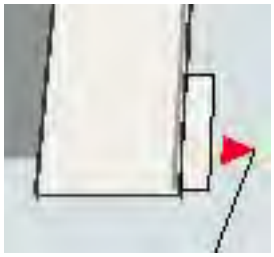
Straight Conveyor 	The Straight Conveyor can simulate conveyor belts or roller conveyors. As its name suggests, this conveyor has a straight shape.	
Curved Conveyor 	The Curved Conveyor is another object that can simulate conveyor belts or roller conveyors. Unlike the Straight Conveyor, this conveyor has a curved shape with varying radius settings.	
Join Conveyors 	Join Conveyors acts more like a tool than an object. Use it to create a curved conveyor connecting two conveyor sections.	

Decision Points 	You can use a Decision Point to build logic into your conveyor system. Decision Points can also be linked to a Merge Controller to notify it when a released slug has cleared designated points in the system.	
Stations 	A Station is an object you can use to add processing points into the conveyor system. The Station works very much like a standard Processor object, except that it is placed as a point in the conveyor system, instead of as an object outside of the conveyor system.	
Photo Eye 	Photo Eyes are similar to Decision Points in that they act as a sensor on the conveyor. However, whereas a Decision Point will fire on every passing item, Photo Eyes function like real-life photo eyes. They will only clear if there is a gap between items on the conveyor. You can also define a block and/or clear time so the respective trigger will only fire if the Photo Eye is continuously blocked/cleared for a defined duration of time.	

Motor 	Motors can be used to control whether the conveyor systems are on or off at a given time. The Motor can also be used to sync dog gaps on a power and free chain loop when simulating a Power and Free conveyor system.	
Merge Controller 	The Merge Controller is an object that can control how different conveyor lanes merge together. Use the Merge Controller to define a lane release strategy for the merge.	

In addition to these objects which are available in the FlexSim Library, there are three additional conveyor objects you should be aware of: Transfers, Entry Transfers, and Exit Transfers. These objects are not available in the FlexSim library but are automatically created whenever you connect conveyors to other objects:

Name	Description	Appearance
Transfers	Transfers are connections from one conveyor to another. They can affect how an item is transferred between conveyors.	

Entry Transfers	Entry Transfers are connections from a non-conveyor object (such as a Source or other Fixed Resource) to a conveyor object. They can affect how an item is transferred to a conveyor.	
Exit Transfers	Exit Transfers are connections from a conveyor to a non-conveyor object (such as a Sink or other Fixed Resource) to a conveyor object. Exit Transfers have the ability to send for a transport, or in other words, a Task Executer to pick up items and deliver them to another object. Task Executers can pick up items from a particular point on the Exit Transfer or a range of possible points along an Exit Transfer.	

The subsequent sections of the chapter will explain how to use these objects to create complex conveyor systems in your simulation model.

Adding Conveyors to a Simulation Model

Fortunately, many of the new conveyor objects can be added to the model the same way you would add any other object from the FlexSim Library: you simply click on the object in the Library and drag it into your simulation.

However, the Straight Conveyor, the Curved Conveyor, and the Join Conveyors each behave a little differently when added to a model. For instance, most of the other objects in the FlexSim Library have a default length, height, and width when you first add them to a simulation model. The Straight and Curved Conveyor objects will have a default size if you click and drag them into the model. However, you also have the option to define the length and/or radius of the conveyors when you first insert them in your model, which will be discussed in the following sections.

The Join Conveyors Object

The unique properties of the Join Conveyors object will be discussed in more detail in the section about Connecting Conveyors.

Adding Conveyor Objects with Default Dimensions

If you want to add a Straight or Curved Conveyor that has the default size, height, and width, you can use the same method for adding an object that you would normally use to add any other FlexSim object from the Library: simply click on the object and drop it into the model.

Adding Conveyor Objects with Custom Dimensions

The process for adding a conveyor object that has a custom length and/or radius is slightly different, but still relatively simple. When you create a custom conveyor, you could think of it as "building" these objects when you first place them in your model. For that reason, this guide generally will refer to the process of adding either a Straight or Curved Conveyor as entering *conveyor building mode*. To add a custom Straight or Curved Conveyor to a simulation model:

1. In the Library under the Conveyors group, click either the Straight Conveyor or the Curved Conveyor object. You will then enter into *conveyor building mode*. When you are in conveyor building mode, your mouse pointer will change to a plus sign with either a Straight Conveyor icon or a Curved Conveyor icon next to it, as shown in the following image:

2. Once in building mode, find the position in your simulation model where you want to place the tail of the conveyor. When you click on that position in the model and start moving the mouse pointer in a different direction, you'll notice that it begins creating a conveyor object.

3. Reposition the mouse pointer until the conveyor head is at the approximate length, angle, and radius you want it to be relative to the tail. Click the mouse again to finish building the conveyor.

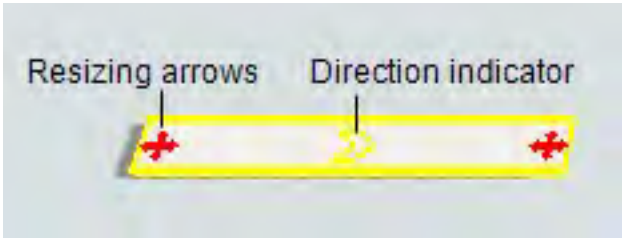
Exiting Building Mode
After you've created your first conveyor, you might notice that you are still in building mode. You can continue to create more conveyors while you are in building mode or you can exit building mode by pressing the Esc key or right-clicking your mouse. You'll know you have exited building mode because your mouse pointer will look like a normal cursor again.

Moving, Resizing, and Reversing Conveyors

Most objects in the FlexSim Library can be moved, rotated, or resized using either the Quick Properties pane or your mouse. You can use these two methods to change a conveyor's properties as well. However, the process is slightly different than the process for other FlexSim objects:

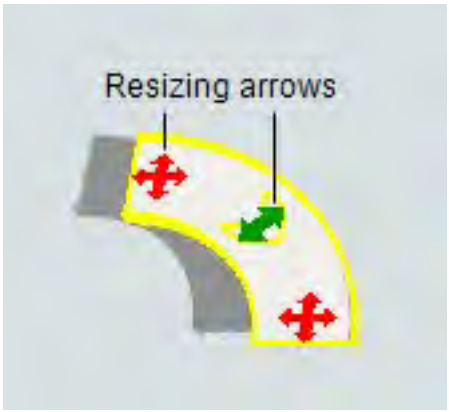
- You can move or resize a conveyor as a single unit, but you can also move or resize the head and tail ends of a conveyor as separate units.
- You can reverse the direction items travel as they move along the conveyor.
- Conveyors cannot be rotated on their X, Y, or Z axes, but you can change the height (Z axis) of either the head or tail ends of the conveyors.
- In addition to the usual X, Y, and Z axes, the Curved Conveyor has controls that allow you to edit its radius, start angle, and sweep angle.

- The editing controls also look and function differently on a conveyor. When a conveyor is highlighted, it will have a set of resizing arrows on both its head and tail end, as shown in the images below. The Curved Conveyor also has an additional green resizing arrow for adjusting its radius, start angle, and sweep angle. Each conveyor also has a direction indicator, which is a large unfilled arrow that shows you the direction the conveyor belt will move while operating.



Example of a highlighted Straight Conveyor

Example of a



highlighted Curved Conveyor

The following sections will discuss how to move, resize, and reverse a conveyor using either the Quick Properties pane or the mouse. Using the mouse to change a conveyor's properties is a little easier, but the Quick Properties pane will allow you to make the conveyor's properties more precise.

Use Conveyor Types to Make Duplicate Conveyors

If you know that you're going to build many similar conveyors with similar properties and behaviors, you should create a specific Conveyor Type for your conveyor. That way when you build any other conveyors, you can assign it to that particular Conveyor Type and it will immediately import the pre-defined settings you saved. See Conveyor Types for more information.

Using the Mouse to Move or Resize a Conveyor

In order to use your mouse to move or resize a conveyor, you must first click once on the conveyor to highlight it. After you've taken that step, use the steps listed in following table, which explains the process for using the mouse to change the conveyor's various properties:

Task	Process

To move the entire conveyor	Click anywhere on the conveyor (except on the resizing arrows) and drag the conveyor to the desired position.
To move either the head or tail of the conveyor	Click on the resizing arrow on either the head or tail of the conveyor. Drag the head or tail to the desired position.
To change the height (Z-axis) of the entire conveyor	Click anywhere on the conveyor (except on the resizing arrows) and scroll the mouse wheel up or down until the conveyor is at its desired height.
To change the height (Z-axis) of either the head or tail of the conveyor	Click on the resizing arrow on either the head or tail of the conveyor. Scroll the mouse wheel up or down until the head or tail is at its desired height.
To change the length of the conveyor	Click on the resizing arrow on either the head or tail of the conveyor. Drag the head or tail to the desired length.

Additional Resizing Options for Curved Conveyors

Everything in the preceding table applies to Curved Conveyors, but you can also use the green resizing arrow to change its length. See Changing the Radius and Angle of Curved Conveyors for more information.

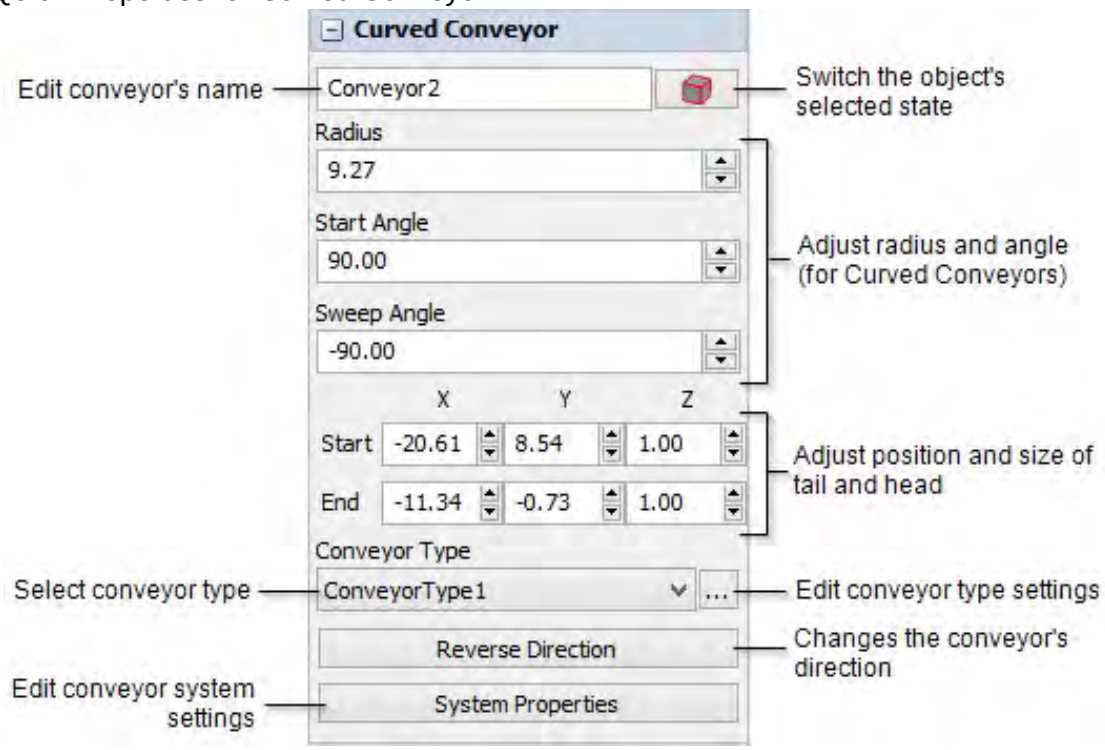
Using Quick Properties to Move or Resize a Conveyor

You might possibly want the location, rotation, and size of the conveyor to be more precise in your model. In that case, it's generally a good practice to use your mouse to move or resize your conveyor until it is in the approximate position or size you want it to be. Then you can use the Quick Properties pane to make it more precise.

When you highlight a conveyor by clicking on it, the right pane displays the Quick Properties for that specific conveyor. The controls that appear in Quick Properties will vary depending on whether you clicked a

Straight or Curved Conveyor. Both kinds of conveyors have boxes that adjust the position and size of the head and tail of the conveyor, as illustrated in the following image. However, Curved Conveyors have additional boxes that adjust the radius, start angle, and sweep angle. The following image shows the Quick Properties for a Curved Conveyor:

Quick Properties for Curved Conveyor



Start adjusts the X, Y, or Z position of the conveyor's tail. End adjusts the conveyor's head.

A conveyor's length is based on the position of the head relative to the tail. For example, if a conveyor is parallel with the X-axis on the grid with its tail at the 0.0 position and the head is in the 10.0 position, the conveyor will be 10 units long. The same is true for the other axes. Experiment with some of the Quick Properties to get your conveyor exactly the way it should be.

Use Conveyor Types to Adjust Settings

The width of conveyors can be edited in the Conveyor Type properties. You can also adjust many other conveyor properties and quickly import conveyor settings that you use frequently. See Conveyor Types for more information.

Reversing the Direction of a Conveyor

To change the direction that items travel on a conveyor:

1. Click on the conveyor to highlight it.
2. In the Quick Properties pane, under the Straight Conveyor or Curved Conveyor group, click the Reverse Direction button, as shown in the preceding image. The direction indicator on the conveyor will now point in the opposite direction.

Repeat this process if you want to change the conveyor's direction back to its original setting.

Changing the Radius and Angle of Curved Conveyors

Curved Conveyors have three additional properties:

- Radius - The radius of the conveyor relative to the midpoint of the hypothetical circle around which the conveyor is drawn. Changing the radius will affect the length of the conveyor because it makes this hypothetical circle larger.
- Start Angle - The angle of the tail end of the conveyor relative to the simulation model grid. For example, if the start angle is set to 90, the edge of the conveyor's tail will be perpendicular to the simulation grid.
- Sweep Angle - The angle of the head of the conveyor relative to the start angle. For example, if the start angle is set to 45 and the sweep angle is set to 90, the edge of the conveyor's head will appear to be at a 45 degree angle relative to the simulation grid.

You can use either the mouse or Quick Properties to change the radius, start angle, and sweep angle of Curved Conveyors. It's generally a good practice to use your mouse to move or resize your conveyor until it is in the approximate position or size you want it to be. Then you can use the Quick Properties pane to make it more precise.

To adjust the radius, start angle, or sweep angle using Quick Properties, go to the Quick Properties Pane and look under the Curved Conveyor group. Then type or select a value in the Radius, Start Angle, or Sweep Angle boxes.

In order to use your mouse to move or resize a conveyor, you must first click once on the conveyor to highlight it. After you've taken that step, use the steps listed in following table, which explains the process for using the mouse to change the conveyor's properties:

Task	Process
To change the radius	Click the green resizing arrow in the middle of the Curved Conveyor and drag the mouse to the desired radius.
To change the start angle	Click the red resizing arrow on the tail end of the Curved Conveyor and drag the mouse until the edge of the tail is at the desired angle.

To change the sweep angle	Click the red resizing arrow on the head of the Curved Conveyor and drag the mouse until the edge of the head is at the desired angle.

Connecting Conveyors

When you create a connection between a conveyor and another object, you make it possible for these objects to interact together. The most common way that objects interact in FlexSim is to transfer items from one object to another or send messages to another object.

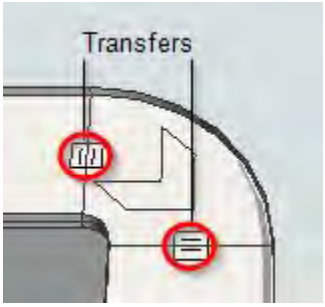
The process for connecting conveyors is relatively similar to the process you would use to connect any other object from the FlexSim Library. However, there are a few differences to be aware of. The following sections will discuss some of the different methods for connecting conveyor objects.

Connecting Two Conveyors

One of the easiest ways to connect two conveyors together in a simulation model is to simply move two conveyors close enough until they snap together. To snap two conveyors together:

1. Click on one conveyor and drag it close to the edge of the other conveyor. When the edges of the two conveyors get close enough, they will snap together.

2. After you release the mouse button, a Transfer will appear that connects the two conveyors together. The Transfer will look like a small square box that overlaps the edges of the two conveyors.



You can customize how the two conveyors transfer objects together by changing the settings for the Transfer. For more information, see Transfer Types.

Using the Join Conveyors Object

If you want to create a Curved Conveyor connecting two conveyors, you can use the Join Conveyors object from the FlexSim Library to connect two conveyors together. To connect conveyors using the Join Conveyors object:

1. In the FlexSim Library under the Conveyors group, click on the Join Conveyors object. You will then enter into *conveyor joining mode*. When you are in conveyor joining mode, your mouse pointer will change to a plus sign with a Join Conveyors icon next to it, as shown in the following image:

2. Once in joining mode, click on the center of the first conveyor you'd like to join. After clicking on the conveyor, you'll notice that a yellow line will follow your mouse as it moves, similar to the line that appears when you are connecting two FlexSim objects.
3. Click on the second conveyor you'd like to join. You'll then notice a new Curved Conveyor will appear between the two conveyors. Two Transfer Points will also appear on both ends of the new conveyor.

Connecting Conveyors to Other FlexSim Objects

You can connect conveyors to other FlexSim objects such as Fixed Resources the same way you would normally connect objects in FlexSim. To connect objects, you can use Connect Objects  on the toolbar or you can use the associated keyboard shortcuts. See Connecting Objects for more information. However, unlike connecting normal FlexSim objects to each other, when you make an input/output connection (an 'A' connect) a conveyor to another FlexSim object, it will create a new an Entry Transfer or Exit Transfer, an in-between object that is associated with a specific point or range on the conveyor. An Exit Transfer is created when you connect a conveyor to another FlexSim object. An Entry Transfer is created when you connect a FlexSim object to a conveyor.

Connecting Motors and Merge Controllers

The Motor and Merge Controller objects will usually only be connected to either a Straight or Curved Conveyor. When connecting the Motor and Merge Controller to a conveyor, you should use an input/output port connection (an 'A' connect). NOTE: This will not act like an input/output port but will merely serve as a reference point between the objects. Use Connect Objects  on the toolbar or the associated keyboard shortcuts to create this kind of connection. See Connecting Objects for more information.

You might possibly want to make a port connection between the Motor and Merge Controller objects and a non-conveyor object such as a Task Executer. In that case, you should create a center port connection (an 'S' connect). A center port connection will allow the Motor or Merge Controller to send messages to other non-conveyor objects. Click the Connect Objects arrow to open a submenu and select Connect Center Ports to create a center port connection.

Flow Control

FlexSim's conveyor objects can simulate a wide range of conveyor technologies that can manage the flow of products or materials throughout a conveyor system. This chapter will introduce different methods for simulating common conveyor tasks such as merging, sorting, gapping, and Power and Free conveyor technologies.

FlexSim simulates these kinds of complex systems using built-in Behaviors (also known as Pick Lists) on Decision Points or Photo Eyes. Some of the Decision Point or Photo Eye Behaviors are:

- Send Item - Can send items to different destinations (such as different conveyor lines) based on criteria you specify
- Stop/Resume - Can stop an item, delay an item, and can set conditions that can make a conveyor motor stop or restart
- Area Restriction - Can create an area that only allows a fixed number of items to flow through a restricted area on the conveyor at a time
- Movement - Can rotate, tilt, or move items at a particular point on a conveyor
- Set Conveyor Speed - Can change the speed of the conveyor based on certain conditions

See the chapter on Decision Point and Photo Eye Pick List Behaviors for a more in-depth discussion of each Behavior and its available settings. The other sections in this chapter will discuss how you can use some of these Behaviors in combination with other conveyor objects to simulate common conveyor functions.

Sorting

In conveyor systems, sorting (sortation) is needed when high quantities of items need to be transported to different destinations based on varying criteria. Sorters can divert items from incoming conveyor lines to shipping lanes, palletizing operations, packing stations, or other sorting stations.

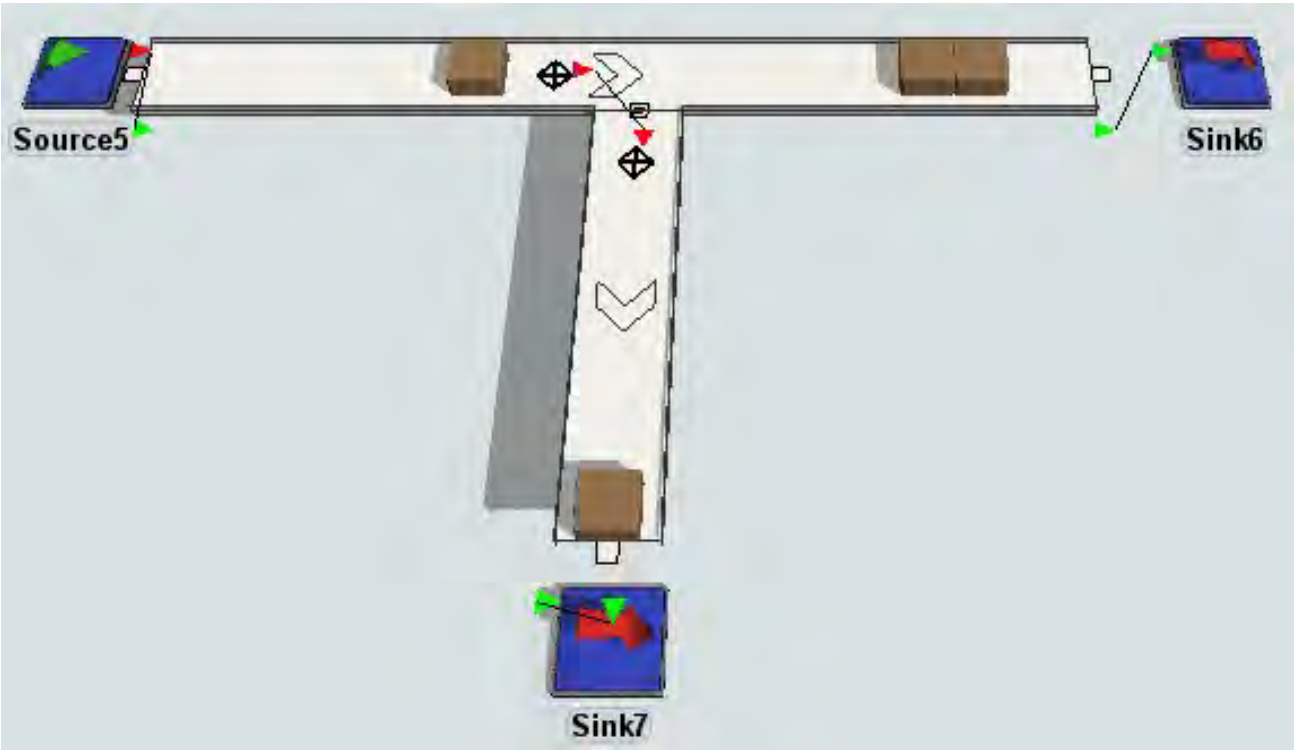
To sort items on a conveyor in FlexSim, you'll need to use a Decision Point or Photo Eye with the Send Item Behavior to sort items to various destinations. The Send Item Behavior uses two fields to determine how it should sort items:

- The Condition field - This field decides "Should I divert this item?" and sorts items based on particular conditions. (See The Condition field for more information.)
- The Destinations field - This field decides "Which destination should I send the item to?" and sorts items to multiple possible destinations based on criteria you define.

You can use a combination of different settings in these two fields to build complex sorting logic into your conveyors. In the following sections, we'll demonstrate two different methods for sorting these items, each one focusing on sorting using a different field.

Sorting Using the Condition Field

To sort items using the Condition field, we will add a Decision Point to the main line conveyor and connect it to a single Decision Point on the diverting conveyor. Then we will add the Send Item Behavior on the OnArrival Trigger to the Decision Point on the main line. The Condition field on the Send Item Behavior will be set to randomly sort 30% of items to the diverting conveyor, as shown in the following image:



To create this kind of sortation system:

1. Add the conveyors and add any additional Fixed Resources such as Sources and Sinks. (See Adding Conveyors to a Simulation Model.)

2. Connect the conveyors together and connect to the conveyors to the Fixed Resources. (See Connecting Conveyors.)

3. Place a Decision Point at the first junction on the conveyor where items could be diverted to different conveyor lines.

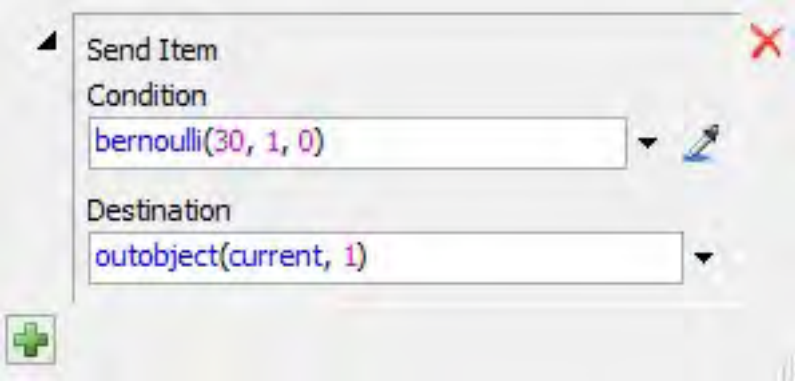
4. Place a Decision Point at the beginning of the conveyor line where 30% of the products will be diverted.

5. Make an input/output port connection ('A' connect) from the sending Decision Point to the receiving Decision Point.

6. Double-click the sending Decision Point to open its Properties dialog box.

7. In the Decision Point tab, click the Plus sign next to the OnArrival Trigger and select Send Item from the menu. (See Send Item for more in-depth information about this Behavior.)

8. A dialog box will pop up. In the Condition field, enter in the following expression: bernoulli(30, 1, 0). This bernoulli distribution will randomly return a value of 1 (true) 30 percent of the time, and 0 (false) 70 percent of the time.



9. From the Destination menu, enter outobject(current, 1). This means when the condition is true (30% of the time), the item will be sent to the Decision Point connected to the first output port of the main line Decision Point, i.e. the divert conveyor's Decision Point.

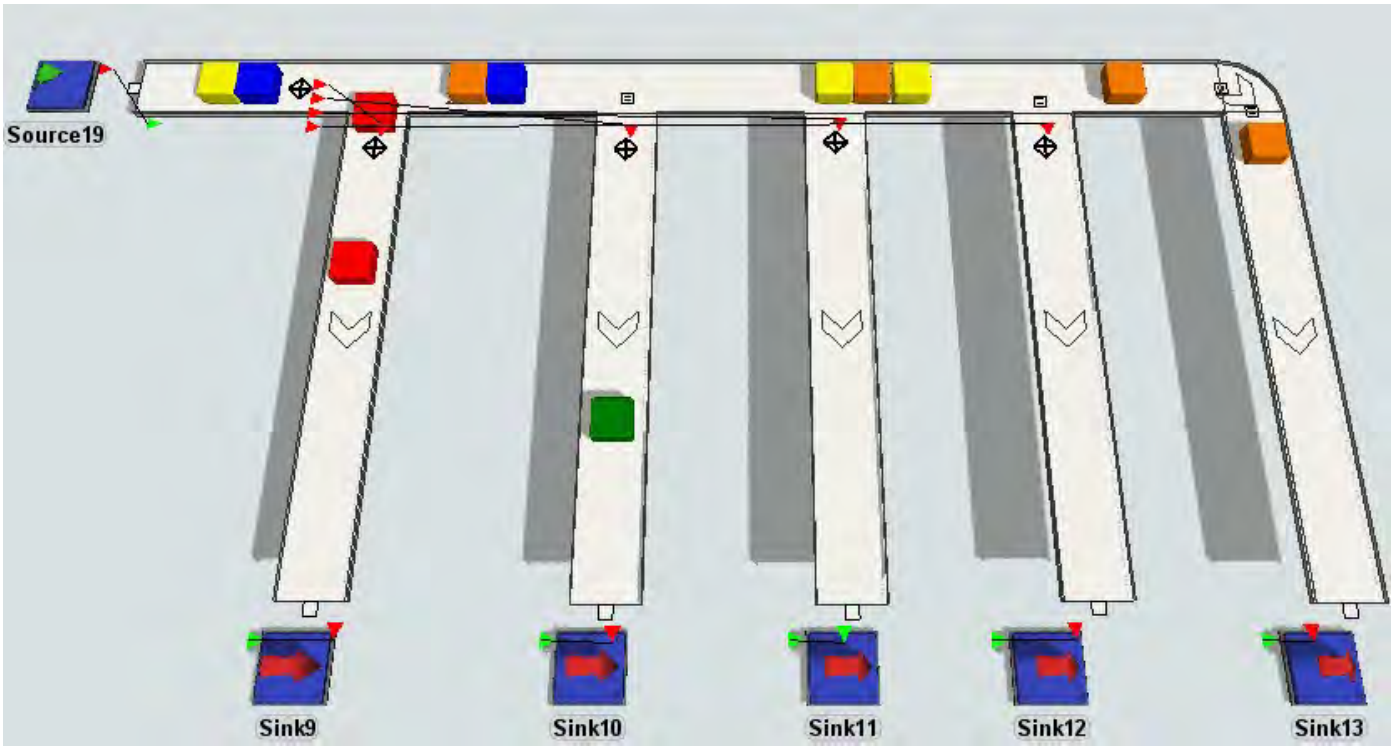
10. Click OK to apply the settings and close the Properties dialog box.

11. To test the simulation, click Reset and Run on the simulation control bar. As the simulation runs, the items should sort to the appropriate conveyor lines.

Sorting Using the Destination Field

To sort items using the Destination field, will will add a single Decision Point to the main line conveyor and connect it to multiple Decision Points on diverting conveyors. Then we will add the Send Item Behavior on the OnArrival Trigger to the Decision Point on the main line. The Destination field on the Send Item

Behavior will determine which lanes the items will be diverted to based on their item type. In this example, items with an itemtype 1 (the red boxes) are diverted to the first conveyor line. Items with an itemtype 2 (the green boxes) are diverted to the second conveyor line, etc., as shown in the image below:

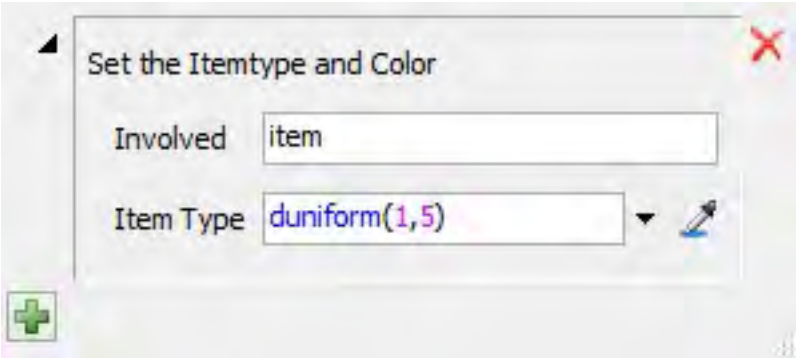


To create this kind of sortation system:

1. Add the conveyors and add any additional Fixed Resources such as Sources and Sinks. (See Adding Conveyors to a Simulation Model.)

2. Connect the conveyors together and connect to the conveyors to the Fixed Resources. (See Connecting Conveyors.)

3. In the Source's Triggers tab, add an OnCreation Trigger to Set Itemtype and Color. Define the Item Type as duniform(1, 5).The Source will randomly create items with a range of itemtypes between 1 and 5. Each itemtype correlates to a specific color.



4. Place a Decision Point at the first junction on the conveyor where items could be diverted to different conveyor lines.

5. Place Decision Points at the beginning of each conveyor line to act as a reference point for the first Decision Point. Notice that in this example, there isn't a Decision Point on the fifth conveyor line. If an item doesn't get diverted to the first four conveyors, it will end in the fifth conveyor line.

6. Make an input/output port connection ('A' connect) from the first Decision Point to each of the other Decision Points.
7. In this example, the order in which you connect the Decision Points matters because the rank of the output port connection will determine where the item is routed. Items with an itemtype 1 will be routed to the Output Port 1, etc. To view or change Output Port rankings, double-click the first Decision Point to open the Properties dialog box. Then click the General tab and select Output Ports from the Ports menu. Re-rank the Output Ports if necessary. (If you need to know the name of each Decision Point, its Properties dialog box displays its name in the top. By default, they are named DP1, DP2, etc. based on the order in which they were added to the model.)

8. Now you will need to add the Send Item behavior to the first Decision Point. With the Properties dialog box still open, click the Decision Point Type tab. Click the Plus sign next to the OnArrival Trigger. Select Send Item from the list of Behaviors. (See Send Item for more in-depth information about this Behavior.)

9. A Behavior Settings dialog box will pop up. In the Condition menu, select Always and the menu will then display the word true. This means that any item arriving at the Decision Point will always be sent.

10. In the Destination menu, select Output Port by Item Type and the menu will then display the expression outobject(current, getitemtype(item)). This means that it will send the item to a port based on its item type. Items with itemtype 1 will be sent to the first Output Port, etc.

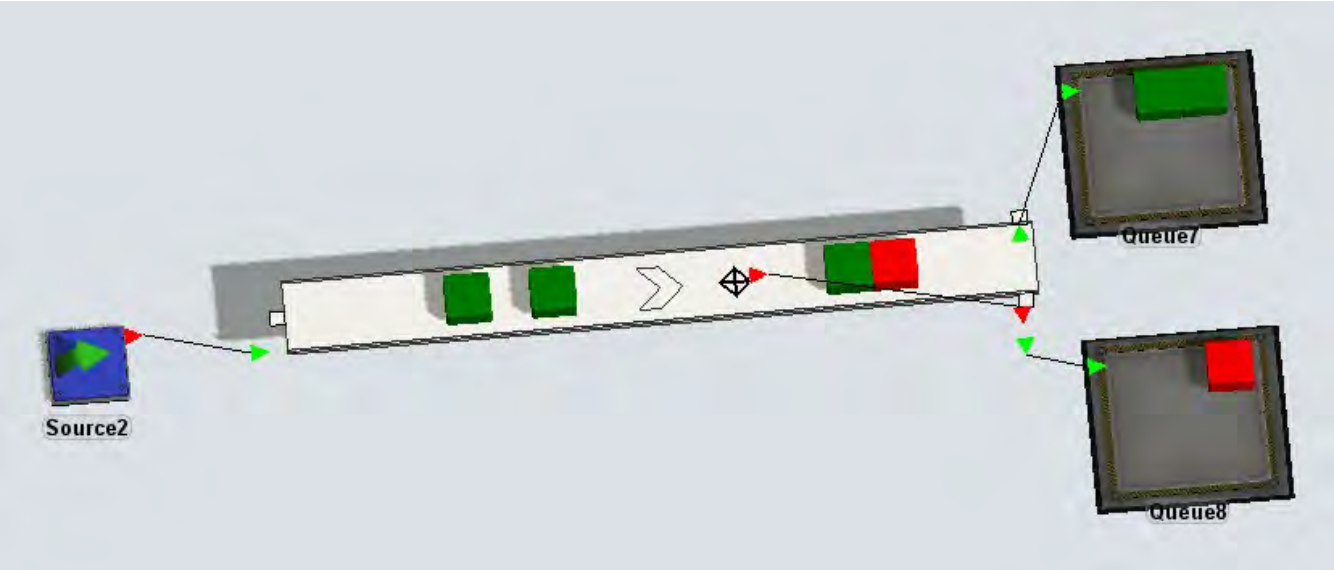
11. Click OK to apply the settings and close the Properties dialog box.

12. To test the simulation, click Reset and Run on the simulation control bar. As the simulation runs, the items should sort to the appropriate conveyor lines.

Sending to Downstream Fixed Resources

When sending items to multiple downstream fixed resources, you connect the Decision Point on the conveyor to Exit Transfers instead of other decision points. Items sent to an exit transfer will go to the object connected to that exit transfer.

The image below shows the result of the decision point sending items with itemtype 1 to the exit transfer connected to Queue8 while any other items are sent to Queue7.



You can use these basic methods to do other kinds of complex sorting. You can familiarize yourself with the available Behaviors by reading the chapter on Photo Eye and Decision Point Behaviors, especially the section on the Send Item Behavior.

Merging and Slug Building

Merge conveyors can organize items flowing from multiple infeed conveyor lines. The most common purpose of merging is to combine items from multiple conveyor lines into a single line, which then sends the items downstream for further processing. This section will describe how to use FlexSim conveyor objects to create various kinds of merges. It will also discuss accumulation methods that could possibly be used when merging items.

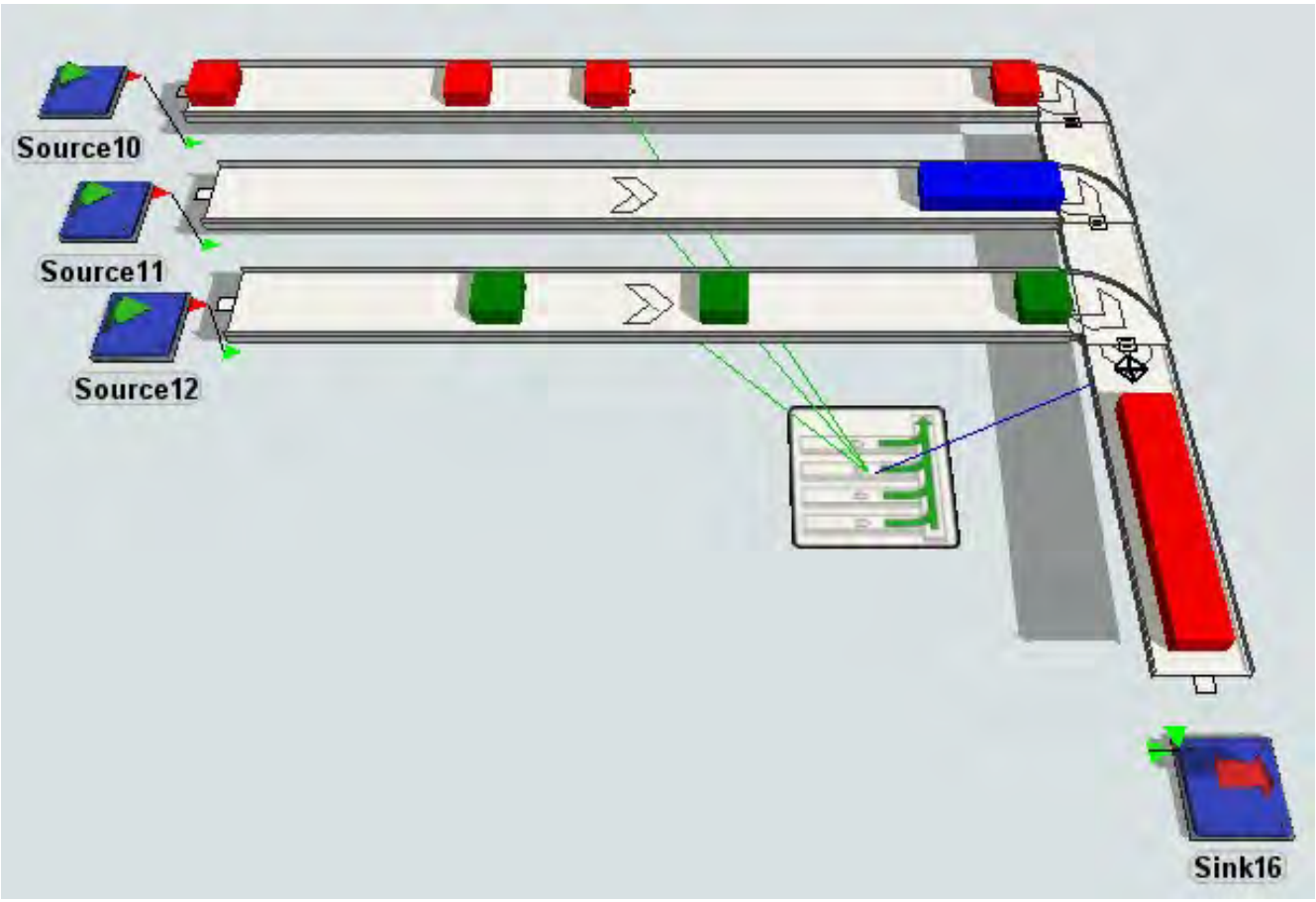
Slug Building

The conveyor industry uses the term *slug* to refer to a queue of accumulated items on a conveyor that will eventually be released downstream as a single group. Building slugs of items can maintain high rates of throughput while keeping equipment speeds as slow as possible.

You can design a conveyor that builds slugs and waits to release them based on a specific set of criteria such as:

- What percentage of the conveyor must be filled before a slug is ready for release
- The number of items that must be on a conveyor before a slug is ready for release
- The amount of time that must elapse in building the slug before a slug is ready for release

For example, in the following image, each conveyor in this system is set to build a slug of at least 5 items before release:



To design conveyors that build slugs before releasing them, you will need to create a Conveyor Type that is specifically designed to build slugs. In the settings for your Conveyor Type, check the Slug Builder check box to enable slug building. Then, in the Ready Criteria, check the boxes next to the conditions you would like the conveyor to use when determining a slug is complete and ready for release. See Conveyor Type Settings - Slug Building Settings for more instructions on these settings.

The next section will provide a step-by-step set of instructions for building the conveyor system used in the preceding image.

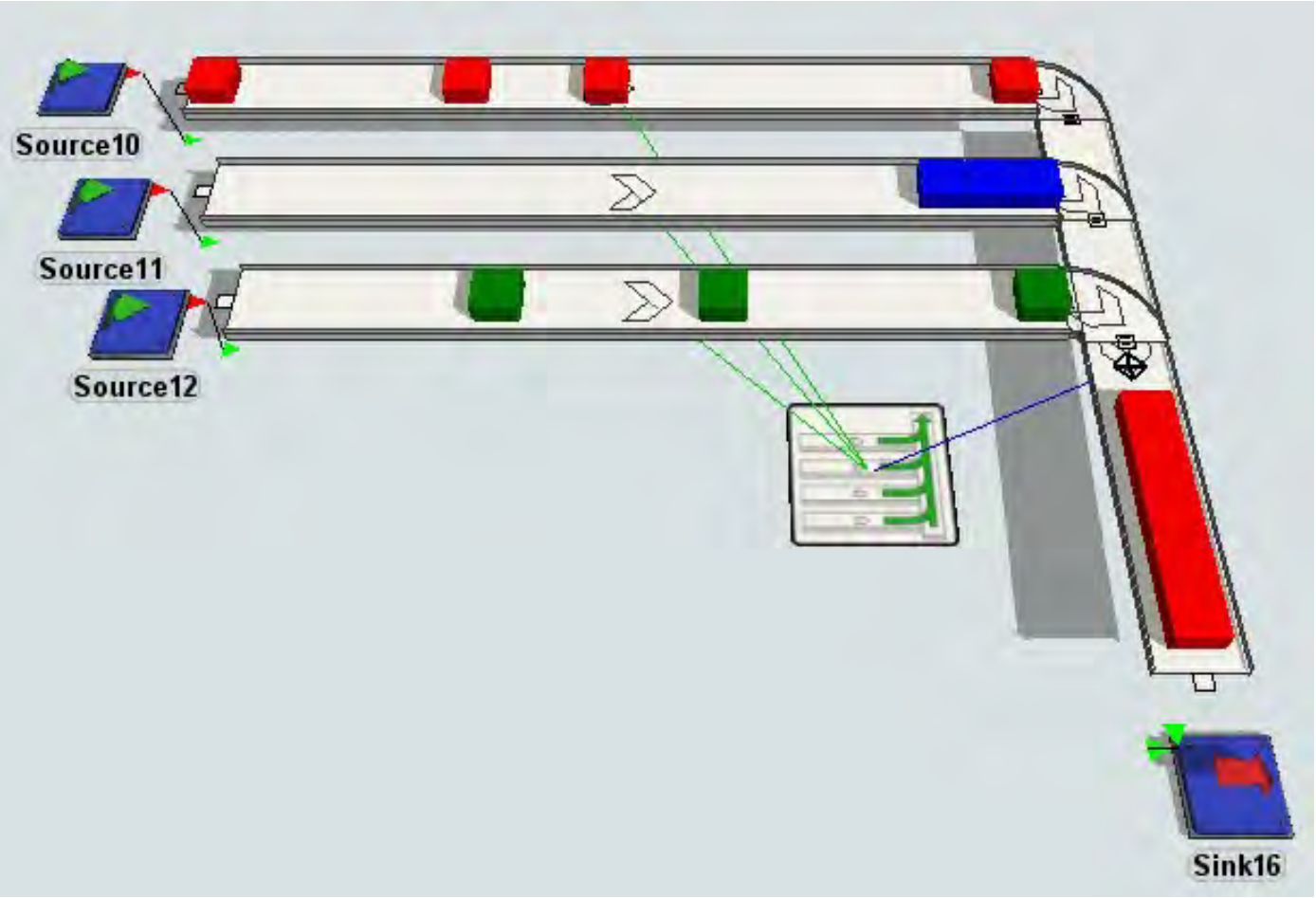
Sawtooth Merging Using the Merge Controller

The Merge Controller uses its Lane Clear Table and Release Strategy in determining how to prioritize merging lanes. Release strategies can be:

- Round Robin - Conveyor lanes will merge in the order the lanes are listed in its Lane Clear Table. Only after the first lane has merged will the second lane merge, etc.
- Round Robin if Available - Conveyor lanes will merge as soon as the lane becomes available for merging. If two lanes are available at the same time, it will then use the ordering in the Lane Clear Table to determine which lane should take priority.

Also, since the Release Strategy is just a code field, you can manually write your own release strategy if the pre-defined options do not fit your needs.

To see the Merge Controller in action, we'll return to the Slug Building example from the previous section. In the following image, each conveyor will not be available to merge until it has built a slug of at least 5 items. The Merge Controller in this case will use the Round Robin if Available strategy and allow lanes to merge whenever the slug is built:



To create this kind of merging system:

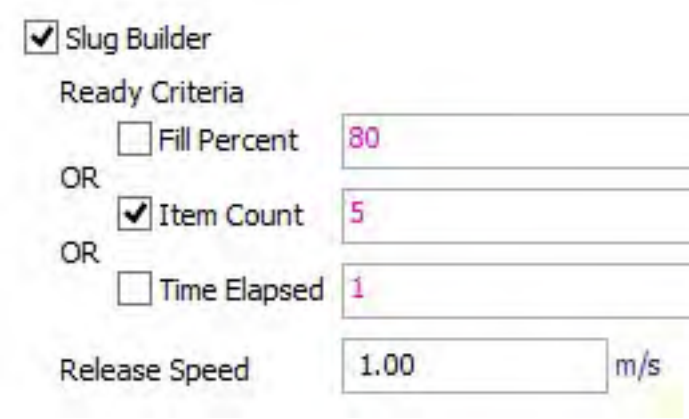
1. Add the conveyors and add any additional Fixed Resources such as Sources and Sinks. (See Adding Conveyors to a Simulation Model.)

2. Connect the conveyors together and connect to the conveyors to the Fixed Resources. (See Connecting Conveyors.)

3. On the View menu, click Toolbox to open the Conveyor System Tool. (See Accessing Type Properties for other methods of opening the tool.)

4. In the Conveyor System list, right-click Conveyor Types and select Add Conveyor Type from the menu. This will help you to create a type of conveyor that builds slugs before making the slug lane available to merge. (See Conveyor Type Settings for more information.)

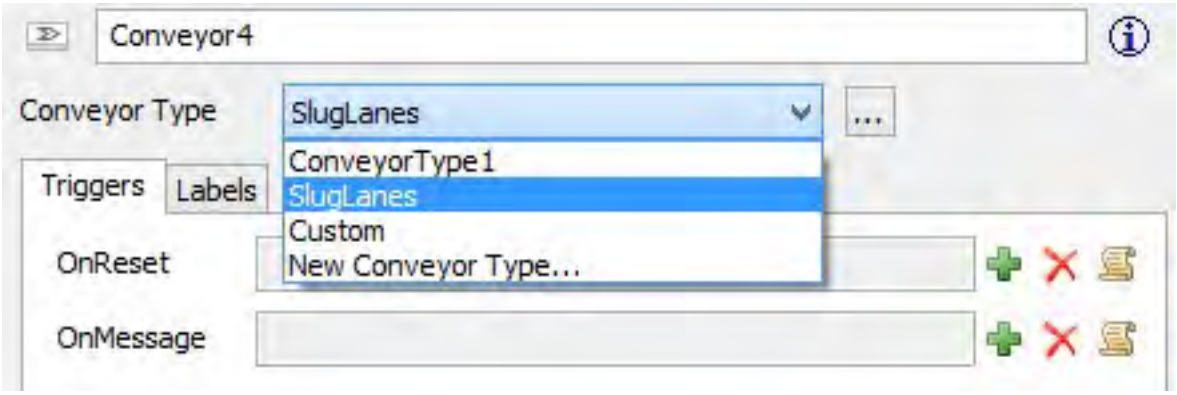
5. In this example, we're going to set the conveyor to build a slug of at least 5 items before becoming available to merge. On the Behavior tab, check the Slug Builder check box. This will enable the Ready Criteria options. Check the Item Count check box and change the number next to it to 5. Now it will build a slug of at least 5 items before making the lane available for release.



6. If you would like, you could change the name of this new Conveyor Type by editing the top field. In this example, we'll change the name to SlugLanes.

7. Click OK to apply these settings and close the Properties dialog box. The SlugLanes conveyor type should now appear in the list of Conveyor Types in the Conveyor System Tool.

8. Double-click each slug-building conveyor to open its Properties dialog box. From the Conveyor Type menu, select the SlugLanes conveyor type. The slug-building conveyor will now inherit all of those settings and will build slugs of at least 5 items.



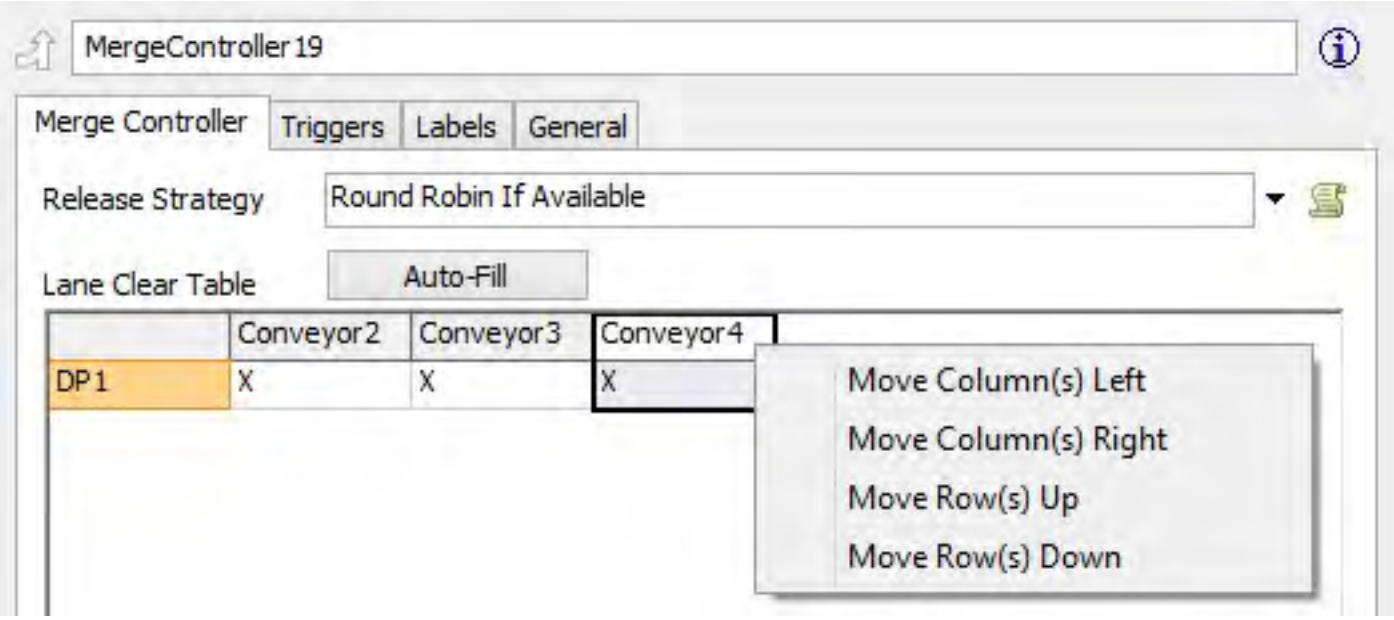
9. Add a Merge Controller anywhere to the simulation model.

10. Add a Decision Point on the conveyor in the location where the merging items will be sent.

11. Using an 'A' connector, connect the Merge Controller to the Decision Point and to each slug-building conveyor.

12. Double-click the Merge Controller to open the Properties dialog box. On the Merge Controller tab, on the Release Strategy menu, select Round Robin If Available.

13. If needed, you can reorganize the order in which the Merge Controller prioritizes the lanes using the Lane Clear Table. In this table, the conveyors will be prioritized from left to right. To reorganize the prioritization, right-click a column and select Move Column(s) Left or Move Column(s) Right.



14. Click OK to apply the settings and close the Properties dialog box.

15. To test the simulation, click Reset and Run on the simulation control bar. As the simulation runs, the lanes should build slugs of at least five items and they should be released one at a time.

You can use these basic methods to do other kinds of complex merging with Merge Controllers. You could possibly familiarize yourself with the available Behaviors by reading the chapter on the Conveyor System Tool and Photo Eye and Decision Point Behaviors.

Inline vs. Side Transfers on a Merge

When using the merge controller, you may or may not decide to use small curved sections to merge the items onto the main merge lane. If you do not use curved sections, you must make sure that the transfers onto the merge lane are inline transfers. If they are side transfers, the additional gapping caused by performing a side transfer may cause the merge to block unexpectedly. To force these transfers to be inline transfers, increase the Max Angle value under For Inline Transfers in the transfers' type properties.

Merging Using Decision Points and Photo Eyes

You can use a Decision Point or Photo Eye to build complex merging logic into your conveyor system. You can build nearly any kind of system by positioning and connecting Decision Point and Photo Eyes and adding some Behaviors to them. A few of the Decision Point or Photo Eye Behaviors that could be useful for merging are:

- Send Item - Can send items to different destinations (such as different conveyor lines) based on criteria you specify
- Stop/Resume - Can stop an item, delay an item, and can set conditions that can make a conveyor motor stop or restart
- Area Restriction - Can create an area that only allows a fixed number of items to flow through a restricted area on the conveyor at a time

See the chapter on Decision Point and Photo Eye Behaviors for a more in-depth discussion of each Behavior and its available settings.

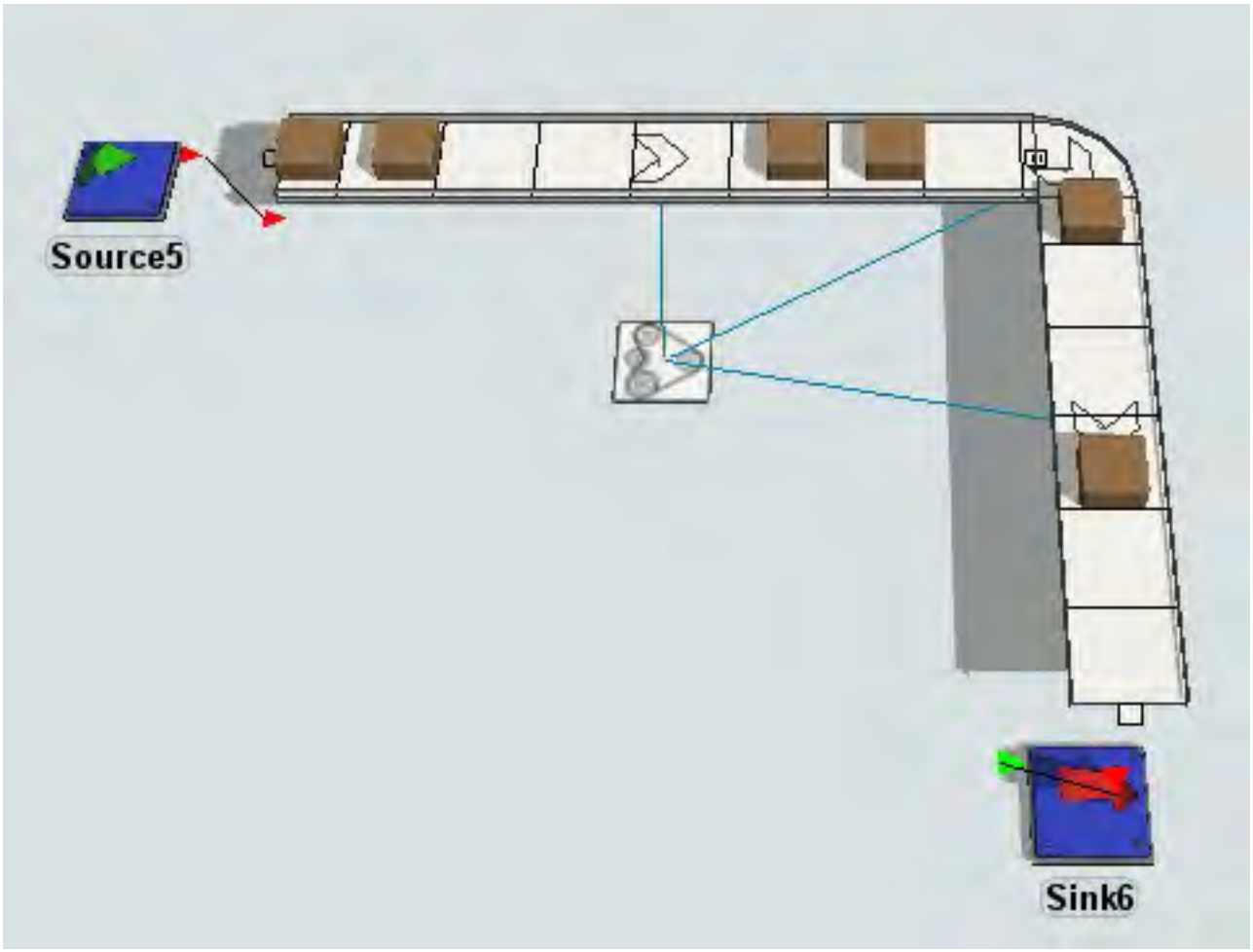
Gapping

There are several methods you can use to create gaps between items on a conveyor:

- Conveyor Types - Define custom Conveyor Type spacing settings to enforce gaps between items.
- Use Decision Points or Photo Eyes o Add Stop Item and Delay to stop an item and wait before resuming.
    - Add Set Conveyor Speed behavior to a Decision Point or Photo Eye to adjust gaps between items by setting conveyor speed.
    - Use Area Restriction behavior to enforce gapping between items.

Power and Free conveyor systems are designed for industrial applications that require more flexibility and a greater capacity than conventional conveyors. They provide the unique ability to stop individual loads without stopping the entire production line.

You can simulate both inverted and overhead power and free conveyors using FlexSim. In these systems, dogs travel at fixed intervals along a looping chain. These dogs pick up carriers in the system as they pass them. In simulation terms it is similar to having a moving space greater than stopping space (there is a caterpillar-like accumulation effect), except that the point at which an item can move on the conveyor is defined by the location of the next passing dog, instead of by the space between it and the item ahead of it. The following image shows an example of a simple inverted Power and Free system in FlexSim:



In order to set up a Power and Free system in FlexSim, you need Straight or Curved Conveyors that have Conveyor Type Settings with Power and Free enabled. In this example, there are three conveyors (two Straight Conveyors and one connecting Curved Conveyor) and a Motor which is connected to all three conveyors. Although we use a Motor in this example, it is not technically required. Motors are only needed if the default dog positioning method is insufficient. Notice that there are also lines running across the conveyors to represent the dogs on the tracks.

To create this kind of Power and Free conveyor system:

1. Add the conveyors and add any additional Fixed Resources such as Sources and Sinks. (See Adding Conveyors to a Simulation Model.)

2. Connect the conveyors together and connect the conveyors to the Fixed Resources. (See Connecting Conveyors.)

3. Place a Motor somewhere in the model.

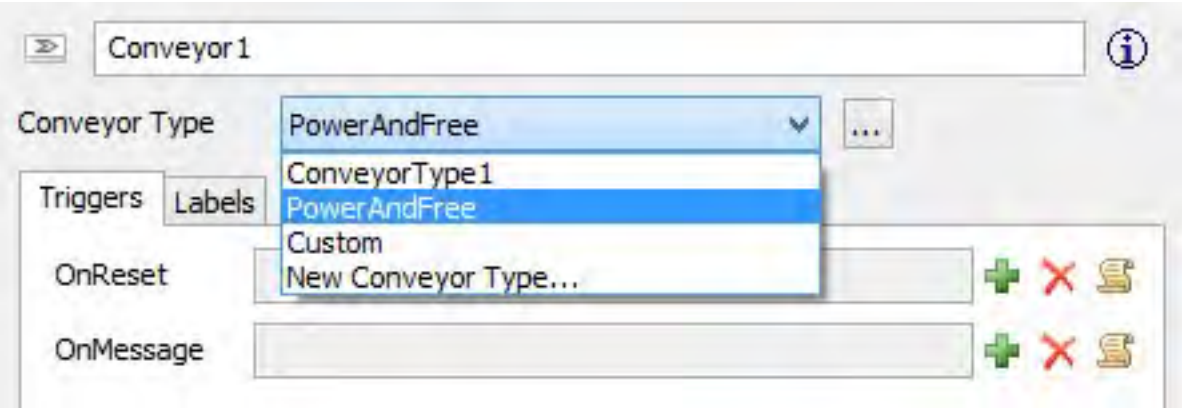
4. Connect the Motor to the conveyors. (See Adding Conveyors to a Motor.)

5. Double-click the Motor to open its Properties dialog box. In the Motor tab, check the Sync Power and Free Dog Positions check box. If needed, you could also check the Adjust Dog Gap for Continuous Loop check box. (See Motors - Power and Free Settings.)

6. If you would like, you could change the name of this new Conveyor Type by editing the top field. In this example, we'll change the name to PowerAndFree.

7. Click OK to apply these settings and close the Properties dialog box. The PowerAndFree conveyor type should now appear in the list of Conveyor Types in the Conveyor System Tool.

8. Double-click each merging conveyor to open its Properties dialog box. From the Conveyor Type menu, select the PowerAndFree conveyor type.



9. Click OK to apply the settings and close the Properties dialog box. The conveyor will now inherit the Power and Free settings.

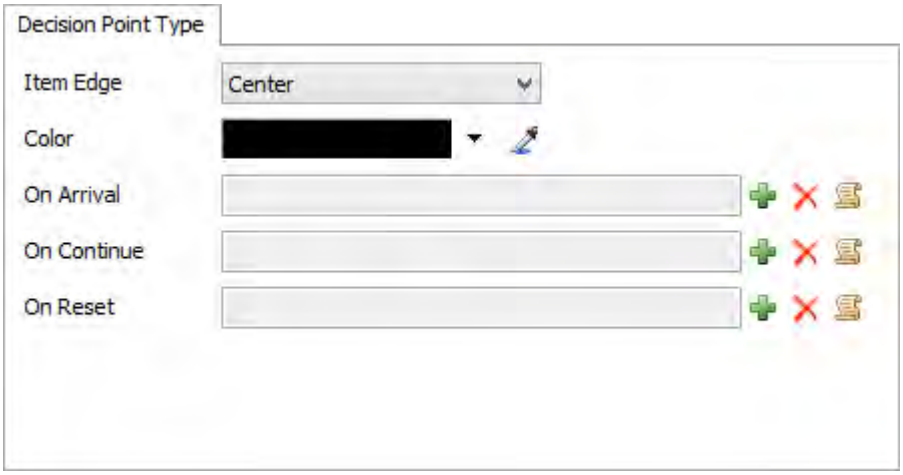
10. To test the simulation, click Reset and Run on the simulation control bar. As the simulation runs, you will notice lines going across the conveyor to represent the dogs in the conveyor.

Overhead Power and Free Systems

To create an overhead power and free system, put a Decision Point or Photo Eye at the beginning (tail) of the first conveyor. On its OnArrival Trigger, select Movement then, click Translate Item. Set the Conveyor Up setting to a negative number, such as -2 or -zsize(item), and it will hang below the conveyor while moving. NOTE: You might also want to raise each conveyor's Z-axis so that they are higher off the ground.

The Decision Point and Photo Eye objects have several useful Pick List Behaviors. Pick Lists are preprogrammed FlexScript functions that simplify common tasks or behaviors in simulation models. Decision Points and Photo Eyes have nearly identical Pick Lists. The main difference between a Decision Point and Photo Eye is in the timing of Triggers. A Decision Point has Triggers that fire when an item arrives at or continues through the point on the conveyor. A Photo Eye has Triggers that fire when an item blocks or clears it. See Decision Point Type Settings and Photo Eye Type Settings for more information on respective Trigger behaviors.

You can add behaviors to a Decision Point or Photo Eye by double clicking on these objects in your simulation model, which opens the Properties dialog box. By default, the Properties dialog box will open to a Decision Point Type tab or a Photo Eye tab that will display various Triggers to which you can add behaviors, as shown in the following images:



You can add a behavior by clicking the Plus sign next to the Trigger.

Decision Points and Photo Eye Connections

Decision Points and Photo Eyes are independent objects in the simulation model. This means you can connect them to each other or to other objects in the model using standard input/output or center port connections. Many of the Pick List Behaviors, such as Area Restriction and Send Item, use this feature by default.

The Current and Conveyor Commands

When you add a Pick List Behavior to a Decision Point or Photo Eye, FlexSim will automatically populate the Pick List field with FlexScript commands. Sometimes you might see the command current in some of the commands. In this case, the current command is a reference to the Decision Point or Photo Eye. It is NOT a reference to the conveyor. This means that if you use a command like centerobject(current, 1) in a Pick List field, this will reference the object connected to the Decision Point or Photo Eye's center port 1, NOT the object connected to the conveyor's center port 1. You can use the conveyor command to refer to data and connections on the conveyor. See Basic Modeling Functions for more information on current and object referencing.

Behaviors

The following are some of the available Decision Point or Photo Eye Behaviors:

- Send Item - Can send items to different destinations (such as different conveyor lines) based on criteria you specify
- Stop/Resume - Can stop an item, delay an item, and can set conditions that can make a conveyor motor stop or restart
- Area Restriction - Can create an area that only allows a fixed number of items to flow through a restricted area on the conveyor at a time
- Movement - Can rotate, tilt, or move items at a particular point on a conveyor
- Set Conveyor Speed - Can change the speed of the conveyor based on certain conditions

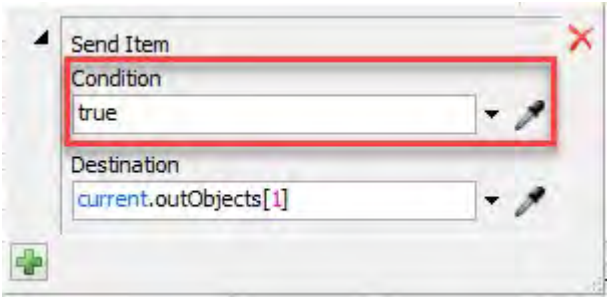
The other sections in this chapter will discuss each of the available behaviors and their settings.

Local Settings vs. Global Type Settings

Certain conveyor objects, such as Decision Points and Photo Eyes can have either local settings or global type settings. That means that you can have settings that are unique to that particular object (local) or you can create a global Type for that kind of object that will allow you to import those settings to other objects of that kind. By default, all Decision Point and Photo Eye settings will be local only and will be labeled as "Custom" in the Type menu. To make the settings a global Type, see Adding and Renaming New Conveyor System Types.

Nearly all of the Behaviors available for Decision Points and Photo Eyes have a Condition field. The Condition field allows you to customize the criteria that causes a Behavior to start. The Condition field is a boolean value in which *true (non-zero)* tells it to perform the Behavior, while *false (zero)* will cause the Behavior not to be executed.

The following image shows an example of the Condition field for the Send Item Behavior:

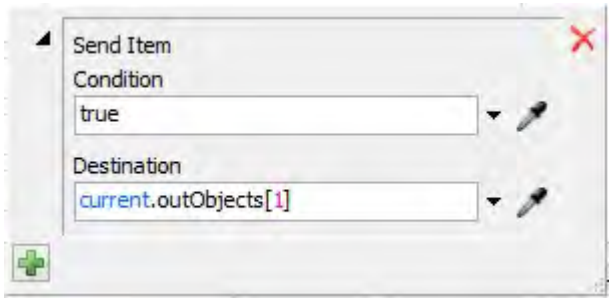


Click the drop-down menu on the right for suggestions on values to use. Possible expressions may be:

- true - The behavior will always be executed.
- getitemtype(item) == 1: The behavior will be executed if the item's itemtype value is equal to 1.
- item.DivertLane == 1: The behavior will be executed if the item's "DivertLane" label value is equal to 1.
- item.Category == current.Category: The behavior will be executed if the item's "Category" label value is equal to the Decision Point/Photo Eye's "Category" label.
- gettablenum("RoutingTable", 1, 1) == 1: Gets the value in row 1, column 1 of the global table named RoutingTable. If that value is 1, the behavior will be executed.

See Writing Logic in FlexSim for more information on building code expressions in FlexSim.

Select Send Item from the pick list if you want the Decision Point or Photo Eye to send the item to a particular destination you define. When selected, the following dialog box will appear:



The Condition field defines the condition by which the item will be sent to the destination. See The Condition Field for more information.

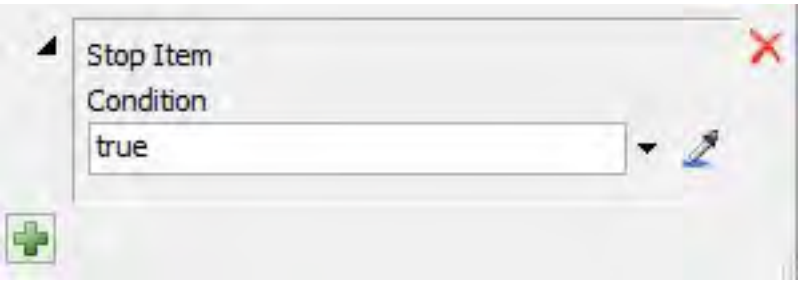
The Destination field determines where to send the item. This should be a reference to an object that is a point on the conveyor system, such as another Decision Point or Photo Eye, or an Entry Transfer or Exit Transfer. The default options assume that you connect output ports from the Decision Point or Photo Eye to the defined destination. See Sorting for help on setting this up. Use the drop-down box on the right for suggestions on what you might enter here. Possible values might include:

- current.outObjects[1]: Sends the item to the object connected to output port 1 of the Decision Point or Photo Eye.
- current.outObjects[duniform(1, current.outObjects.length)]: Sends the item to the object connected to a random output port of the Decision Point or Photo Eye.
- current.outObjects[getitemtype(item)]: Sends the item to the object connected to the output port number of the Decision Point or Photo Eye that is the same as the item's itemtype value.
- current.outObjects[item.Routing]: Sends the item to the object connected to the output port number of the Decision Point or Photo Eye that is the same as the item's "Routing" label value.

Select Stop/Resume from the list of Behaviors if you want the Decision Point or Photo Eye to allow the item to stop when the Trigger fires and then possibly later resume. If you select Stop/Resume from the list of Behaviors, a submenu will appear with several options. The following sections will discuss each submenu option.

Stop Item

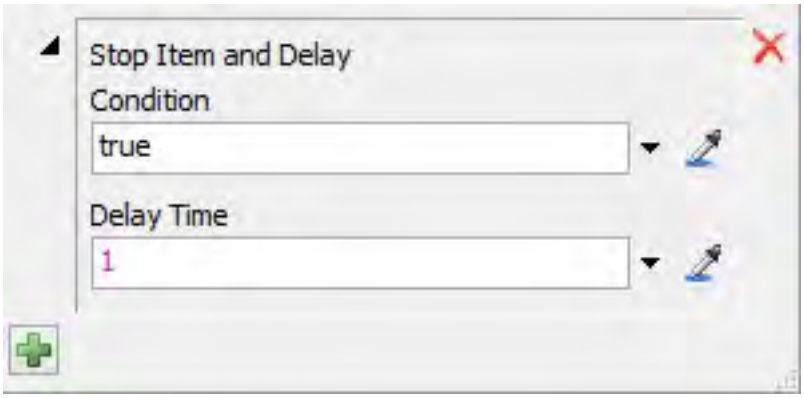
Select Stop Item from the menu if you want the Decision Point or Photo Eye to stop the item when the Trigger fires. When selected, the following dialog box will appear:



The Condition field defines the condition by which the item will be stopped. See The Condition Field for more information.

Stop Item and Delay

Select Stop Item and Delay from the field if you want the Decision Point or Photo Eye to stop the item for a period of time before releasing it. When selected, the following dialog box will appear:

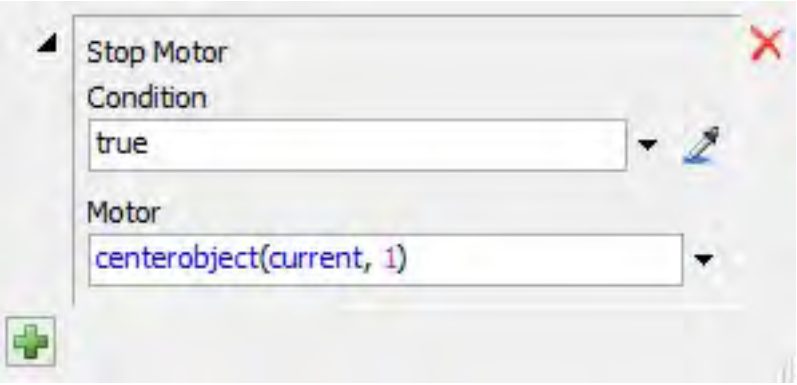


The Condition field defines the condition by which the item will be stopped. See The Condition Field for more information.

The Delay Time field defines the time that the item will wait before resuming, defined in simulation model units.

Stop Motor

Select Stop Motor from the list of Behaviors if you want the Decision Point or Photo Eye to stop a conveyor motor when the Trigger is fired. See Motor for more information. When selected, the following dialog box will appear:

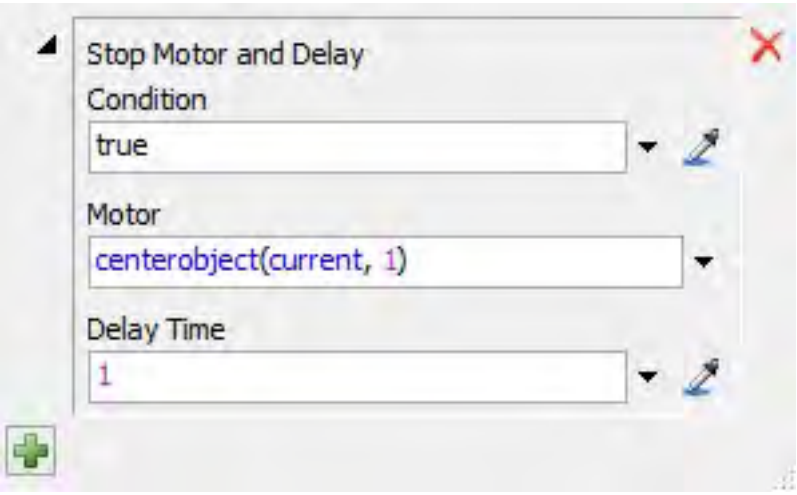


The Condition field defines the condition by which the item will be stopped. See The Condition Field for more information.

The Motor field is a reference to the motor that you want to stop. The default is centerobject(current, 1). If you use this default, you should make sure that the motor is connected to the Decision Point or Photo Eye's first center port ('S' connect).

Stop Motor and Delay

Select Stop Motor from the list of Behaviors if you want the Decision Point or Photo Eye to stop a conveyor motor for a defined period of time. See Motor for more information. When selected, the following dialog box will appear:



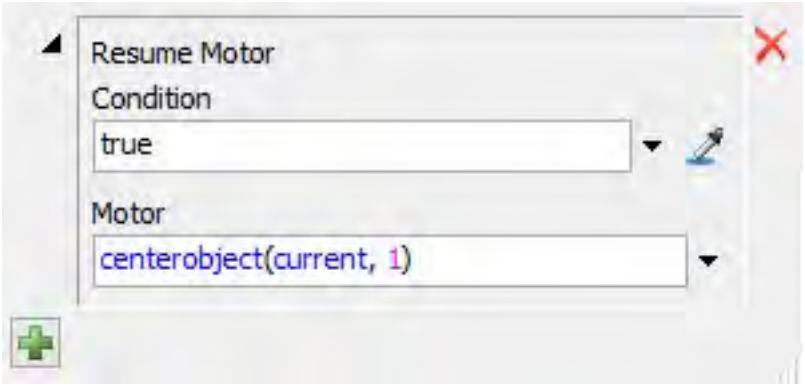
The Condition field defines the condition by which the motor will be stopped. See The Condition Field for more information.

The Motor field is a reference to the motor that you want to stop. The default is centerobject(current, 1). If you use this default, you should make sure that the motor is connected to the Decision Point or Photo Eye's first center port ('S' connect).

The Delay Time field defines the time that the motor will be stopped before resuming, defined in simulation model units.

Resume Motor

Select Resume Motor from the list of Behaviors if you want the Decision Point or Photo Eye to restart a Motor. See Motor for more information. When selected, the following dialog box will appear:



The Condition field defines the condition by which the motor will be resumed. See The Condition Field for more information.

The Motor field is a reference to the motor that you want to stop. The default is centerobject(current, 1). If you use this default, you should make sure that the motor is connected to the Decision Point or Photo Eye's first center port ('S' connect).

Area Restriction

The Area Restriction behavior allows you to build logic for conveyor merging and flow control. When you have multiple merging conveyor lines that should only merge one at a time, you could possibly use Area Restriction to control access to the receiving conveyor line.

See the end of this topic for examples on how to use Area Restriction.

In order to understand how Area Restriction works, you should understand a few key terms, as they are used within FlexSim:

- Restricted Area - An area within the conveyor system that is restricted, meaning items can only pass through the Area under certain conditions. The physical boundaries of the Restricted Area can be defined by the logic you build into Decision Points or Photo Eyes and by where you place the Decision Points or Photo Eyes on the conveyor (e.g. you could use Decision Points or Photo Eyes as entry/acquire points, and as exit/release points). You can restrict access to the Area by limiting the number of items (or slugs if you are using Photo Eyes) that can be inside the Area at a time, or by allowing only one Area Claimer to acquire the Area at a time.

- Area Claimer - If you are using the Acquire Area behavior, you need to define the object that is *claiming* the Area. The Claimer can be an item, a Decision Point or Photo Eye, or a conveyor. The Claimer defines how other items will be excluded from the Area. If the Claimer is an item, then only the item that has acquired the Area can continue into the Area. If the Claimer is a Decision Point or Photo Eye, then once it acquires the Area, any item passing over that Decision Point or Photo Eye can continue into the Area. If it is a conveyor, then once it aquires the Area, any item on the same conveyor can continue into the Area. In the following image, we show an example of a scenario in which Decision Points are acting as Area Claimer, but keep in mind that there are many other possible types of Claimers.

- Area Owner - A Decision Point or Photo Eye that can determine whether to allow access to an Area Claimer so that the accumulated items can pass through. Typically an Area Owner is downstream on the conveyor object.

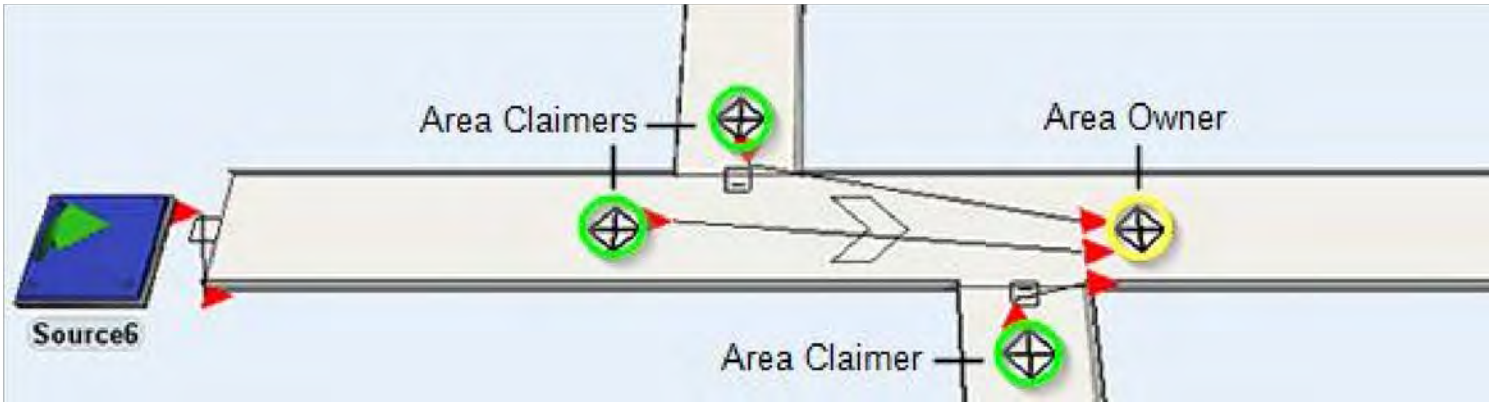
- Acquire Area - When the items being held by the Area Claimer are granted exclusive access to pass through an Area.

- Release Area - When all items have finished passing through the restricted Area and the Area returns to the control of the Area Owner.

Area Restriction requires interaction between two or more Decision Points or Photo Eyes. Typically you will associate these objects with each other by creating input/output port connections ('A' connect). The following image illustrates an example of Area Restriction system which merges three different conveyor lines into one. Each merging line has an Area Claimer (in this case, a Decision Point), each of which has output ports connected to a single downstream Area Owner.

In a Restricted Area, each Area Claimer would need to have Area Restriction behaviors that can coordinate with the corresponding Area Restriction behavior on the Area Owner. There are two possible Area Restriction combinations:

- Enter/Exit Area - Use this combination when you want to give multiple items or groups of items access to a Restricted Area at a time. The items will attempt to enter the Area until the item limit is reached. At that point, any subsequent items must wait at the entry point until an item exits the area. When you build this kind of Area Restriction system, use the Enter Area behavior on the Area Claimers and the Exit Area behavior on the Area Owner.



- Acquire/Release Area - Use this combination when you only want one item to be in the Restricted Area at a time. The Area Claimer will request permission to get exclusive access to pass through a Restricted Area.Only one Area Claimer can acquire the area at a time. Once released, another Claimer may acquire the area. When you build this kind of Area Restriction system, use the Acquire Area behavior on the Area Claimers and the Release Area behavior on the Area Owner.

Each of the four available behaviors has one field in common: the Area Owner field. The Area Owner field allows you to determine which object will be the Area Owner for the Restricted Area. For entering/acquiring, the default is current.outObjects[1], whereas for exiting/releasing, the default is current. If you use these default values, you should connect the entry/acquire points' first output port to the exit/release point (the Area Owner).

The following sections will discuss each of the Area Restriction behavior settings.

Enter Area

Select Enter Area from the Restricted Area submenu if you want to limit the number of items that are allowed into the Area at a single time.

When selected, the following dialog will pop up:

The Condition field allows you to customize the criteria that the Decision Point or Photo Eye will use to determine whether to start the behavior when a particular Trigger fires. For more information about the options available in the Condition field, see The Condition Field.

The Area Owner field allows you to determine which object will be the Area Owner for the Restricted Area. See the description of the menu options in the introductory section for this topic for more information.

You can type in a specific number in the Max Content box. The number will specify how many items can have access to the Restricted Area at a given time.

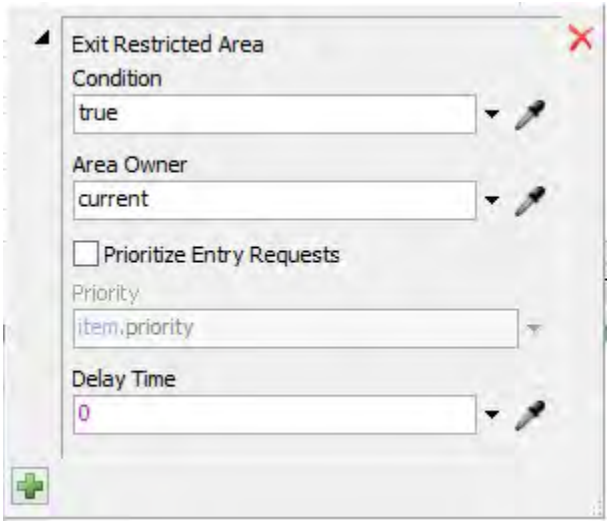
All Objects Should Use the Same Max Content

If there are multiple Decision Points or Photo Eyes that are using the Enter Area behavior, they should all have the same Max Content number.

Exit Area

Select Exit Area from the Restricted Area submenu if you want an item to exit the restricted area at the Decision Point or Photo Eye.

When selected, the following dialog box will pop up:



The Condition field allows you to customize the criteria that the Decision Point or Photo Eye will use to determine whether to start the behavior when a particular Trigger fires. For more information about the options available in the Condition field, see The Condition Field.

The Area Owner field allows you to determine which object will be the Area Owner for the Restricted Area. See the description of the menu options in the introductory section for this topic for more information.

If you don't check the Prioritize Entry Requests check box, by default the Decision Point or Photo Eye will accept incoming Area Requests on a first in, first out (FIFO) basis. However, if you check this box, the Decision Point or Photo Eye will prioritize certain Area Requests over others. Once this box is checked, the Priority menu will become available.

The Priority field defines the priority value associated with a specific item trying to enter the area. Use the drop-down on the right for suggestions. Possible values might include:

- item.up == conveyor: Gives priority to requesting items already on the release point's conveyor, i.e. if the item's current conveyor (item.up) is the same as the release point conveyor (conveyor), the == comparison returns true (1), which is a higher priority value than false (0).

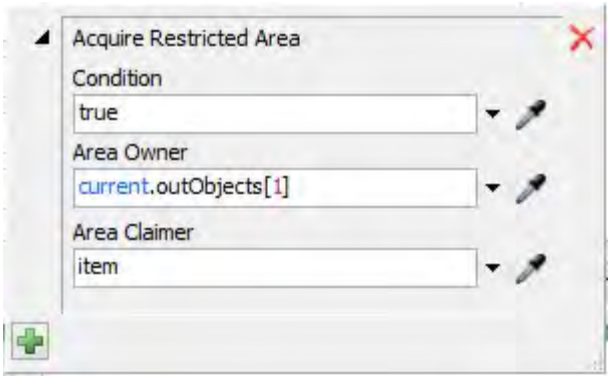
- item.priority: Use the value of the requesting item's "priority" label for the priority.

- getitemtype(item): Use the value of the requesting item's itemtype for the priority.

The Delay Time field lets you define an optional delay time to wait before actually executing the behavior to exit the area.

Acquire Area

Select Acquire Area from the Restricted Area submenu if you want the Decision Point or Photo Eye to request permission to get exclusive access to pass through a Restricted Area when the Trigger fires. When selected, the following dialog box will pop up:



The Condition field allows you to customize the criteria that the Decision Point or Photo Eye will use to determine whether to start the behavior when a particular Trigger fires. For more information about the options available in the Condition field, see The Condition Field.

The Area Owner field allows you to determine which object will be the Area Owner for the Restricted Area. See the description of the menu options in the introductory section for this topic.

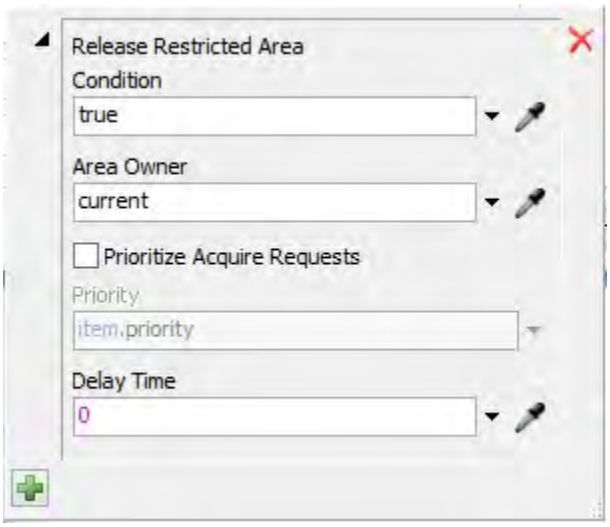
The Area Claimer menu allows you to determine which object will be the Area Claimer for the Restricted Area. Typically this will be one of the following values:

- item - The current item on the Decision Point or Photo Eye will be the Area Claimer.
- current - The current Decision Point or Photo Eye will be the Area Claimer.
- conveyor - The conveyor will be the Area Claimer.

Release Area

Select Release Area from the Restricted Area submenu if you want the Area Owner to release the Area, meaning it will resume control of the Area after an item that has been given access passes through the Area.

When selected, the following dialog box will pop up:



The Condition field allows you to customize the criteria that the Decision Point or Photo Eye will use to determine whether to start the behavior when a particular Trigger fires. For more information about the options available in the Condition field, see The Condition Field.

The Area Owner field allows you to determine which object will be the Area Owner for the Restricted Area. See the description of the field options in the introductory section for this topic for more information.

If you don't check the Prioritize Acquire Requests check box, by default the Decision Point or Photo Eye will accept incoming Area Requests on a first in, first out (FIFO) basis. However, if you check this box, the Decision Point or Photo Eye will prioritize certain Area Requests over others. Once this box is checked, the Priority field will become available.

The Priority field defines the priority value associated with a specific item trying to enter the area. Use the drop-down on the right for suggestions. Possible values might include:

- item.up == conveyor: Gives priority to requesting items already on the release point's conveyor, i.e. if the item's current conveyor (item.up) is the same as the release point conveyor (conveyor), the == comparison returns true (1), which is a higher priority value than false (0).
- item.priority: Use the value of the requesting item's "priority" label for the priority.
- getitemtype(item): Use the value of the requesting item's itemtype for the priority.

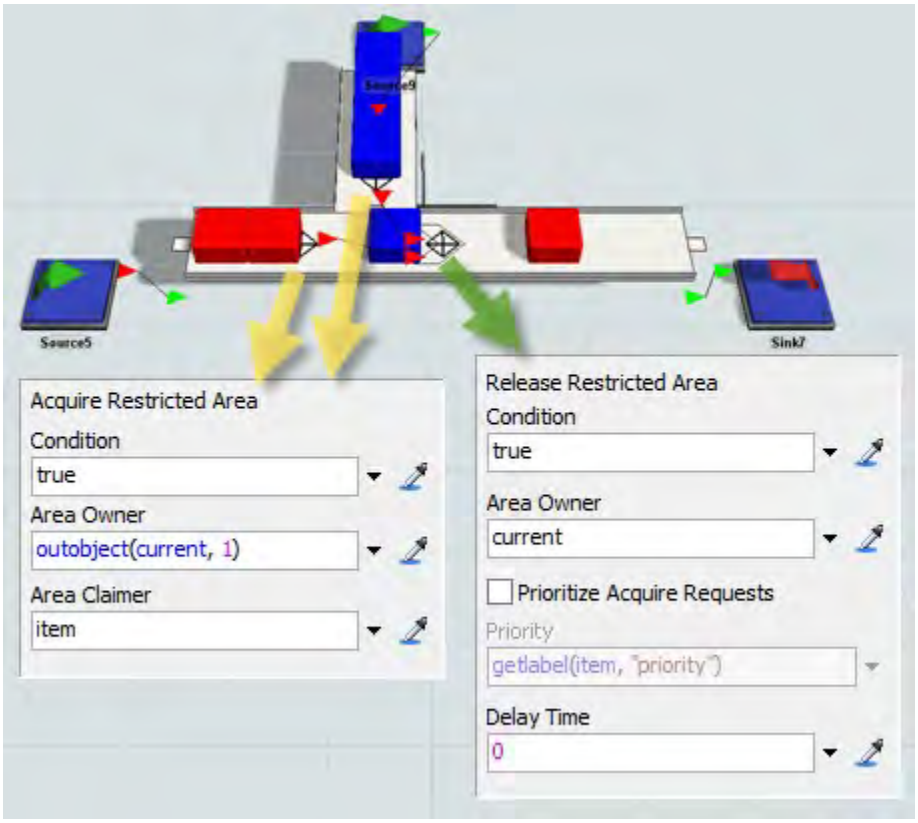
The Delay Time field lets you define an optional delay time to wait before actually executing the behavior to releasing the area.

Examples

Below are several examples of using Area Restriction.

One-at-a-Time Merging

The following diagram shows an example of a simple one-at-a-time merge.



To build this model:

1. Create and connect the sources, sink, and two conveyors as shown.
2. Drop three Decision Points onto the conveyors at the positions shown.
3. Connect the two upstream Decision Points to the single downstream Decision Point using the 'A' connection tool.
4. Double click on each of the upstream Decision Points, and in the On Arrival trigger, choose Area Restriction > Acquire Area. Make sure the behavior properties are the same as in the diagram shown above.
5. Double click on the downstream Decision Point, and in its On Arrival trigger, choose Area Restriction > Release Area. Make sure the behavior properties match those in the diagram above.
6. Reset and run the simulation.

The merge should only let one item into the area at a time. Once an item passes the release point, the next item will be allowed in. By default, the items will enter in fifo order, i.e. the first item that hits the acquire point will be the first one to enter the area.

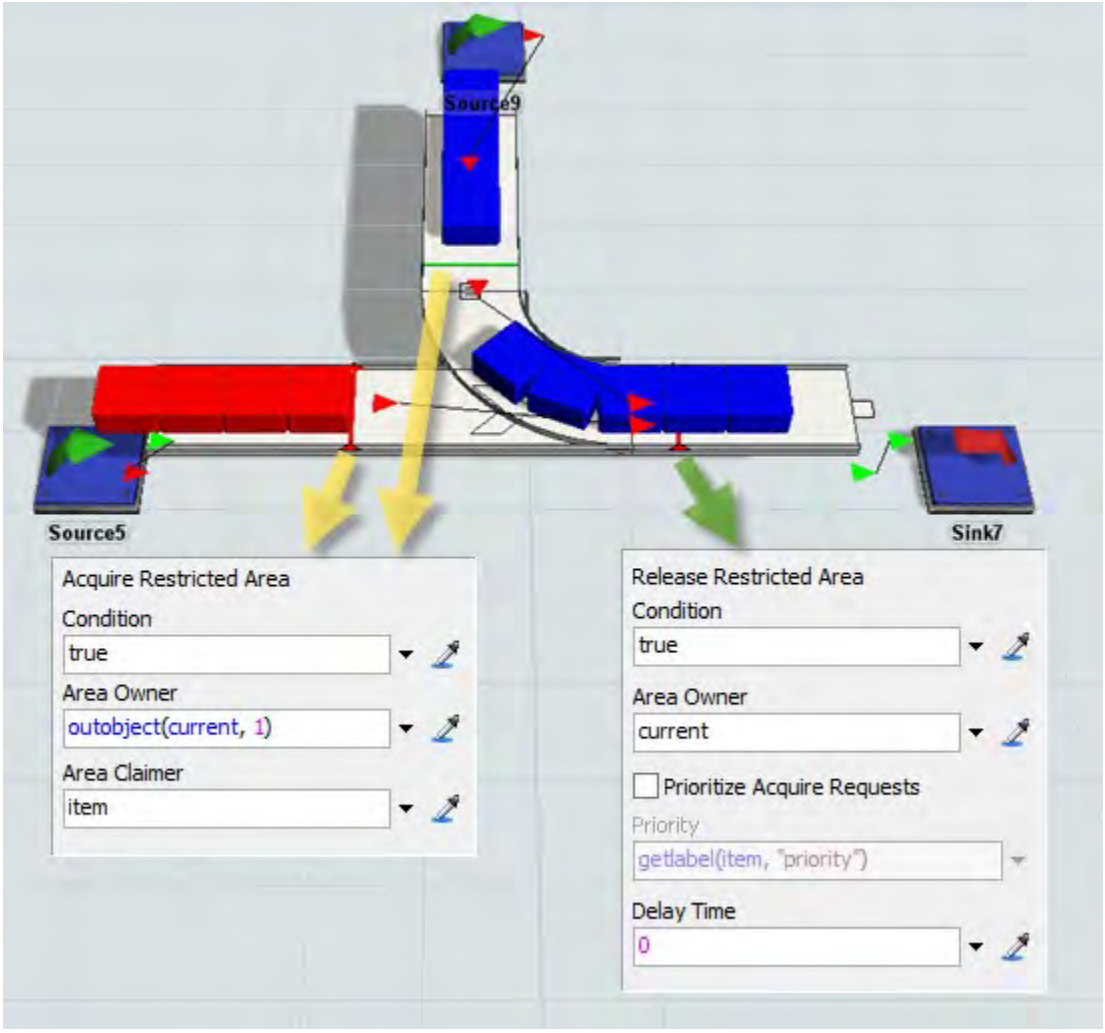
Now let's change a setting and see how it affects the functionality. On the upstream acquire points, change the Area Claimer, from item to current. Then run the simulation again.

In this case, the Decision Point, not the item itself, will "claim" the area. The effect is that as long as items acquire through the same Decision Point that has claimed the area, they will be allowed in. Only once they've all exited the area will it allow the next batch in from a different "claimer". Note here that the mechanism will keep a counter of how many items have entered. Only when the counter goes to zero will the area be released. This can have subtle and sometimes unintended consequences especially if you are using Photo Eyes, where blocking/clearing is dependent on a gap between the items. If gaps are introduced or removed while items are

in the area, the counter may get off and break the merging mechanism. Hence you should be careful when using a non-item claimer in conjunction with Photo Eyes.

Slug-Based Merging

Slug-based merging is very similar to the One-at-a-Time example, but here we instead use Photo Eyes. The difference is that if multiple items are backed up against each other, by default the Photo Eye will block/clear only once for the entire slug, meaning a whole slug can pass through for a single acquire. The diagram below shows an example of a simple slug-based merge.



To build this model you can refactor the one-at-a-time example, or build from scratch:

1. Create and connect the sources, sink, and two straight conveyors as shown.

2. Using the Join Conveyors tool, create a curved conveyor from the merging conveyor to the main line conveyor.

3. Drop three Photo Eyes onto the conveyors at the positions shown.

4. Connect the two upstream Photo Eyes to the single downstream Photo Eyes using the 'A' connection tool.
5. Double click on each of the upstream Photo Eyes, and in the On Block trigger, choose Area

Restriction > Acquire Area. Make sure the behavior properties are the same as in the diagram shown above.

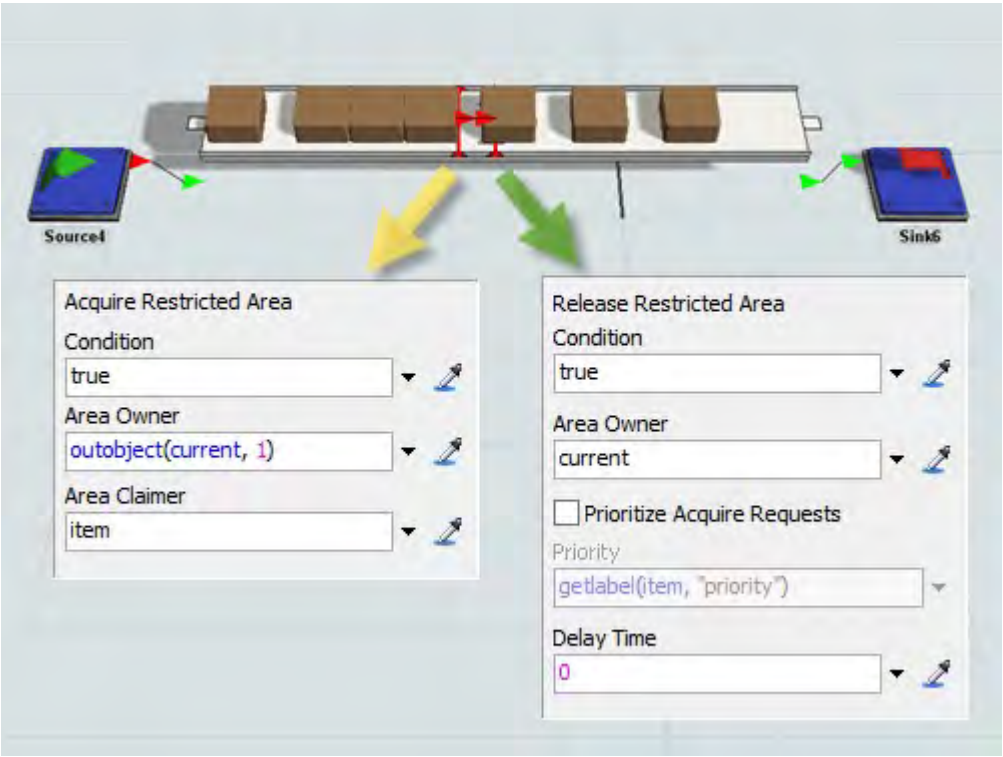
6. Double click on the downstream Photo Eye, and in its On Clear trigger, choose Area Restriction > Release Area. Make sure the behavior properties match those in the diagram above.

7. Reset and run the simulation.

Now entire slugs should pass through the area.

Gapping

You can also use Area Restriction to define gapping. The following diagram shows an example gapping model.



To build this model:

1. Create and connect the sources, sink, and conveyor as shown.

2. Drop two Photo Eyes onto the conveyor at the positions shown.

3. Connect the upstream Photo Eye to the downstream Photo Eye using the 'A' connection tool.

4. Double-click on the upstream Photo Eye, and uncheck the box Require Gap to Clear.

5. In the Photo Eye's On Block trigger, choose Area Restriction > Acquire Area. Make sure the behavior properties are the same as in the diagram shown above.

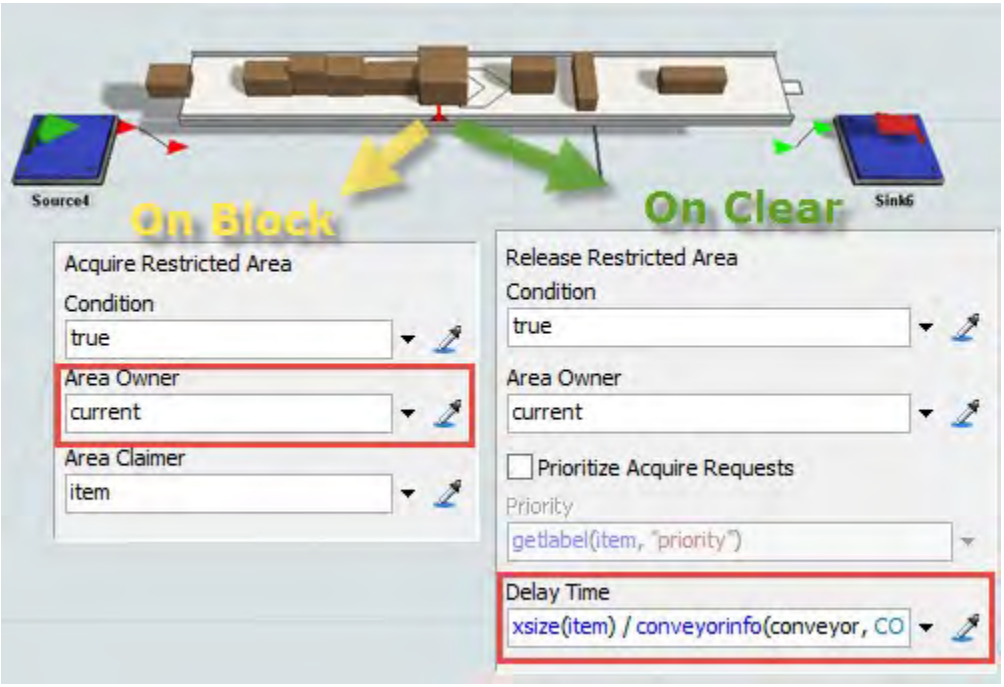
6. Double click on the downstream Photo Eye, and in its On Clear trigger, choose Area Restriction > Release Area. Make sure the behavior properties match those in the diagram above.

7. Reset and run the simulation.

This will enforce a gap between items that pass over the photo eyes. The gap from back-to-front will be the distance between the Photo Eyes.

Variable Gapping

You can also use Area Restriction to simulate variable gapping. The diagram below shows a model that implements variable gaps between items, based on the item size.



To build this model, you can refactor the previous gapping model, or build from scratch:

1. Create and connect the sources, sink, and conveyor as shown.
2. Drop a Photo Eye onto the conveyor at the position shown.
3. Double-click on the Photo Eye, and uncheck the box Require Gap to Clear.
4. In the Photo Eye's On Block trigger, choose Area Restriction > Acquire Area. Set the behavior properties to match the diagram shown above. Make sure you change the Area Owner to current.
5. In the Photo Eye's On Clear trigger, choose Area Restriction > Release Area. Set the behavior properties to match those in the diagram above. Make sure you change Delay Time by choosing Time to Convey Item X Size from its drop-down.
6. Reset and run the simulation.

This will enforce a variable gap between items, equal to the x size of the ahead item.

Movement

Use the Movement submenu if you want the Decision Point or Photo Eye to change the orientation (rotation, tilt angle, etc.) or location of the item. The Movement submenu has two options: Rotate Item and Translate Item.

The Rotate Item and Translate Item Behaviors use kinematics-based offsets. This means that these Behaviors don't change the item's properties, logic, or orientation (with some exceptions, as noted below). These Behaviors are primarily only visual. In light of the fact that items will only appear to change visually, you should keep in mind:

- You will likely not want to translate the item along the conveyor-forward axis (forward or backward along the conveyor), unless you are willing to have accumulation and other conveyor logic fire at weird points visually because the item has been visually offset forward or backward from the location where the conveyor is tracking it.
- If items are offset on the conveyor-left axis, and they pass over photo eyes with an offset angle applied, they will still cover/clear the photo eye as if they were in the center of the conveyor.

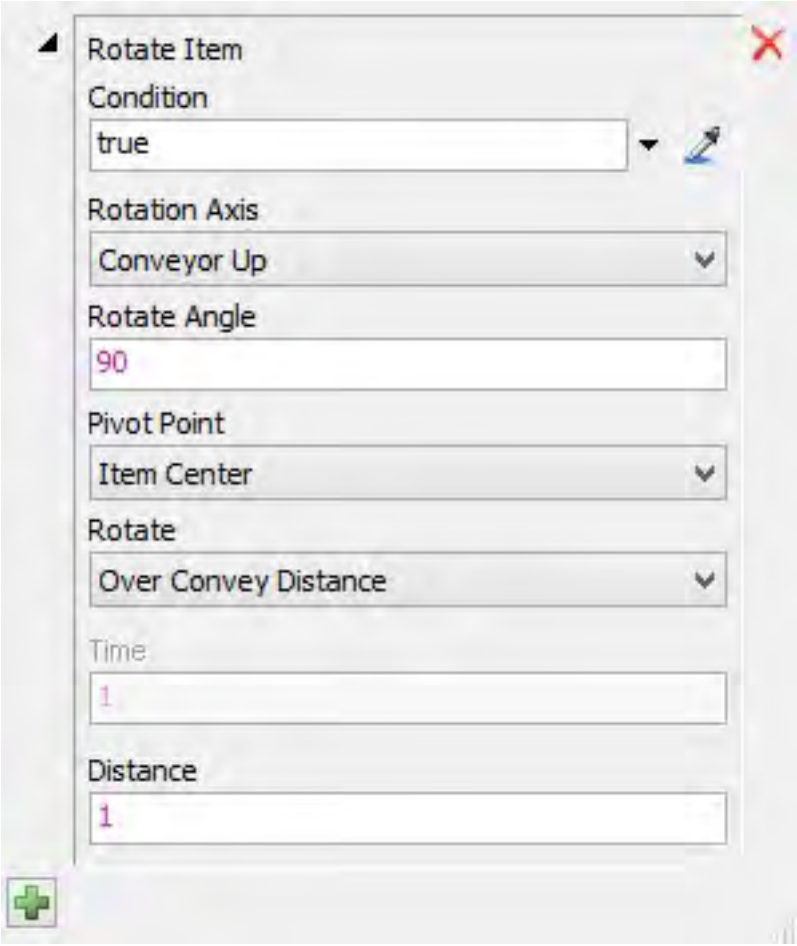
However, there are some exceptions to this visual-only rule:

- When an item starts a side transfer, the transfer logic will take the item's conveyor-left offset into account in determining how far the item must move to complete the transfer.
- If a photo eye's height is non-zero, then when an item's front edge reaches the photo eye, the cover logic will take the item's current conveyor-up offset into account in determining whether the item covers the photo eye.

The following sections will discuss each Movement submenu option.

Rotate Item

Select Rotate Item from the Movement submenu if you want the Decision Point or Photo Eye to rotate and/or tilt the item when the Trigger fires. When selected, the following dialog box will appear:



The Condition field defines the condition by which the item should be rotated. See The Condition Field for more information.

The Rotation Axis menu determines the axis around which the item will be rotated, relative to the conveyor. It has the following three options:

- Conveyor Up - The item will rotate around the axis pointing up on the conveyor.
- Conveyor Left - The item will rotate around the axis pointing left on the conveyor.
- Conveyor Forward - The item will rotate around the axis pointing forward along the conveyor.

The Rotate Angle text box determines the degree of the angle that the item will rotate. So, if the Rotate Angle is set to 90, the item will rotate 90 degrees. You can also enter in a negative number to move in the opposite direction.

Rotation Angle Between 60 and 120

If you enter in a rotation angle between 60 and 120 (or between -60 and -120), the behavior will change the conveyor orientation of the item. This means that the item's length, width and height, as tracked by the conveyor, will be recalculated based on the newly-rotated orientation of the item. Note that this may have weird effects if the item is straddling a conveyor transfer when the behavior executes. In such a case, make sure the Decision Point that executes the logic is on the upstream conveyor.

The Pivot Point menu controls what part of the item will be the pivot point when it rotates. The available options depend on which axis is being rotated. For Conveyor Up, for example, the available options will be:

- Item Center

- Item Front Left
- Item Back Left
- Item Back Right
- Item Front Right

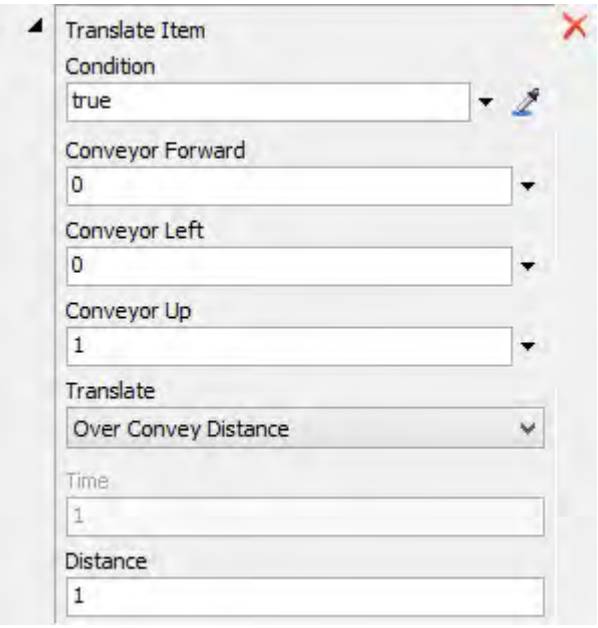
The Rotate menu determines how or when the item will complete its rotation. It has the following options:

- Over Time - Once the rotation begins, the item will complete the rotation after a set amount of time has elapsed. The rotation will be completed whether the item gets stopped on the conveyor or not.
- Over Conveyor Distance - Once the rotation begins, the item will complete the rotation over a particular convey distance. The rotation will pause if the item gets stopped on the conveyor.

The Time text box is only available when Over Time is selected from the Rotate menu. Enter in the amount of time it should take the item to rotate. This text box will use the model's units of measurement, so if you enter `10` and the model's time unit is in seconds, it will take 10 seconds to rotate.
The Distance text box is only available when Over Conveyor Distance is selected from the Rotate menu. Enter in the distance the item will need to travel before it rotates. This text box will use the model's units of measurement, so if you enter `1` and the model's length unit is in meters, it will take 1 meter to rotate.

Translate Item

Select Translate Item from the Movement submenu if you want the Decision Point or Photo Eye to move the item when the Trigger fires. When selected, the following dialog box will appear:



The Condition field defines the condition by which the item will be moved. See The Condition Field for more information.

The Conveyor Forward, Conveyor Left, and Conveyor Up fields determine the position of the item as it moves. The position is relative to the conveyor:

- Conveyor Forward - The item will move forward on the conveyor (the conveyor's head).
- Conveyor Left - The item will move left on the conveyor.
- Conveyor Up - The item will move up on the conveyor.

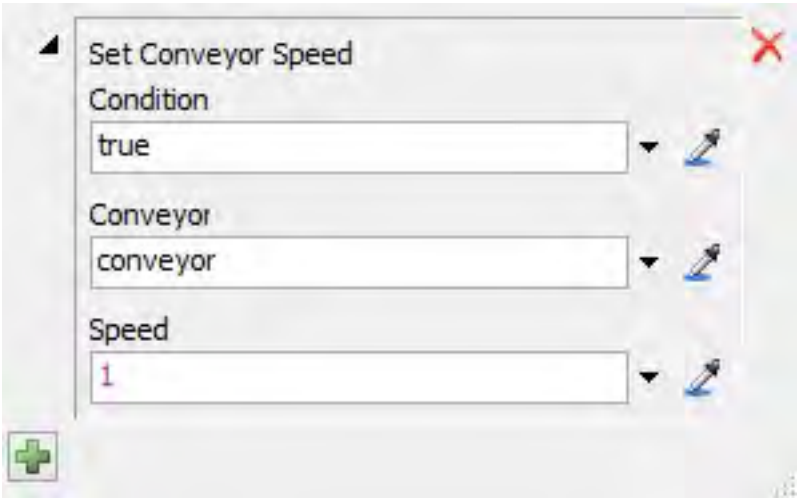
The Translate menu determines how or when the item will complete its movement. It has the following options:

- Over Time - Once the movement begins, the item will complete the movement after a set amount of time has elapsed. The movement will be completed whether the item gets stopped on the conveyor or not.
- Over Conveyor Distance - Once the movement begins, the item will complete the movement over a particular convey distance. The movement will pause if the item gets stopped on the conveyor.

The Time text box is only available when Over Time is selected from the Translate menu. Enter in the amount of time it should take the item to move. This text box will use the simulation model's units of measurement, so if you enter 10 and the model's time unit is in seconds, it will take 10 seconds to move. The Distance text box is only available when Over Conveyor Distance is selected from the Translate menu. Enter in the distance the item will need to travel before it moves. This text box will use the simulation model's units of measurement, so if you enter 1 and the model's length unit is in meters, it will take 1 meter to move.

Set Conveyor Speed

Select Set Conveyor Speed from the menu if you want the Decision Point or Photo Eye to change the speed of a conveyor when the Trigger fires. When selected, the following dialog box will appear:



The Condition field defines the condition by which the speed will be changed. See The Condition Field for more information.

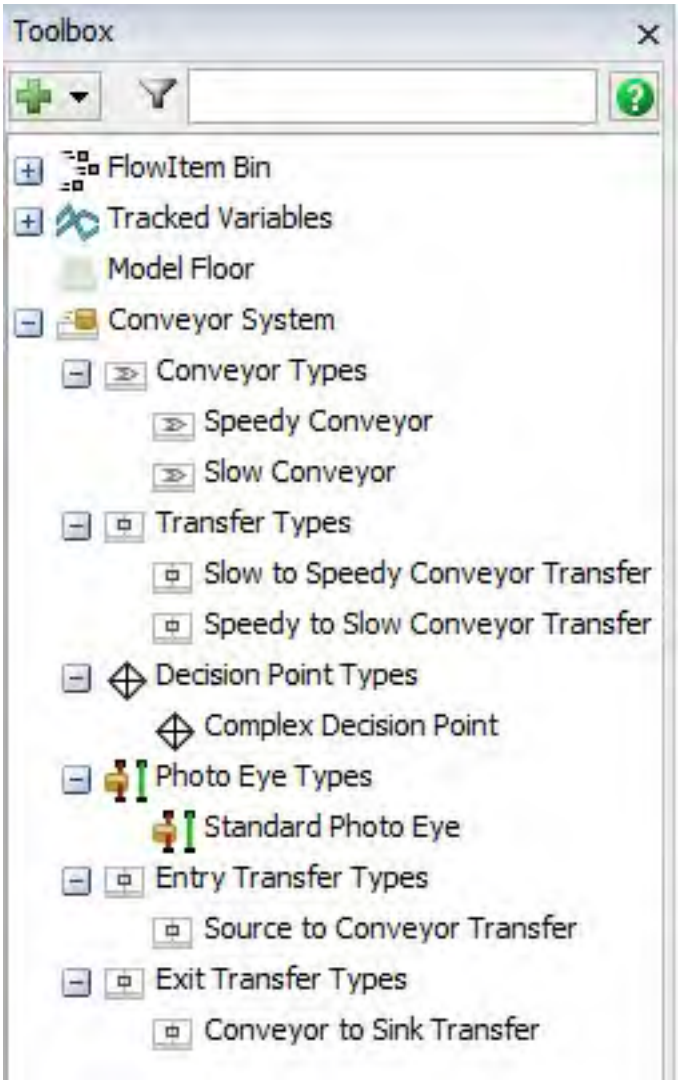
The Conveyor field allows you to select which conveyor's speed will change when the Behavior starts. Use the drop-down arrow on the right for suggestions. Possible values may include:

- conveyor - Changes the speed of the current conveyor that the Decision Point or Photo Eye is on.
- centerobject(current, 1) - Changes the speed of the conveyor that is connected to center port 1 of the Decision Point or Photo Eye.
- centerobject(conveyor, 1) - Changes the speed of the conveyor that is connected to center port 1 of the conveyor that the Decision Point or Photo Eye is on. You can edit the number 1 to change it to a different output port if needed.

The Speed field specifies the speed to set, defined in simulation model units.

Introduction to the Conveyor System Tool

The Conveyor System tool allows you to edit the properties of all the different conveyor objects in a single, easy-to-use interface. Using the Conveyor System tool, you can create global settings for various conveyor objects and then quickly import those settings to other conveyor objects. The following image shows an example of a Conveyor System Toolbox:



In the FlexSim User Interface, each group of global settings for a particular object is referred to as a *Type*. You could possibly think of each Type as a class of objects that will inherit the same custom settings. You can create an unlimited number of Types for each conveyor object, each with its own custom settings. You can also give each Type a unique, custom name.

For example, imagine that you wanted to create several conveyors that will all have a speed of 100.0 feet per minute. Using the Conveyor System Tool, you could create a Type of conveyor with a speed of 100.0 feet per minute and assign the title "Speedy Conveyor" to this new conveyor Type. Then, when you add other conveyors to the model, you could assign the Speedy Conveyor Type to the conveyors and they would all immediately have a speed of 100.0 feet per minute. You can use the Conveyor System tool to create Types for:

- Conveyors
- Transfers (including transfers between conveyors and transfers between other FlexSim objects) • Decision Points
- Photo Eyes

Within the Conveyor System toolbox, you can open the Type Properties dialog box which will allow you to create or edit the global settings for each Type. The Type Properties dialog box will look different for each kind of conveyor object because each object has different kinds of settings and behaviors. So, the Type

Properties box for a Straight Conveyor will be different than the Type Properties box for a Photo Eye, etc. For more information about the Type Properties dialog box and different settings for each object, see:

- Conveyor Types
- Transfer Types
- Decision Point Types
- Photo Eye Types
- Entry Transfer Types
- Exit Transfer Types

Local Settings vs. Global Type Settings

All conveyor objects can have either local settings or global Type settings. That means that you can have settings that are unique to that particular object (local) or you can create a global Type for that kind of object that will allow you to import those settings to other objects of that kind.

When you add either a Straight or Curved Conveyor to a simulation model, FlexSim automatically creates two new global Types in the Conveyor System: a Conveyor Type labeled *ConveyorType1* and a Transfer Type labeled *TransferType1*. Both of these Types are global and will inherit the default settings until you change the settings for these Types. See Managing Conveyor System Types for more information.

All other Types (Decision Point Types, Photo Eye Types, Entry Transfer Types, or Exit Transfer Types) will be local unless you specify otherwise. They will be labeled *Custom* in the Type menu. See Adding and Renaming New Conveyor System Types for more information.

Accessing Type Properties

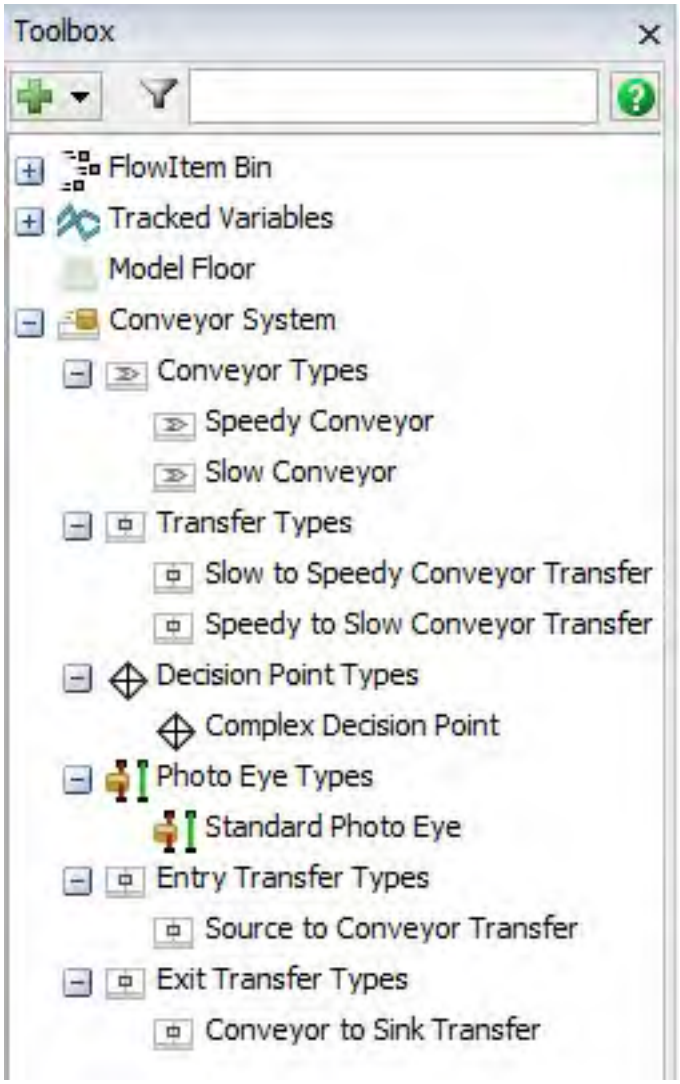
Use the Type Properties dialog box to create or edit the settings for each Type. You can access the Type Properties dialog box using either of the following methods:

- Through the Conveyor System tool in the Toolbox
- Quick Properties
- Properties dialog box

The method you should use is a matter of personal preference, but only the Toolbox method will give you a full overview of entire conveyor system at the same time. Each method for accessing the Conveyor System Tool will be discussed in the following sections.

Using the Toolbox

To access the Type Properties dialog box through the Conveyor System tool in the Toolbox, click View on the main menu, then click Toolbox to open the Toolbox. Alternatively, you could access the Toolbox by clicking the Tools button on the toolbar. From here, you can manage the global settings and Types for your conveyor model. The following image shows an example of a Conveyor System Toolbox:



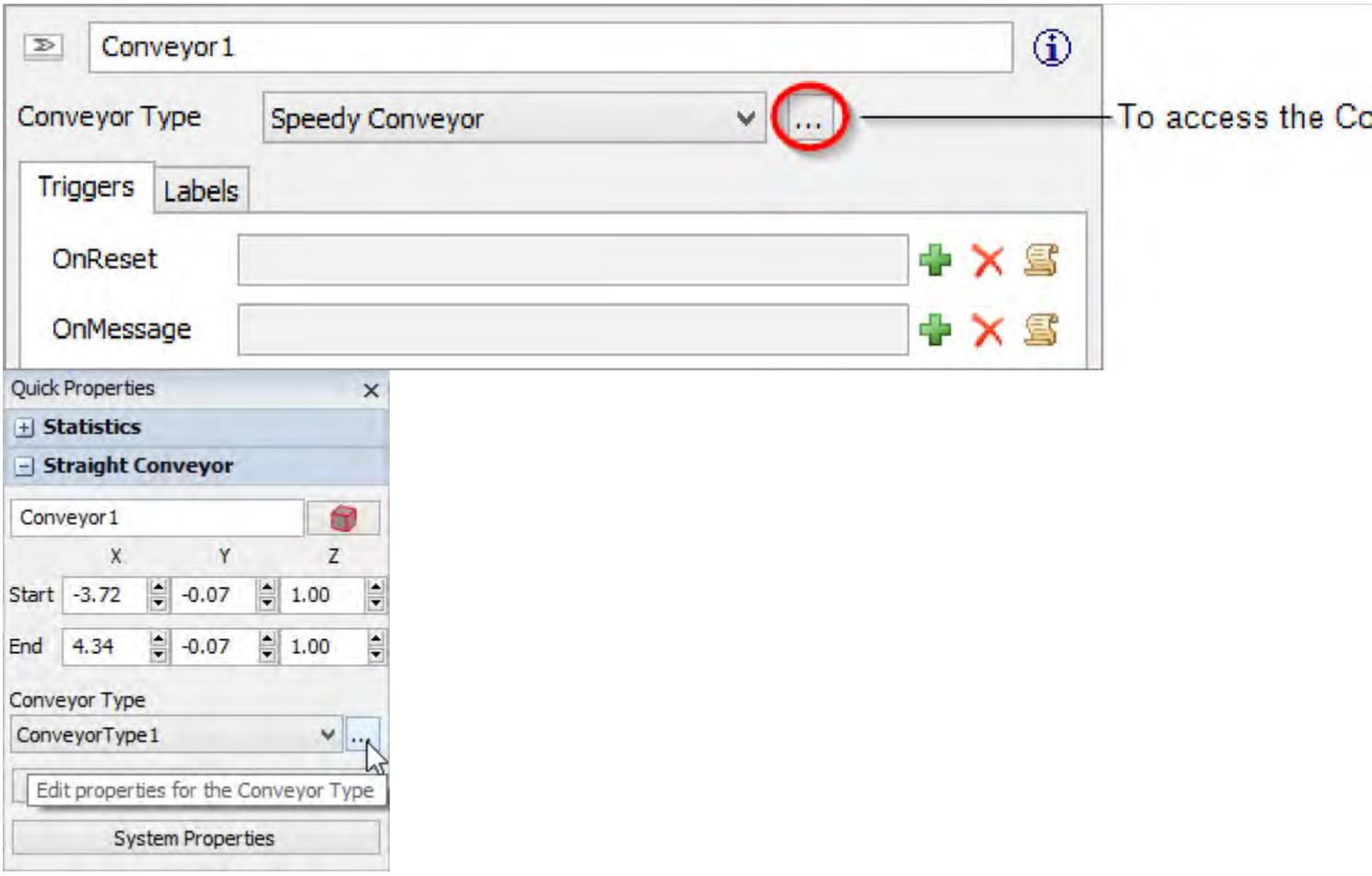
Using Quick Properties

To access the Type Properties dialog box through Quick Properties:

1. Click a conveyor object in your simulation model. You should click the kind of object for which you want to create or edit a Type. For example, if you want to create or edit a Conveyor Type, click a Straight or Curved Conveyor to highlight it. If you want to create/edit a Photo Eye Type, click a Photo Eye to highlight it, etc.

2. In the Quick Properties pane under the group for that particular object, look for a pull-down menu labeled Type, as shown in the image below. The full label for this pull-down menu is somewhat dynamic based on the kind of object you highlighted. For example, if you clicked a Straight or Curved Conveyor, the pull-down menu would be labeled Conveyor Type. If you clicked a Photo Eye, the pulldown menu would be labeled Photo Eye Type, etc.

3. Next to the Type menu, click the ... button to open the Type Properties dialog box for the object you've highlighted. From here you can create or edit the settings for this kind of conveyor object.



Using the Object's Properties Dialog Box

To access the Type Properties dialog box through the object's Properties dialog box:

1. Double-click a conveyor object in your simulation model to bring up the Properties dialog box.

2. Near the top of the dialog box you'll see a text box with a label indicating the object's current type. Underneath the label, look for a pull-down menu labeled Type, as shown in the image below. The full label for this pull-down menu is somewhat dynamic based on the kind of object you highlighted. For example, if you clicked a Straight or Curved Conveyor, the pull-down menu would be labeled Conveyor Type. If you clicked a Photo Eye, the pull-down menu would be labeled Photo Eye Type, etc.

3. Next to the Type menu, click the ... button to open the Type Properties dialog box for the object you've highlighted. From here you can create or edit the settings for this kind of conveyor object.

Conveyor System Settings

This section will provide more information about how to change the settings for the general Conveyor System. Before reading this section, you should be familiar with the purpose and basic functionality of the Conveyor System tool. See the following sections for more information:

- Introduction to the Conveyor System Tool
- Accessing Type Properties
- Managing Conveyor System Types

The Conveyor System Settings properties window has two tabs:

- Default Types - settings that determine the default Types that are assigned to conveyor objects when they are first added to the simulation model.
- General - settings for the visual appearance of conveyor objects within the model.

Each of the different settings available on each tab is explained in the following sections.

Conveyor System Settings vs. Conveyor Type Settings
Conveyor System Settings should not be confused with Conveyor Type Settings, which are the settings for Straight or Curved Conveyors.

Accessing Conveyor System Settings

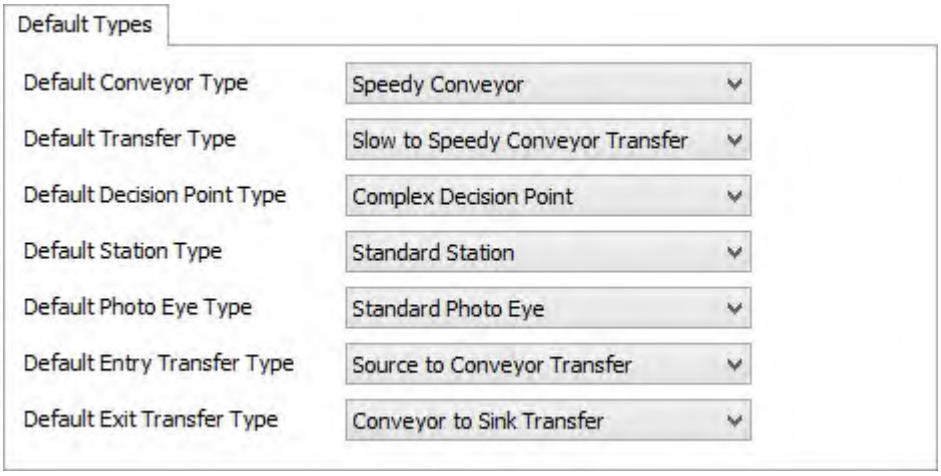
To access the Conveyor System Settings dialog box, you must first ensure that there is at least one Straight or Curved Conveyor in your model. Otherwise, the Conveyor System tool will not appear in your toolbox. Once you have a conveyor in your model:

1. Click View on the main menu, then click Toolbox to open the Toolbox. Alternatively, you could access the Toolbox by clicking the Tools button on the toolbar.

2. You'll see several different tools listed in the toolbox. Double-click Conveyor System to open the Conveyor System Settings dialog box.

The Default Types Tab

The following image shows the various settings available in the Default Types tab:



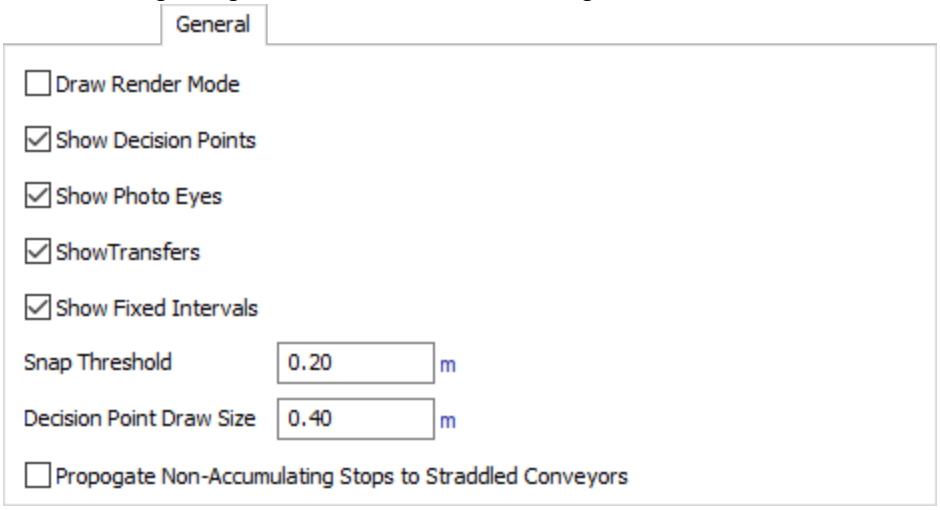
You'll notice there are several fields with pull-down menus. From each pull-down menu you can select the default Types that are assigned to various conveyor objects when they are first added to the simulation model.

For more information about the Type Properties dialog box and different settings for each object, see:

- Conveyor Types
- Transfer Types
- Decision Point Types
- Station Types
- Photo Eye Types
- Entry Transfer Types
- Exit Transfer Types

The General Tab

The following image shows the various settings available in the General tab:



The available settings are:

- Draw Render Mode - When checked, Straight or Curved Conveyors will visually appear to have rollers (for accumulating conveyors) or belts (for non-accumulating conveyors). See Conveyor Type Settings - The Visual Tab for more information about simulating roller conveyors.

- Show Decision Points - When checked, Decision Points will appear as a diamond icon on a conveyor.

- Show Photo Eyes - When checked, Photo Eyes will appear as a green or red line across a conveyor.

- Show Transfers - When checked, Transfers will appear as a square icon on a conveyor.

- Show Fixed Intervals - When checked, conveyors will show a line on a conveyor indicating where the dogs are in a Power and Free conveyor system. See Conveyor Types - Power and Free Settings for more information.

- Snap Threshold - Conveyors will snap and join together when they are within range of the value indicated in this field.

- Decision Point Draw Size - Determines what size Decision Points should be drawn visually.
- Propogate Non-Accumulating Stops to Straddled Conveyors - If checked, stopping/blocking items that are straddling multiple non-accumulating conveyors will also stop all upstream conveyors they are straddling (instead of just the conveyor that is driving the item's speed).

Units of Measurement

Many of the fields in the Conveyor Type Properties dialog box have editable Unit of Measurement fields. See Adjusting Units of Measurement for more information.

Managing Conveyor System Types

This section will provide more information about how to add, rename, and delete Types using the Conveyor System Tool. Before reading this section, you should be familiar with the purpose and functionality of the Conveyor System tool. See Introduction to the Conveyor System Tool for more information.

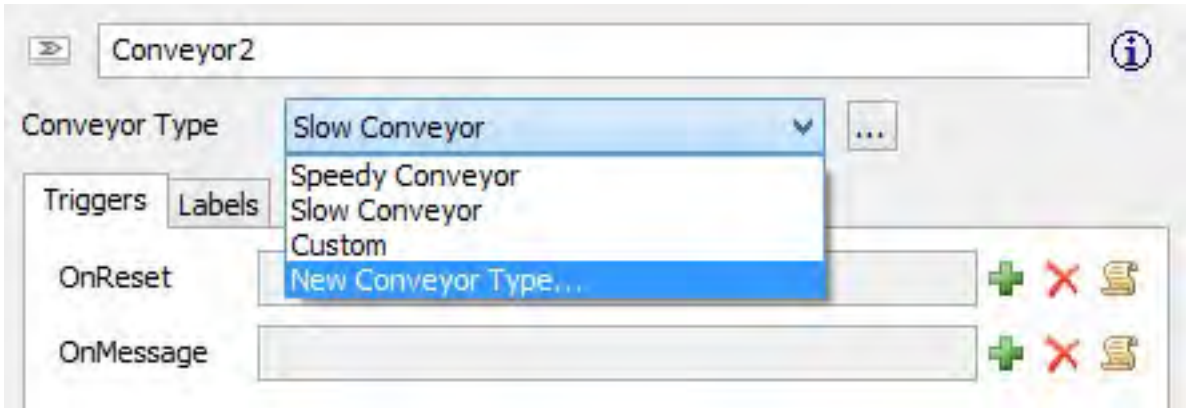
Adding and Renaming New Conveyor System Types

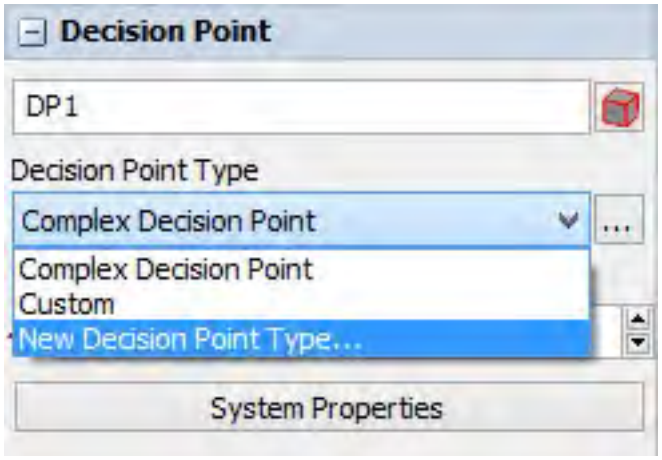
You can add a new Type to your Conveyor System using Quick Properties or the Properties dialog box (see Accessing Type Properties for information on how to open the tool using these methods):

1. In either Quick Properties or the Properties dialog box, look for a pull-down menu labeled Type. The full label for this pull-down menu is somewhat dynamic based on the kind of object you highlighted. For example, if you clicked a Straight or Curved Conveyor, the pull-down menu would be labeled Conveyor Type. If you clicked a Photo Eye, the pull-down menu would be labeled Photo Eye Type, etc.

2. When you open the Type menu, you will see a list of all the Types that have been added to the system so far. From the Type menu, select New [Object] Type... . Again, the specific label for this menu item is dynamic based on what kind of object you currently have highlighted, as shown in the following images.

Adding a New Conveyor Type





Decision Point Type

Adding a New

3. The Type Properties dialog box for the kind of object you've highlighted will now appear. At the top of the dialog box you will see a text box containing the name of the new Type. By default, the new Type will be named something such as *ConveyorType2* or *PEType1*, etc. You can edit the text box to give it any name you prefer.

4. At this point, you can edit any of the settings for that particular kind of object. For information about the setting for each kind of object see:

- Conveyor Types
- Transfer Types
- Decision Point Types
- Station Types
- Photo Eye Types
- Entry Transfer Types
- Exit Transfer Types

Assigning Conveyor System Types to New Objects

Once you've created the custom settings for each Type using the Conveyor System Tool, you can easily export the settings to any new conveyor objects or transfers in your simulation model. To assign a Type to an object or transfer:

1. In either Quick Properties or the Properties dialog box, look for a pull-down menu labeled Type. The full label for this pull-down menu is somewhat dynamic based on the kind of object you highlighted. For example, if you clicked a Straight or Curved Conveyor, the pull-down menu would be labeled Conveyor Type. If you clicked a Photo Eye, the pull-down menu would be labeled Photo Eye Type, etc.

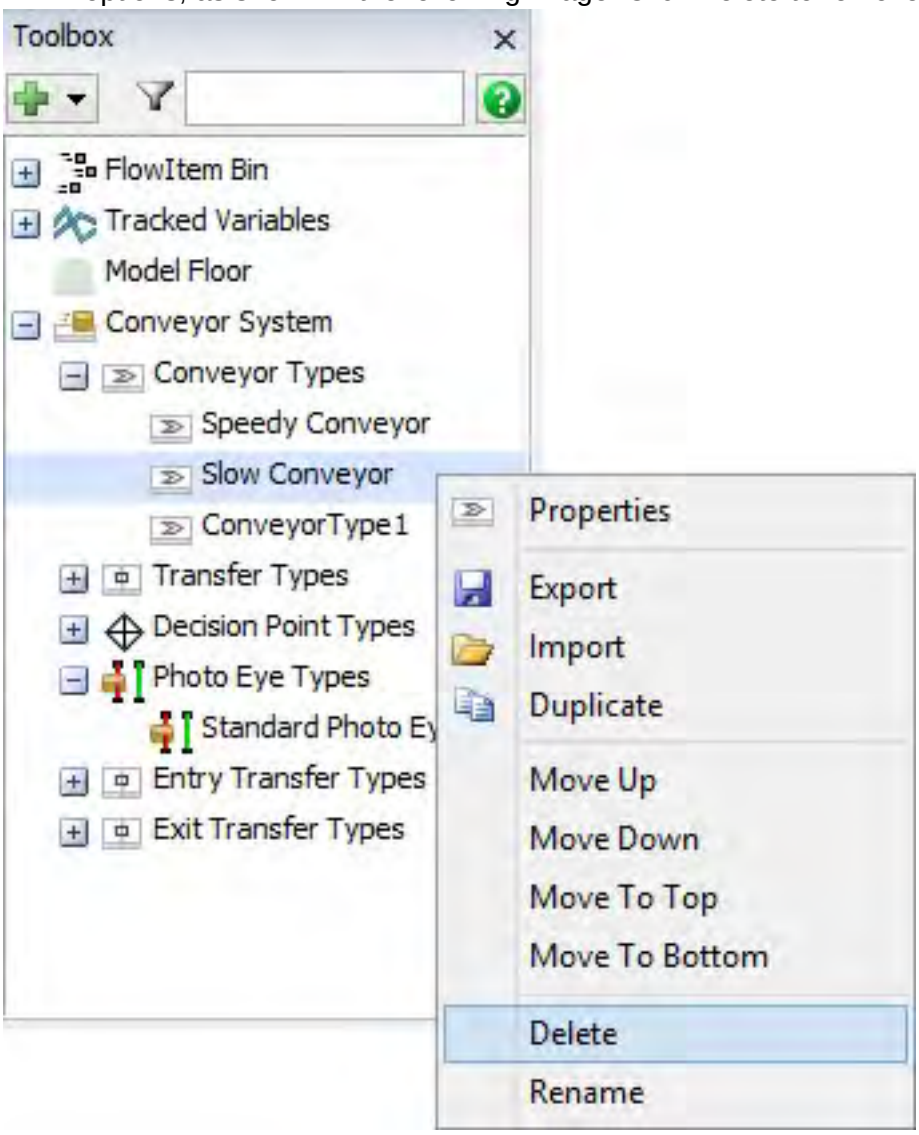
2. When you open the Type menu, you will see a list of all the Types that have been added to the system so far. From the Type menu, select the Type that should be assigned to this object.

Deleting Conveyor System Types

To delete a Conveyor System Type:

1. On the View menu, click Toolbox to open the Conveyor System tool.

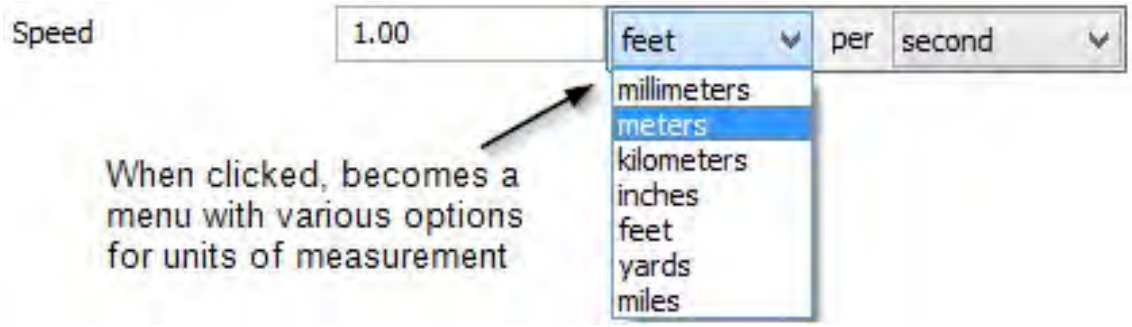
2. In the Conveyor System tool, right-click on the Type you want to delete. A menu will open with several options, as shown in the following image. Click Delete to remove this Type from the Conveyor System.



Adjusting Units of Measurement

Many of the Type Properties dialog boxes have fields that allow you to customize units of measurement. These fields have blue letters indicating units of measurement, as shown in the following image:

When you click on the blue letters, a menu field will open that will allow you to change the units of measurement for that setting.



Conveyor Type Settings

Conveyor Types are global settings that you can import to any Straight or Curved Conveyor object in your simulation model.

This section will explain the settings that you can customize when you create or edit Conveyor Types using the Conveyor System tool. Before reading this section, you should be familiar with the purpose and basic functionality of the Conveyor System tool. See the following sections for more information:

- Introduction to the Conveyor System Tool
- Accessing Type Properties
- Managing Conveyor System Types

You can adjust the global settings for different Conveyor Types using the Conveyor Type Properties dialog box. This dialog box has two tabs:

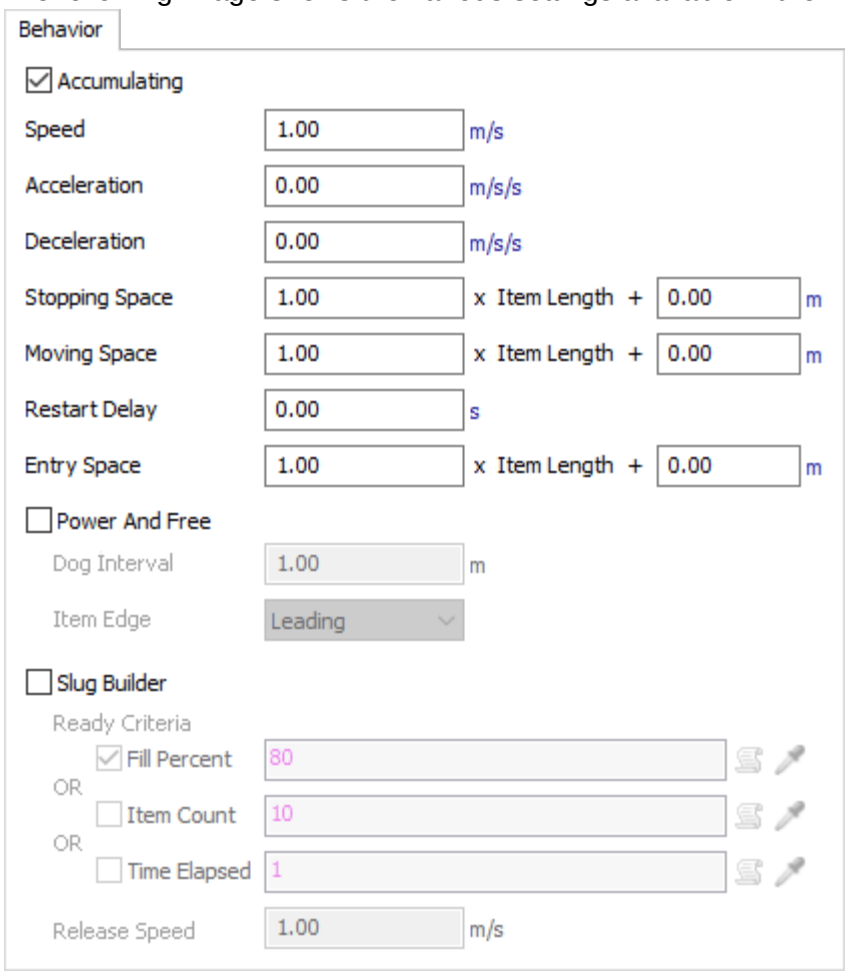
- Behavior - settings that affect the behavior for all conveyors that are assigned this Type.
- Visual - settings that affect the appearance of all conveyors that are assigned this Type.

Units of Measurement

Many of the fields in the Conveyor Type Properties dialog box have editable Unit of Measurement fields. See Adjusting Units of Measurement for more information.

The Behavior Tab

The following image shows the various settings available in the Behavior tab:



Conveyor Movement Settings

The first four settings in the Behavior tab control conveyor movement:

- Accumulating - If this box is checked, items will accumulate on the conveyor, meaning they will collect together along the conveyor, similar to a roller conveyor. If this check box is cleared, items will move in lock-step with each other, like on a belt conveyor. By default, this check box is cleared.
- Speed - The default speed of the conveyor.
- Acceleration - The conveyor's speed as it moves when it first starts or when it changes from a slower speed to a faster speed.
- Deceleration - The conveyor's speed as it slows down to a stop or when it changes from a faster speed to a slower speed.

Acceleration and Deceleration

Acceleration and deceleration are only applied when the conveyor's speed changes. They are not used when simulating per-item motion such as when an item is blocked by a downstream conveyor or by accumulation.

Spacing Settings

The next three settings in the Behavior tab control the spacing of items as they move along the conveyor:

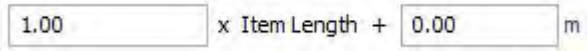
- Stopping Space - The minimum amount of space between items while they travel along the conveyor of while they are accumulating. If the space between items is less than the specified space, the item will stop moving.
- Moving Space - If items are stopped along a conveyor, this is the amount of space needed between items before they can resume movement.
- Restart Delay - If you define a non-zero restart delay time, after items are stopped on a conveyor, this additional time will be applied before starting the items again after their move space has been cleared.
- Entry Space - The amount of space that must be clear before an item will enter a conveyor.

Spacing Rule: Stopping Space ≤ Moving Space

A conveyor's Moving Space must be greater than or equal to its Stopping Space. If the Moving Space is equal to the Stopping Space, items will immediately follow behind items they accumulate against. If the Moving Space is greater than Stopping Space (or if the Restart Delay is greater than zero), items will stop in place when accumulating, and then resume when the Moving Space is cleared (and the Restart Delay timer has expired). This results in a caterpillar-like accumulation effect.

Spacing Rule: Stopping Space ≤ Entry Space

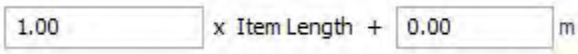
A conveyor's Entry Space must be greater than or equal to its Stopping Space. Each spacing setting has two different fields, as illustrated in the image below:



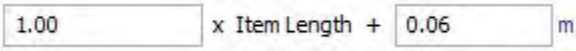
The first editable number field, the Item Percent field, allows you to a percentage of space multiplied by the item's length. 1.00 is equivalent to 100%, or in other words the full length of the item. The value 0.5 would be equivalent to 50%, half the item's length. And 2.00 would be equivalent to 200%, twice the item's length.

The second editable number field, the Additional Spacing field, allows you to add an additional amount of space based on the units of measurements you specify. (See Adjusting Units of Measurement for more information.)

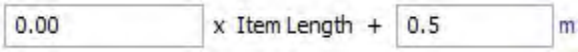
Using these two fields, you can create a dynamic set of spacing rules that is either relative or absolute to the item's length. The following images show examples and explanations of different spacing settings:



The default of 1.00 x Item Length + 0.00m defines spacing based purely on item len gth.



Defining a spacing value of 1.00 x Item Length + 0.06m adds a gap of 6 centimeters between each item.



A spacing of 0.00 x Item Length + 0.5m defines a 0.5 meter spacing, independent of item length.

Power and Free Settings

The next group of settings is designed for use in simulating Power and Free conveyor systems. In these systems, dogs travel at fixed intervals along a looping chain. These dogs pick up carriers in the system as they pass them. In simulation terms, it is similar to having a moving space greater than stopping space (there is a caterpillar-like accumulation effect), except that the point at which an item can move on the conveyor is defined by the location of the next passing dog, instead of by the space between it and the item ahead of it. (See Power and Free Systems for more information.) The available settings are:

- Power and Free - If checked, items can only move on the conveyor at fixed intervals, at the points where simulated dogs pass the item's position.
- Dog Interval - The spacing between each dog on the Power and Free chain. If you want to simulate a repeating irregular pattern of gaps between dogs, you can enter in a series of custom numbers separated by commas.
- Item Edge - Defines the associated item edge to be picked up by a dog.

Initial Dog Positions

When Power and Free settings are enabled, by default the conveyor will assign the dog positions based on the first item that enters the conveyor. If you need more control over defining dog positions across multiple conveyors, you can use a Motor.

Carriers

Power and Free conveyors do not simulate carriers automatically. If you need to simulate carriers as limited resources in the simulation, you can simulate them explicitly by having items that are picked up by the dogs represent the carriers, and then use Decision Points to move other items into and out of those carrier items, representing the actual loads that are being moved by the carriers.

Slug Building Settings

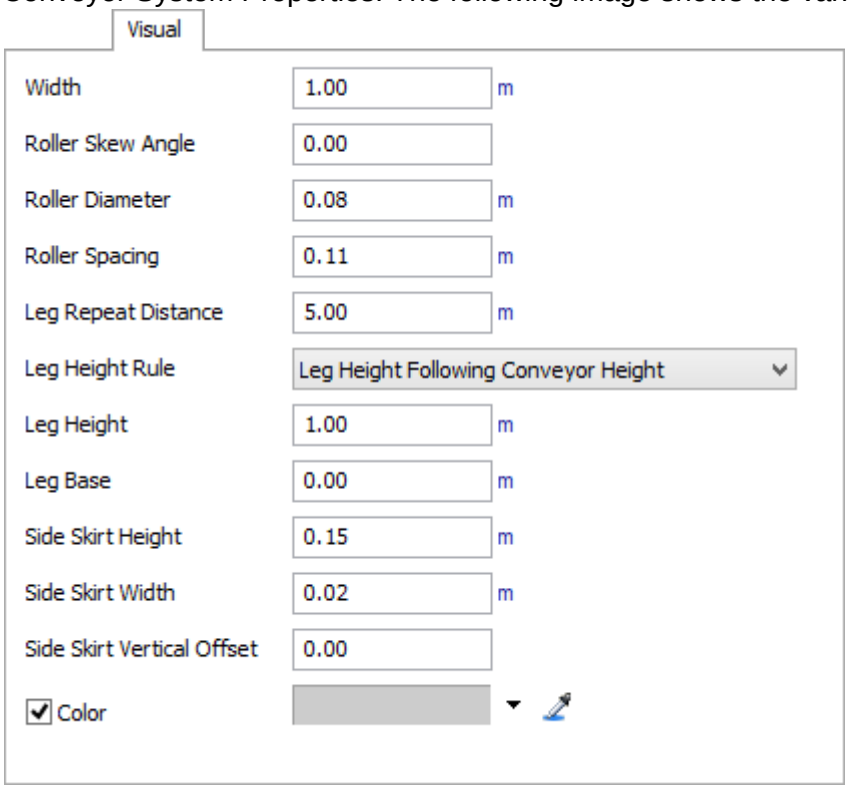
The last group of settings allow conveyors to build slugs. Conveyors can build and release slugs as part of a saw-tooth merge. See Merging for more information on how to build a saw-tooth merge. The Slug Building settings are as follows:

- Slug Builder - Enables slug building when checked.

- Ready Criteria - Defines when a slug lane can be considered ready for release. One or more of three options should be chosen. If any of the checked items is fulfilled, the slug will be marked as ready for release. The slug may continue to build while waiting for release. If the conveyor is not connected to a Merge Controller, it will release slugs as soon as they are ready for release.
    - Fill Percent - Select this box and enter a value to define the percentage of the full conveyor length that is required for the slug to be ready for release.
    - Item Count - Select this box and enter a value to define the number of items in a slug required to be ready for release.
    - Time Elapsed - Select this box and enter a value to define a maximum elapsed slug-build time. The time starts when the first item is added to the slug. If the slug is still not ready when the timer finishes, the slug will be made ready for release.
- Release Speed - The speed at which a slug will be released.

The Visual Tab

Visual settings mostly define how conveyors look when drawing in render mode. Render mode can be enabled in Conveyor System Properties. The following image shows the various settings available in the Visual tab:



The Visual settings are as follows:

- Width - The width of the conveyor.
- Roller Skew Angle - The angle, in degrees, at which the rollers are skewed. If non-zero, items will convey to one side of the conveyor as that travel down it.

- Roller Diameter - The diameter to draw rollers when in render mode. Only used in render mode.
- Roller Spacing - The distance from the center of one roller to the center of the next roller. Should be greater than Roller Diameter. Only used in render mode.
- Leg Repeat Distance - The distance between each leg of the conveyor. Only used in render mode.
- Leg Height Rule - Defines how the height of each leg will be determined. Only used in render mode.
    - Leg Height Following Conveyor Height - Legs will be a set height no matter how the conveyor rises or falls. If the conveyor rises, leg bases will rise with it.
    - Leg Height from Lowest Point on Conveyor - Leg height is determined from the lowest point on the conveyor. For example, if the conveyor plane goes from z = 1.00 to z = 5.00, and Leg Height is defined as 1.00, the leg base will be at z = 0.00 all along the conveyor (lowest point (1.0) minus leg height (1.0)).
    - Leg Base Relative to Model - Legs will extend to a fixed model z position, defined by Leg Base, no matter how high the conveyor is.
- Leg Height - The conveyor leg height. Only used in render mode when Leg Height Rule equals Leg Height Following Conveyor Height or Leg Height From Lowest Point on Conveyor.
- Leg Base - The leg base z-point. Only used in render mode when Leg Height Rule equals Leg Base Relative to Model.
- Side Skirt Height - The height of the conveyor's side skirt.
- Side Skirt Width - The width of the side skirt. Only used in render mode.
- Side Skirt Vertical Offset - Defines the position of the side skirt relative to the conveyor plane. This is defined as a ratio of the Side Skirt Height. Usually the value will be between -0.5 and 0.5. For a value of -0.5, the side skirt will extend down from the conveyor plane. For a value of 0.0, the side skirt's center will follow the conveyor plane. For a value of 0.5, the side skirt will extend up from the conveyor plane.
- Color - Defines the color by which conveyors of this type will be drawn. Only used for non-render mode.

Transfer Type Settings

Transfers are connections between conveyor objects, represented by a white box on a conveyor in the simulation model. Transfer Types are global settings that you can import to any Transfer that connects one conveyor to another conveyor. Conveyor-to-conveyor transfers are different from Entry Transfer Types or Exit Transfer Types, which are for Transfers to non-conveyor objects.

This section will explain the settings that you can customize when you create or edit Transfer Types using the Conveyor System tool. Before reading this section, you should be familiar with the purpose and basic functionality of the Conveyor System tool. See the following sections for more information:

- Introduction to the Conveyor System Tool
- Accessing Type Properties
- Managing Conveyor System Types

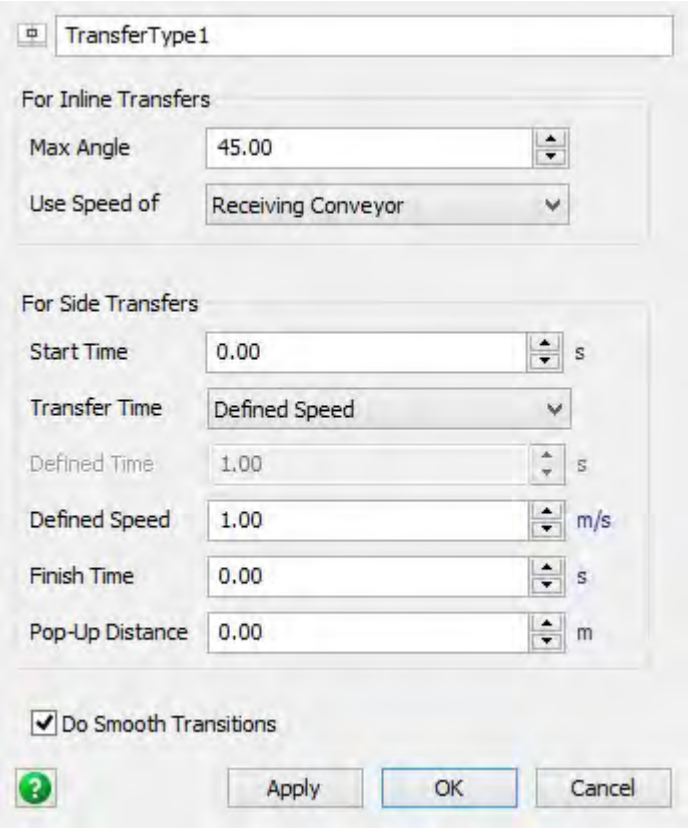
Before using the Transfer Type Properties dialog box, you need to be aware of the differences between a Side Transfer and an Inline Transfer:

- Inline Transfer - A transfer in which the item's orientation will remain unchanged as it moves across the transfer. Inline Transfers start when the item's leading edge reaches the transfer point and finish when the item's trailing edge passes across the transfer point.

- Side Transfer - A transfer in which the item's orientation will change during the transfer. For a diverting side transfer, the transfer starts when the item's center reaches the transfer point, and finishes when the items is completely on the divert conveyor. For a merging side transfer, the transfer starts when the transfer reaches the end of the upstream conveyor, and ends when it is completely transferred onto the merge conveyor.

You will need to define the angles at which transferring conveyors will be considered an Inline Transfer or a Side Transfer, as discussed below.

The following image shows the various settings available in the Transfer Type Properties dialog box:



Units of Measurement

Many of the fields in the Transfer Type Properties dialog box have editable Unit of Measurement fields. See Adjusting Units of Measurement for more information.

Inline Transfer Settings

The Inline Transfer group box has the following settings:

- Max Angle - This setting determines the maximum angle the incoming conveyor must be relative to the receiving conveyor in order to be considered an Inline Transfer. For example, if you enter 45 as and the incoming conveyor is greater than a 45 degree angle relative to the receiving conveyor, it will act like a Side Transfer rather than an Inline Transfer.
- Use Speed of - Defines which conveyor's speed the item will take as it crosses the transfer.
    - Receiving Conveyor - The item will take the speed of the destination conveyor. o Sending Conveyor - The item will take the speed of the origin conveyor. o Faster Conveyor - The item will take the speed of the faster conveyor. o Slower Conveyor - The item will take the speed of the slower conveyor.

- Conveyor Under Item Center - The item will take the speed of the origin conveyor for the first half of its length, and then take the speed of the destination conveyor for the second half of its length.

Side Transfer Settings

When an item moves over a side transfer, it will go through the following steps.

1. If the transfer type has a non-zero Start Time, the item will stop and wait for the Start Time at the point where the transfer begins.

2. The item will move across the transfer. Use the Transfer Time field to define the time it will take.

    - Defined Speed - If selected, the item will use the Defined Speed to move across the transfer. o Defined Time - If selected, the item will use the Defined Time to move across the transfer.
    - Motion-Aligned Conveyor Speed - If selected, the item will take the speed of the conveyor whose motion is aligned with the transfer motion of the item. For a diverting side transfer, this is the speed of the destination conveyor, or the conveyor that the item is being diverted to. For a merging side transfer, it is the speed of the origin conveyor, or the conveyor pushing the item onto the destination merge conveyor.

3. If the transfer type has a non-zero Finish Time, the item will stop and wait for the Finish Time at the point where the transfer ends.

Pop-Up Distance - If non-zero, the item will move this distance vertically during the start and finish times. It will move up for the Start Time, and back down for the Finish Time.

Additional Settings

Do Smooth Transitions - Check this box if you want an item to gradually change its orientation and position so that it matches the angle of the receiving conveyor as it moves across the transfer. This is most applicable for inline transfers that are not exactly inline (they have a transfer angle not equal to zero), for Inline Transfers where the location of the transfer is not in the center of the conveyor, or for Side Transfers where the transfer is not an exact 90-degree transfer.

Decision Point Type Settings

Decision Point Types are global settings that you can import to any Decision Point. This section will explain the settings that you can customize when you create or edit Decision Point Types using the Conveyor System tool.

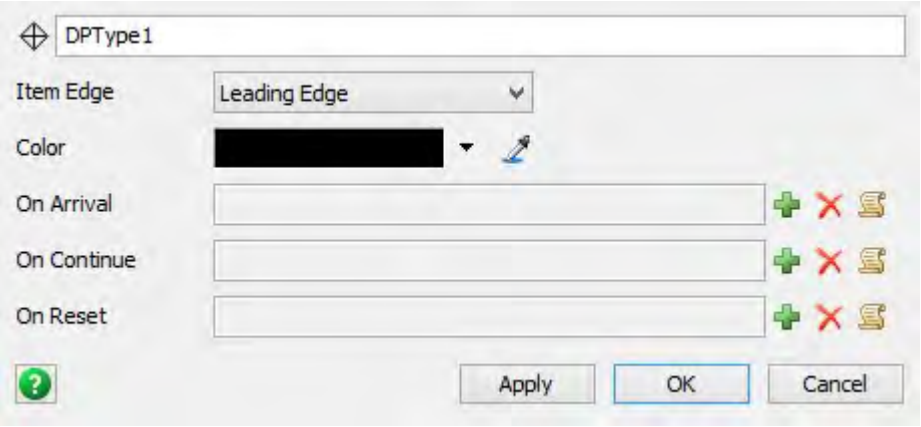
Before reading this section, you should be familiar with the purpose and basic functionality of the Conveyor System tool. See the following sections for more information:

- Introduction to the Conveyor System Tool
- Accessing Type Properties
- Managing Conveyor System Types

Pick List Behaviors

For more information on how to use some of the Pick List Behaviors for Decision Points and Photo Eyes, see Decision Point and Photo Eye Behaviors.

The following image shows the various settings available in the Decision Point Type Properties dialog box:



There are four available settings for Decision Points Types:

- Item Edge - The item edge that will trigger Decision Point events.
- Color - The color that Decision Points of this type will be drawn with.
- On Arrival - The Trigger that will fire when the defined item edge arrives at the Decision Point.
- On Continue - The Trigger that will fire when the defined item item edge continues through the Decision Point.
- On Reset - A Trigger that fires when the simulation model is reset. This can be used to reset state variables such as labels, etc., when the model is reset.

On Arrival vs. On Continue
Usually the OnContinue trigger will be fired immediately after the OnArrival. However, if you stop the item as part of the OnArrival, the OnContinue will not be fired until the item is resumed.

Photo Eye Types are global settings that you can import to any Photo Eye. This section will explain the settings that you can customize when you create or edit Photo Eye Types using the Conveyor System tool. This can be one of a global list of Photo Eye Types, or it can be a custom Photo Eye Type associated only with one specific Photo Eye. The default Photo Eye Type is labeled Custom and is only assigned to one specific Photo Eye.

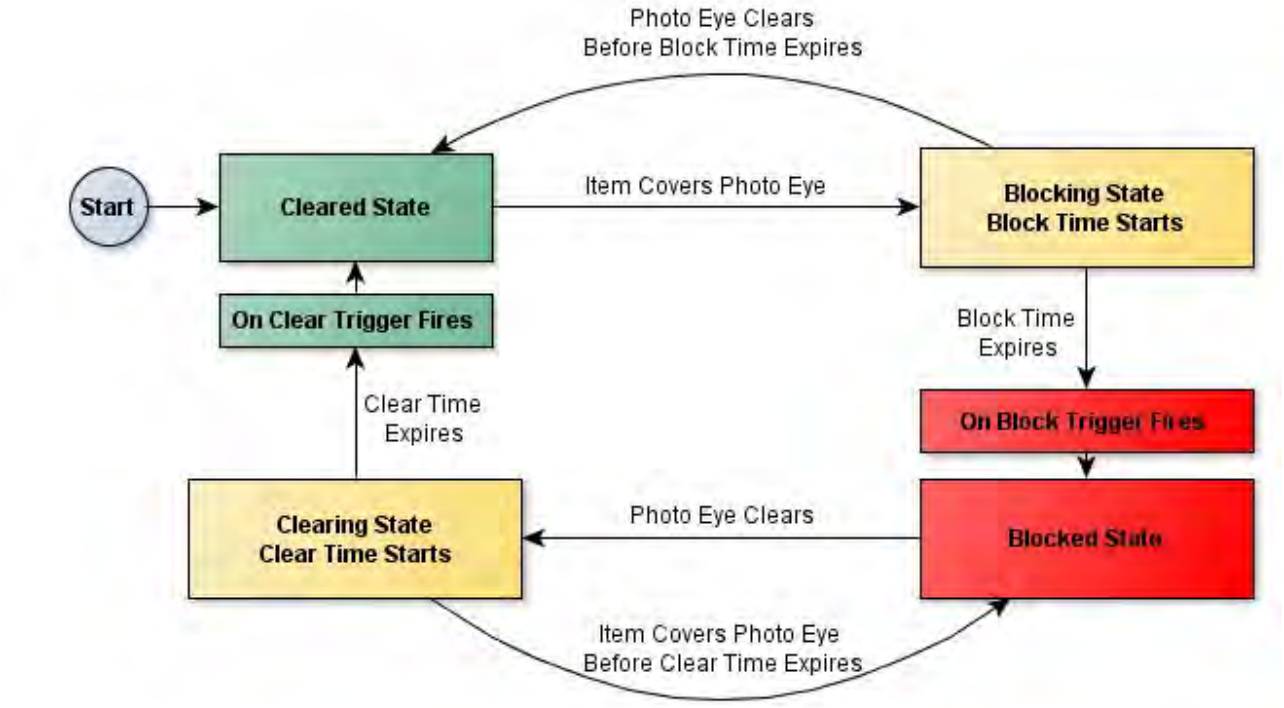
Before reading this section, you should be familiar with the purpose and basic functionality of the Conveyor System tool. See the following sections for more information:

- Introduction to the Conveyor System Tool
- Accessing Type Properties
- Managing Conveyor System Types

Pick List Behaviors

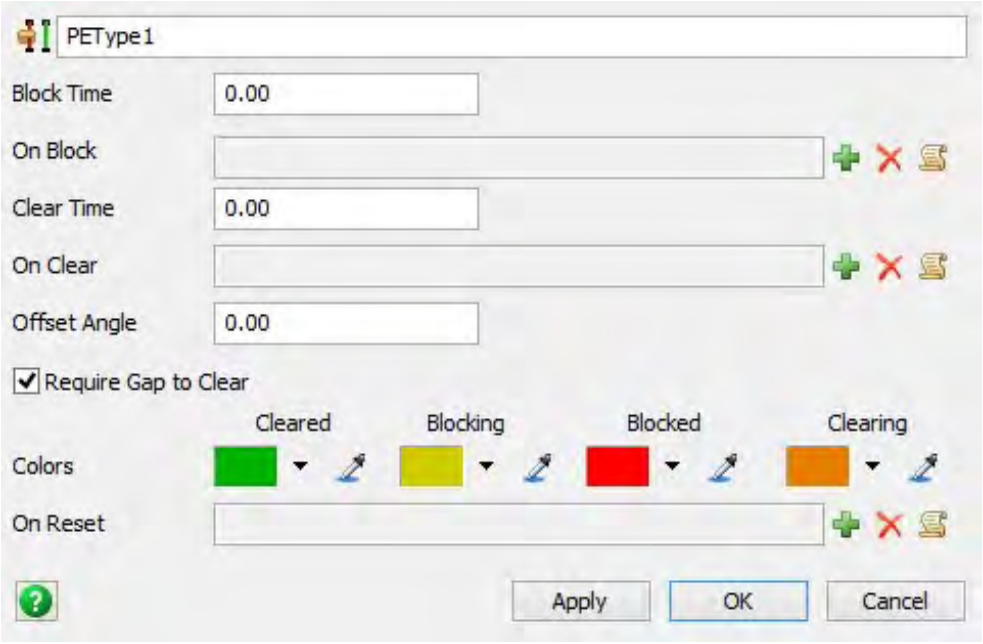
For more information on how to use some of the Pick List Behaviors for Decision Points and Photo Eyes, see Decision Point and Photo Eye Behaviors.

A Photo Eye can be in one of four states while it is interacting with items on a conveyor: Blocked, Cleared, Blocking and Clearing. Triggers fire when the Photo Eye goes into the Blocked and Cleared states. The following image illustrates the various Photo Eye states and triggers while interacting with items on a conveyor:



You can assign a Photo Eye custom behaviors or actions that it can perform while it is in either a Blocked or Cleared state. You can also determine how long a Photo Eye should be in a Blocked or Cleared state.

The following image shows the various settings available in the Photo Eye Type Properties dialog box:



There are six available settings and triggers for Photo Eye Types:

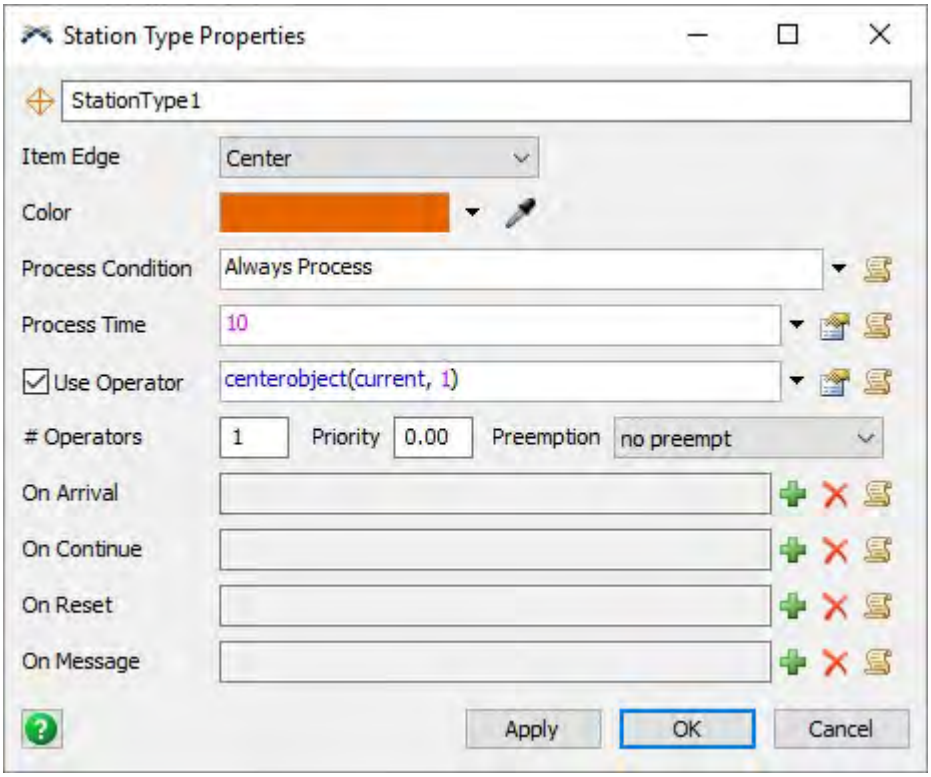
- Block Time - The amount of time an item must block a Photo Eye before it will change from a Cleared state to a Blocked state.
- On Block - The trigger logic that fires when the Photo Eye enters the Blocked state.
- Clear Time - The amount of time a Photo Eye must be clear from items before changing from a Blocked state to a Cleared state.
- On Clear - The trigger logic that fires when the Photo Eye enters the Cleared state (after having been Blocked).
- Offset Angle - The offset angle by which the Photo Eye beam crosses the conveyor plane. If 0, the beam crosses straight across the conveyor.
- Require Gap to Clear - If checked (default), there must be a gap between items in order to clear the Photo Eye. You might uncheck this box if you want the On Clear Trigger to fire on every item, no matter the spacing between items.
- Colors - Defines the colors by which the Photo Eye will be drawn for each of the states mentioned above.
- On Reset - A trigger that fires when the simulation model is reset. You can use this to reset labels or other data when the model is reset.

Station Types contain global settings that you can import to any Station. This section will explain the settings that you can customize when you create or edit Station Types using the Conveyor System tool.

Before reading this section, you should be familiar with the purpose and basic functionality of the Conveyor System tool. See the following sections for more information:

- Introduction to the Conveyor System Tool
- Accessing Type Properties
- Managing Conveyor System Types

The following image shows the various settings available in the Station Type Properties dialog box:



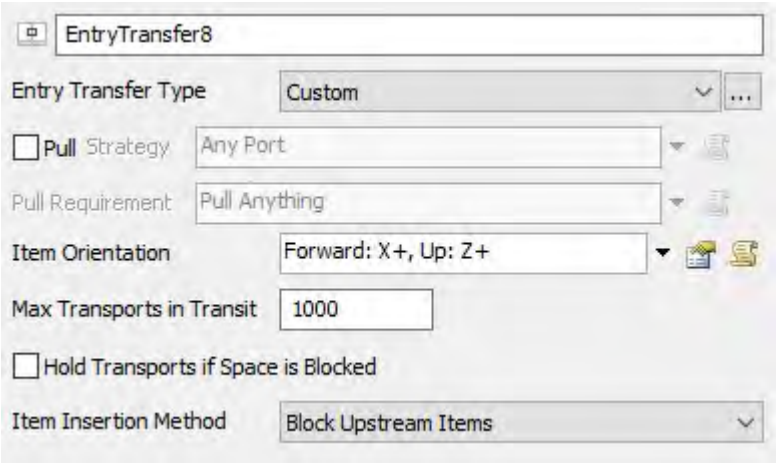
- Item Edge The item edge that will trigger station processing.
- Color The color that stations of this type will be drawn with.
- Process Condition Returns a value, 1 or 0, defining the condition by which a given item passing over the station should be processed. Return 1 to process the item. Return 0 to let the item pass without processing it.
- Process Time Defines the processing time for a given item at the station.
- Use Operator Defines settings for calling an operator to the station to process the item. Check the box and then define the reference to the operator or dispatcher to give the task to, as well as the # of Operators to request, and the Priority and Preemption values of the request task sequence.
- On Arrival The trigger that will fire when the defined item edge arrives at the station.
- On Continue The trigger that will fire when the defined item item edge continues through the station.
- On Reset A trigger that fires when the simulation model is reset. This can be used to reset state variables such as labels, etc., when the model is reset.
- On Message A trigger that fires when the station is sent a message.

Entry Transfers are connections going from a non-conveyor object (such as a Source or other Fixed Resource) to a conveyor object. They are represented by a white box on a conveyor in the simulation model. Entry Transfer Types are settings that you can import to any Entry Transfer. Be aware that Entry Transfers are different from Transfers, (which connect conveyors to other conveyors) and Exit Transfers (which are connections from a conveyor object to a non-conveyor object).

Before reading this section, you should be familiar with the purpose and basic functionality of the Conveyor System tool. See the following sections for more information:

- Introduction to the Conveyor System Tool
- Accessing Type Properties
- Managing Conveyor System Types

The following image shows the various settings available in the Entry Transfer Type Properties dialog box:



The Entry Transfer Type dialog box has the following settings:

- Pull Strategy / Pull Requirement - Define the entry transfer's pull strategy and pull requirement. See Fixed Resource Concepts for more information.

- Item Orientation - Defines what part of the item is considered the front of the item as it enters the conveyor.

- Max Transports in Transit - Defines the maximum number of items can be "in transit" to the Entry Transfer at a time, i.e. the max number of TaskExecuter requests that can be simultaneously made to transport items into the Entry Transfer.

- Hold Transports if Space is Blocked - If checked, TaskExecuters dropping the item off to the Entry Transfer must wait until the entry point is available before finishing unloading the item onto the conveyor.

- Item Insertion Method - Determines how items entering the conveyor through the Entry Transfer will be inserted onto the conveyor. Options are:

    - Simple - Items must wait until the entry location is clear. Items will be visually placed at the side of the conveyor until the space is available.
    - Block Upstream Items - If the entry location is blocked when the item attempts to enter, upstream items will be stopped until the space is clear and the item can enter on.

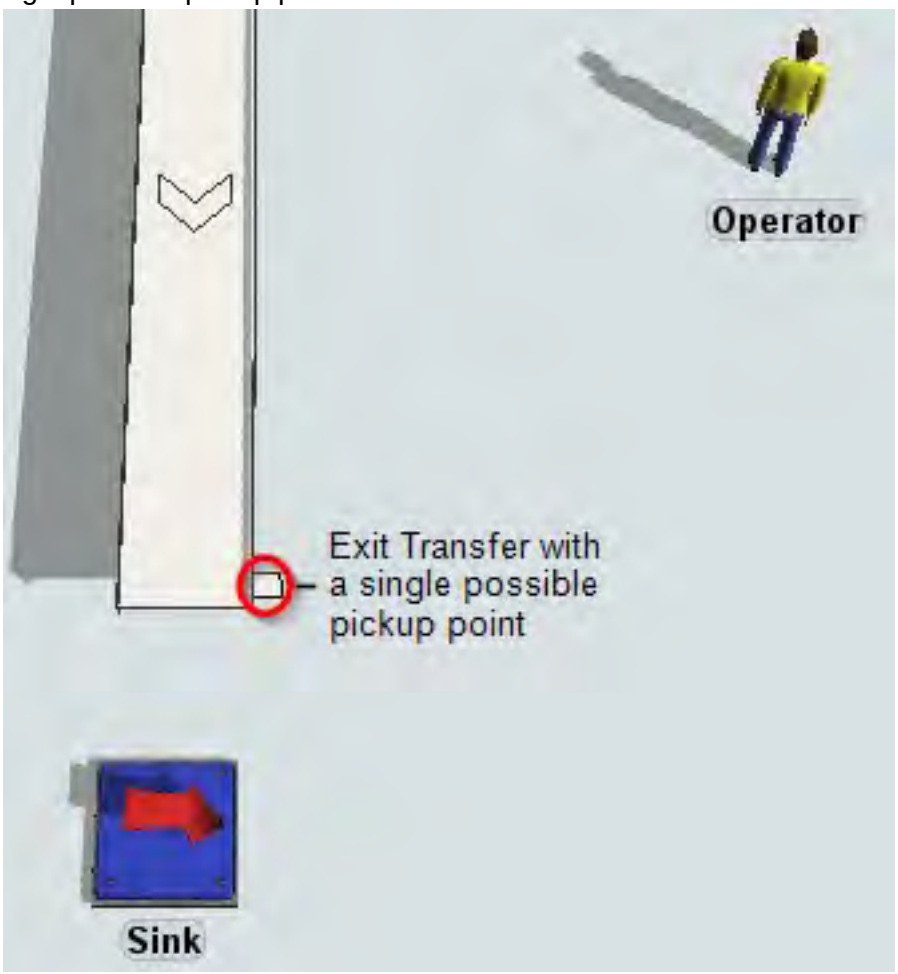
- Clear Available Space - If the entry location is blocked when the item attempts to enter, upstream items will be pushed backward until the required space is available. The time to perform the push-back is based on the items being pushed back at twice the conveyor speed. Note that the push-back time is only important if transports are held when space is blocked. In this case, the transports are held for the push-back time, and then they finish the unload. Note also that if this option is chosen, there should be enough "clear" space on the conveyor to push items back. Behavior is undefined if items must be pushed back off the conveyor, or if they are pushed past conveyor transfers. If they are pushed back over Decision Points or Photo Eyes, the Triggers for those objects will NOT be fired again.

Exit Transfers are connections going from a conveyor to a non-conveyor Fixed Resource. They are represented by a white box on a conveyor in the simulation model. Exit Transfer Types are settings that you can import to any Exit Transfer. Be aware that Exit Transfers are different from Transfers (which connect conveyors to other conveyors) and Entry Transfers (which are connections from a non-conveyor object to a conveyor).

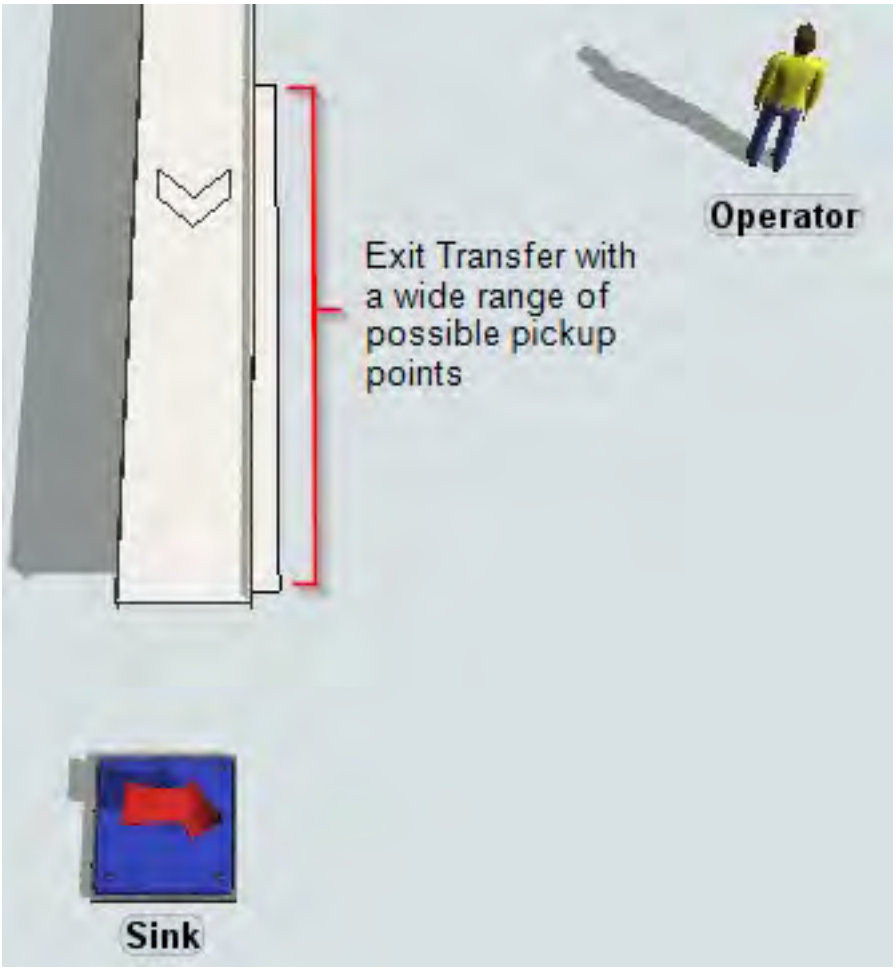
This section will explain the settings that you can customize when you create or edit Exit Transfer Types using the Conveyor System tool. Before reading this section, you should be familiar with the purpose and basic functionality of the Conveyor System tool. See the following sections for more information:

- Introduction to the Conveyor System Tool
- Accessing Type Properties
- Managing Conveyor System Types

Exit Transfers have the ability to send for a transport, or in other words a Task Executer, to pick up items and deliver them to another object. Task Executers can pick up items from a particular point on the conveyor or a range of possible points along the conveyor, as shown in the two following images. Exit Transfer with a single possible pickup point

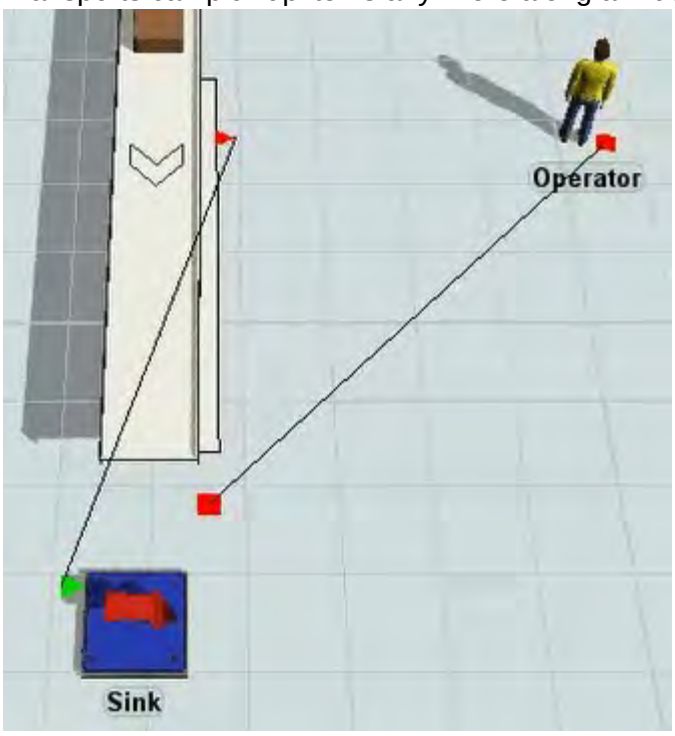


Exit Transfer with a wide range of possible pickup points



The Task Executor will begin traveling toward the Exit Transfer as soon as the center of an item arrives at the edge of the Exit Transfer on the conveyor. If the Exit Transfer has a wide range of possible pickup points, the Task Executer can pick it up anywhere along the Exit Transfer, as shown in the following animated image. You also have the option of allowing the Task Executer to use Continuous Pickup Point Prediction, which means that the Task Executer recheck the position of the item at intervals and alter its course to a different pickup point based on how the item is predicted to move along the conveyor.

Transports can pick up items anywhere along a wide Exit Transfer



These sections will be discussed in the following sections, which explain how to use Exit Transfers in a simulation model generally.

Moving or Expanding the Range of Possible Pickup Points

To expand the range of possible pickup points that a Task Executer can use to pick up items along a conveyor:

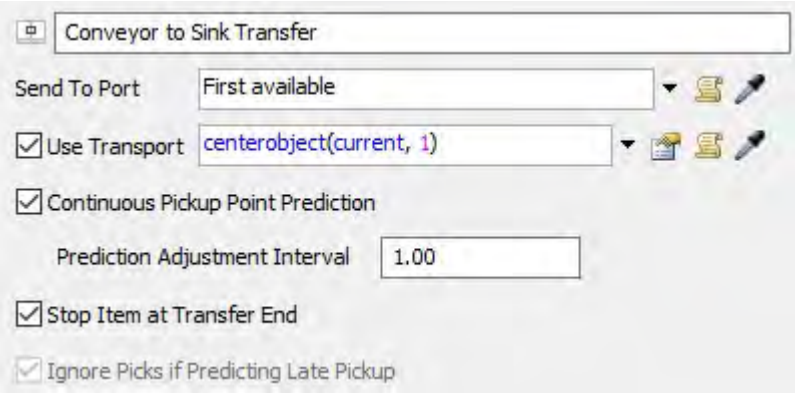
1. Create an input/output port connection (an 'A' connect) from a conveyor to a Fixed Resource, such as a sink. (See Connecting Objects.)
2. An Exit Transfer will appear at the end of the conveyor. It will look like a white box with a port connected to the Fixed Resource.
3. Click on the Exit Transfer and drag it to the desired position on the conveyor. NOTE: You can move the Exit Transfer to any position on the conveyor, including along its sides.
4. Two red sizing arrows appear on both sides of the Exit Transfer when you click on it. Click either red sizing arrow and drag it away from the Exit Transfer until it reaches the desired size (range).

Using a Transport to Pick Up Items from a Conveyor

To use a transport, or in other words a Task Executer, to pick up items and deliver them to another object:

1. Create a center port connection (an 'S' connect) from the Exit Transfer to a Task Executer, such as an Operator. (See Connecting Objects.)

2. Double-click the Exit Transfer to open the Exit Transfer Properties dialog box, as shown in the following image.



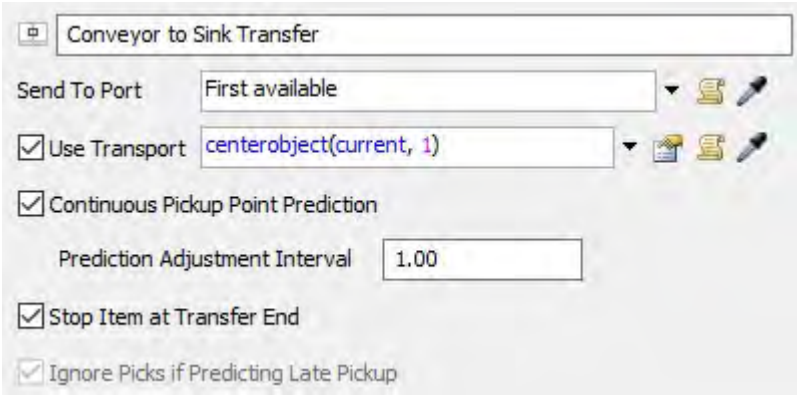
3. Check the Use Transport check box. By default, the Exit Transfer will request a transport from the Task Executer because they share a center port connection. You can adjust these settings if desired.

4. A Task Executer can modify its course while it is traveling to the item pickup point if there is a wide range of possible pickup points. The Task Executer can estimate and change where it can pick up an item based on the item's movement along the conveyor. To use this feature, check the Continuous Pickup Point Prediction check box.

5. You can adjust how frequently the Task Executer updates its predictions by changing the value in the Prediction Adjustment Interval text box. The value you enter corresponds to the units of measurement for time that you used when you first created your simulation model.

6. Click OK to save the settings. When you reset and run your model, the Task Executer will pick up the item from the conveyor and deliver it to a Fixed Resource.

Settings Reference



- Send To Port / Use Transport Defines Fixed Resource send-to logic. See Fixed Resource Concepts for more information.

Pickup Point Prediction

When an exit transfer has an expanded range along the conveyor and Use Transport is checked, Task Executers loading from the conveyor will use a pickup point prediction heuristic to determine where on the conveyor to "meet" the item to load it. This heuristic makes a best guess based on the current speed and position of the item, the distance from the Task Executer to the conveyor, and the max speed of the Task Executer. This may not be perfect, especially if there are other factors affecting the item's location, such as accumulation, or if acceleration/deceleration of the Task Executer make up a large portion of its travel.

- Continuous Pickup Point Prediction If checked, Task Executers who are picking up an item from the exit transfer will re-evaluate the estimated pick up point for the item at defined intervals.
- Prediction Adjustment Interval If Continuous Pickup Point Prediction is checked, this defines the time interval between pickup point prediction updates.
- Stop Item at Transfer End If checked, the item will be stopped if it reaches the end of the exit transfer's range before exiting. Then it will wait until it exits. If this is not checked, the item will be "unreleased" at the end of the exit transfer (it will no longer exit through the exit transfer) and move on.
- Ignore Picks if Predicting Late Pickup This is only valid if Use Transport is checked, Stop Item at Transfer End is not checked, and the exit transfer has an expanded range along the conveyor. If checked, Task Executers who predict the pickup point to be after the item has passed the end of the exit transfer will abort their transport task sequence.

The Straight and Curved Conveyors can simulate conveyor belts or roller conveyors. Straight and Curved Conveyors function slightly differently than some of the other objects in the FlexSim Library. For example, the process for adding them to a simulation model is slightly different. For more information, see:

- Adding Conveyors to a Simulation Model
- Moving, Resizing, and Reversing Conveyors
- Connecting Conveyors

Another difference is that the majority of the settings for a Straight and Curved Conveyors can only be changed by using the Conveyor System Tool to change the Conveyor's Type settings. For more information, see:

- Introduction to the Conveyor System Tool
- Accessing Type Properties
- Conveyor Type Settings

Lastly, if you double-click on a Straight and Curved Conveyor, you will open its Properties dialog box. There are only two tabs available for conveyors: Triggers and Labels. These tabs have options that function the same way as most other objects in the FlexSim library.

Statistics

Conveyors track the following statistics. These can be viewed by clicking on a conveyor and then going to the Statistics pane in Quick Properties.

- Content The number of items in the conveyor.
- Input The total number of items that have entered the conveyor.
- Output The total number of items that have exited the conveyor.
- Staytime The durations that items are in the conveyor.
- State The conveyor uses the following states:
    - Empty - There are no items on the conveyor o Conveying - There is at least one item on the conveyor o Stopped - There is at least one item on the conveyor, but the conveyor's speed has is set to zero. The conveyor will go into this state immediately when its speed is set to zero (i.e. at the time it STARTS to decelerate to zero).
    - Blocked - Only used for non-accumulating conveyors. When an item on a non-accumulating conveyor is stopped or blocked, the conveyor will become blocked. Note again that accumulating conveyors do not use this state. If you want to get better blocked state statistics on accumulating conveyors, you should use photo eyes.
- Percent Full The percent that the conveyor is "filled" with items. This uses the following formula: `100 * (TotalStopSpace / ConveyorLength)` where `TotalStopSpace` is the sum of the stop spaces of all the items

currently on the conveyor. Note that this percentage may go above 100%, specifically when there are items straddling multiple conveyors.

Join Conveyors acts more like a tool than an object. Use it to create a curved conveyor connecting two conveyor sections. For more information on Join Conveyors, see Connecting Conveyors - Using the Join Conveyors Object.

A Decision Point is a versatile object you can use to build logic into your conveyor system. Decision Points can also be linked to a Merge Controller to notify it when a released slug has cleared designated points in the system.

For ideas about how to use Decision Points to create complex sortation systems or merges, see Flow Control.

Additionally, you can skim through the chapter on Pick List Behaviors for more information about some of the Decision Point's capabilities.

The majority of the settings for a Decision Point are found in the Decision Point's Type settings. For more information, see:

- Introduction to the Conveyor System Tool
- Accessing Type Properties
- Decision Point Type Settings

Statistics

Decision Points track the following statistics. These can be viewed by clicking on a object and then going to the Statistics pane in Quick Properties.

- Content The number of items that have arrived but not yet continued through the decision point. This will always be 0 or 1.
- Input The total number of items that have arrived at the decision point.
- Output The total number of items that have continued through the decision point.

A Photo Eye is a versatile object you can use to build logic into your conveyor system. Photo Eyes can also be linked to a Merge Controller to notify it when a released slug has cleared designated points in the system.

For ideas about how to use Photo Eyes to create complex sortation systems or merges, see Flow Control. Additionally, you could possibly skim through the chapter on Decision Point and Photo Eye Behaviors (Picklists) for more information about some of the Photo Eye's capabilities.

Another difference is that the majority of the settings for a Photo Eye Photo Eye's Type settings. For more information, see:

- Introduction to the Conveyor System Tool
- Accessing Type Properties
- Photo Eye Type Settings

The previous topic, Photo Eye Type Settings, also has a good explanation of the relationship between Blocked and Cleared states.

Statistics

Photo Eyes track the following statistics. These can be viewed by clicking on a photo eye and then viewing the Statistics pane in Quick Properties.

- Content The number of items that have covered the photo eye but have not yet cleared it. This is usually either 0 or 1, but if the photo eye has an offset angle, it may be greater than 1.
- Input The total number of items that have covered the photo eye.
- Output The total number of items that have cleared the photo eye.
- State The photo eye tracks state using two different state profiles, namely the default state profile, and a custom "PhotoEye" state profile. The custom profile uses the following states: o Clear The photo eye is completely clear. o Blocking The photo eye was covered from a Clear state, but its Block Time has not yet expired.
  - o Blocked The photo eye is covered and its Block Time has expired.
  - o Clearing The photo eye was uncovered from a Blocked state, but its Clear Time has not yet expired. For the default state profile, it uses the following states:
  - o Idle The idle state corresponds to the Clear plus Blocking states in the photo eye's custom state profile.
  - o Blocked The blocked state corresponds to the Blocked plus Clearing states in the photo eye's custom state profile.

A Station is an object you can use to add processing points into the conveyor system. The station works very much like a standard Processor object, except that it is placed as a point in the conveyor system, instead of as an object outside of the conveyor system.

The majority of the settings for a Station are found in the Station's Type settings. For more information, see:

- Introduction to the Conveyor System Tool
- Accessing Type Properties
- Station Type Settings

Statistics

Stations track the following statistics. These can be viewed by clicking on a station and then viewing the Statistics pane in Quick Properties.

- Content The number of items being processed by the station. This will always be either 0 or 1.
- Input The total number of items that have started being processed by the station. Note that this may be different than the total number of items that have conveyed over the station, depending on whether the Process Condition always processes items.

- Output The total number of items that have finished being processed by the station. Like Input, this may be different than the total number of items that have conveyed over the station.
- Staytime The time from when an item arrives at the station to the time it is finished being processed. This is only recorded for items that are actually processed at the station.
- State The station uses the following states: o Idle The station is not processing an item. o Processing The station is processing an item.
  - o Waiting for Operator The station is waiting for an operator to come to process the item.

Motors can be used to control whether the conveyor systems are on or off at a given time. The Motor can also be used to sync dog gaps on a power and free chain loop when simulating a power and free conveyor system.

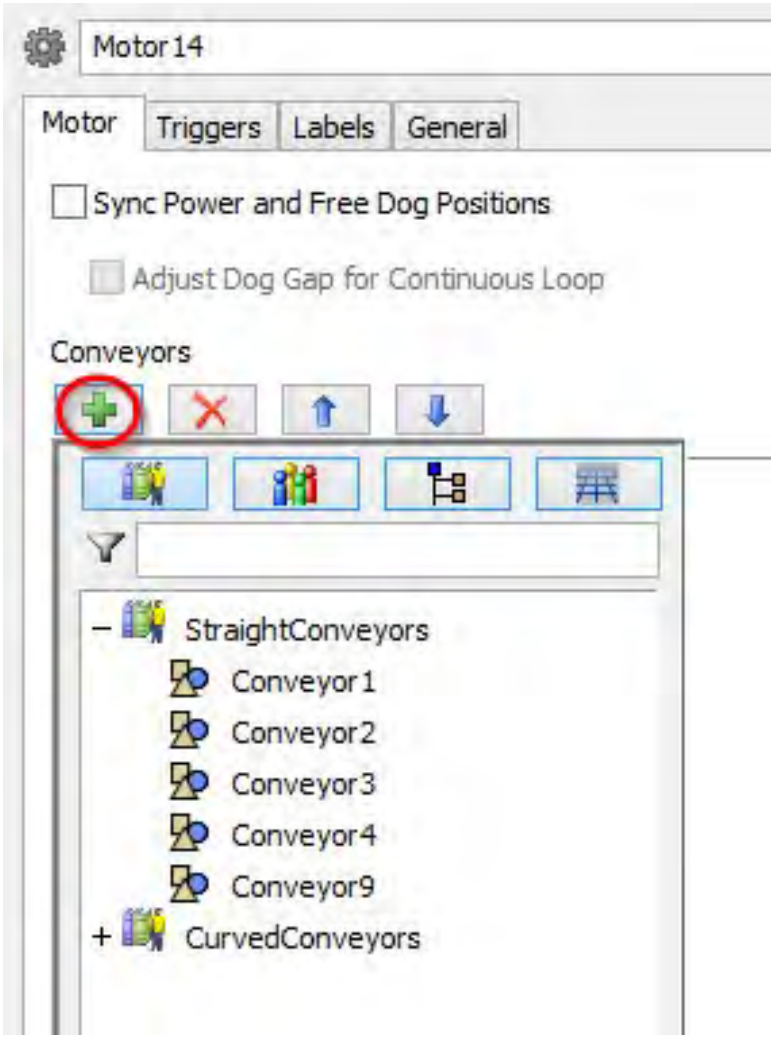
Adding Conveyors to a Motor

When you add conveyors to a Motor, the Motor will stop and restart all connected conveyor lines under certain conditions (such as when it receives a message from another object or when a Decision Point or Photo Eye alerts it to stop/resume). You can add conveyors to a Motor two ways:

The first way is to make an input/output port connection ('A' connect) between the Motor and the conveyors.

The second way is to use the Motor Properties dialog box to add conveyors:

1. Double-click the Motor to open its Properties dialog box. In the Motor tab under Conveyors, click the Plus sign . A pop-up box will appear that will allow you to select a conveyor from a list of conveyors, as shown in the following image:


Motor14
Motor
Triggers
Labels
General
Sync Power and Free Dog Positions
Adjust Dog Gap for Continuous Loop
Conveyors
StraightConveyors
Conveyor1
Conveyor2
Conveyor3
Conveyor4
Conveyor9
CurvedConveyors

2. Click StraightConveyors and/or CurvedConveyors to expand the list and view the available conveyors. You can click on multiple if needed. Click Select to add the conveyors. The conveyor you selected should now appear in the Conveyors list.

3. Click OK to apply the settings and close the Properties dialog box.

A blue line will now go from the Motor to all the conveyors to which it is connected.

Using Decision Points or Photo Eyes to Control a Motor

Follow these steps to use a Decision Point or Photo Eye to stop or restart a Motor:

1. Make sure that there is a center port connection (an 'S' connect) from the Decision Point or Photo Eye to the Motor.

2. Also make sure that the Motor is connected to all the conveyors it should control. (See the previous section, Adding Conveyors to a Motor.)

3. Double-click the Decision Point or Photo Eye to open its Properties dialog box.

4. Decide what conditions should cause the Motor to stop and what should cause it to restart (such as when an item when an item arrives at a Decision Point or Photo Eye). Find the appropriate Trigger on the Decision Point or Photo Eye and click the Plus sign to add a new Behavior.

5. Select Stop/Resume from the list of available Behaviors, then select either Stop Motor, Stop Motor and Delay, or Resume Motor. (See Stop/Resume for more information about each Behavior and its available settings.)

6. Click OK to apply the settings and close the Properties dialog box.

See Stop/Resume for more information about controlling Motors with Decision Point or Photo Eye.

Power and Free Settings

When you double-click a Motor, you will open the Properties dialog box. The Motor tab has two check boxes that affect Power and Free systems. (See Power and Free Systems for information about creating Power and Free simulations.)

If checked, the Sync Power and Free Dog Positions will use the Motor to ensure that the initial dog positions will be aligned on all of the conveyors connected to it. For example, if the Motor's first connected conveyor is 10.2 meters long, and the dog gap is 1.0 meters, then it will set the dog position to start at 0.0 on the first conveyor and at 0.8 on the second connected conveyor (the remaining distance to the next dog after the end of the first conveyor).

Motor Connection Order

The order of connected conveyors is important when using Sync Power and Free Dog Positions. As mentioned in the above example, the first dog position on the second connected conveyor will be set to to 0.8. This is the

second conveyor connected to the motor (as listed in the Conveyors Listbox of the Motor properties) and not necessarily the conveyor connected to the end of the first conveyor. Make sure that the conveyors are connected to the motor in the same order that the conveyors in the loop are connected or the dog positioning will be incorrect. You can adjust the order of connected conveyors by using the up and down arrows above the list of connected conveyors.

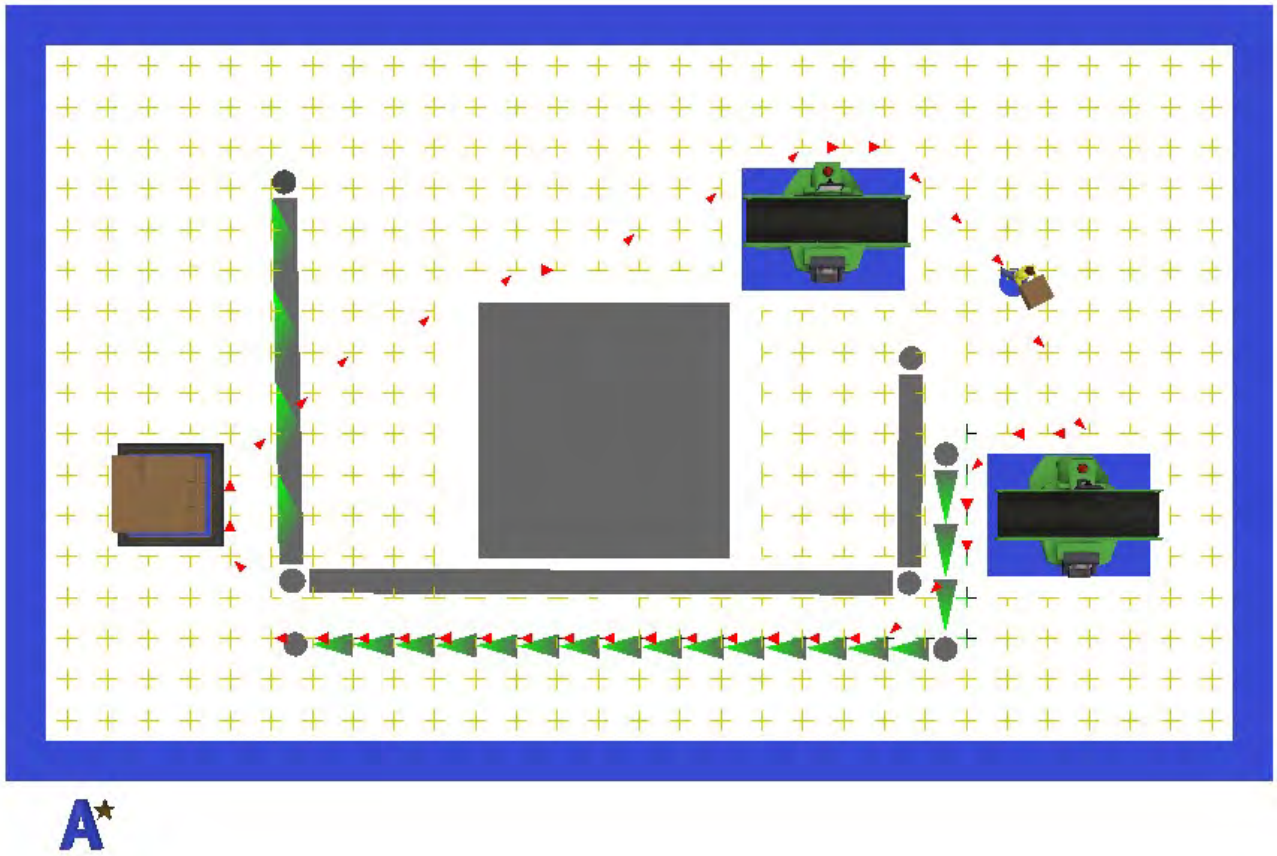
If Sync Power and Free Dog Positions is unchecked, or if Power and Free conveyors do not have an associated Motor, the dog positions will be defined automatically when the first item enters the conveyor.

The Adjust Dog Gap for Continuous Loop check box is available if Sync Power and Free Dog Positions is checked. When checked, the Motor will assume that the conveyors connected to it represent a looping Power and Free chain. The Motor will calculate the total length of the loop based on the combined lengths of the connected Power and Free conveyors and will adjust the dog gap of each conveyor so that the total length of the loop divides evenly by the dog gap. For example, if the total length of a Motor's conveyors adds up to 100.2 meters, and the dog gap is 1.0 meters, the Motor will adjust the dog gap of each conveyor to be 1.002 meters (100.2 / 100), so that the full loop makes up exactly 100 dogs. The algorithm rounds to the nearest number of dogs.

The Merge Controller is an object that can control how different conveyor lanes merge together. Use the Merge Controller to define a lane release strategy for the merge. To learn how to use a Merge Controller works, see Merging and Slug Building - Sawtooth Merging Using a Merge Controller.

Transfers are not available in the FlexSim Library. Rather, they are created whenever a conveyor is connected to another object. There are three kinds of Transfers:

- Transfers - Connections from one conveyor to another. They can affect how an item is transferred between conveyors. See Transfer Type Settings for more information.
- Entry Transfers - Connections from a non-conveyor object (such as a Source or other Fixed Resource) to a conveyor object. They can affect how an item is transferred to a conveyor. See Entry Transfer Type Settings for more information.
- Exits Transfers - Connections from a conveyor to a non-conveyor object (such as a Sink or other Fixed Resource) to a conveyor object. Exit Transfers have the ability to send for a transport, or in other words, a Task Executer to pick up items and deliver them to another object. Task Executers can pick up items from a particular point on the Exit Transfer or a range of possible points along an Exit Transfer. See Exit Transfer Type Settings for more information.



Topics

- Travel Members
- Barriers
- Deep Search
- Cached Paths
- Draw Modes

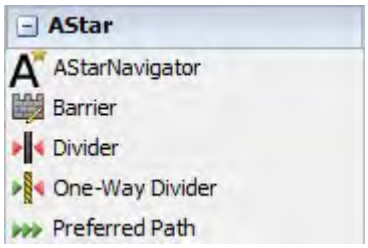
A Star, also known as A*, is a search algorithm used to find a path between points. As seen in the above image, the algorithm uses a grid of nodes that Task Executer objects (travel members) travel through. Each node specifies in which direction travel members can move. In the above images, the + lines represent these nodes. Each line points in the direction that can be travelled. The algorithm will look at nodes in the direction of travel and determine which direction is the fastest, including travelling diagonally between nodes. For more information see the Deep Search. The grid of nodes can be modified by creating Barriers restricting where the travel members can move or even influencing them to travel along certain paths.

A model may only contain one A* Navigator object, but it may have any number of barriers or members associated with it.

Visit Wikipedia for more information on how the algorithm works.

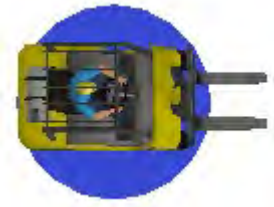
Note on performance: The A* algorithm requires more processing than a standard network node path. The search begins and the start node, then searches surrounding nodes toward the destination node. In a large model with many barriers, this search can cover a large amount of nodes. Performance can be improved through Cached Paths.

When the A* Module is loaded, the Library Icon Grid will display the following new objects:



Barriers can be created by single-clicking on the desired barrier, then clicking in the 3D view at the desired start and end points.

Travel Members



To connect TaskExecuter objects to the A* Navigator, hold down the 'A' key and click and drag from the A* Navigator object to the TaskExecuter object, or from the TaskExecuter object to the A* Navigator object. If the Draw Mode is set to Show Members, a blue circle will be drawn below the connected TaskExecuter objects.

If the Draw Mode is set to Show Traffic, red arrows will be draw along travel member paths. These paths decay (fade out) if they are not travelled upon. This allows you to see the most frequently travelled paths. If travel members are using the same paths frequently, it may be useful to Cache Paths. This will improve performance.

A list of members and their current state can be found on the A* Navigator's Properties Page.

Note on Travel Members: The AStar grid, or bounds, must cover the entire working area of Travel Members. This will help avoid issues that can occur when a Travel Member leaves the grid during offset travel.

Barriers

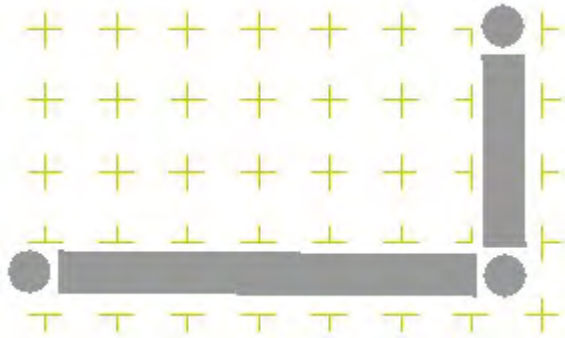
There are 5 types of barriers used by the A* Navigator. These barriers modify the grid used in the search algorithm.

Note: Once barriers are created or modified, the model must be reset in order to recalculate the node table.

Solid Barrier

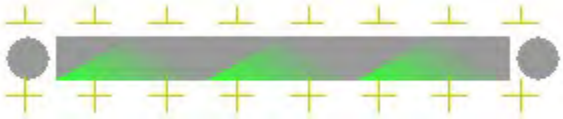
The simplest of barriers, the solid barrier modifies the grid to remove a rectangular area, preventing travel members from moving into the barrier.

Divider



Dividers disconnect nodes along their path, preventing travel members from crossing the divider. Dividers can be modified in the 3D view by dragging the end or middle points. Any number of points may be added to a divider.

One-Way Divider



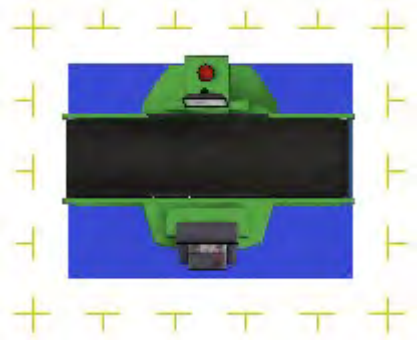
The one-way divider works very similarly to the divider, however, travel members are allowed to move through the divider in the direction of the green arrows, in this case, toward the top of the page. One-way dividers can be modified in the 3D view by dragging the end or middle points. Any number of points may be added to a one-way divider.

Preferred Path

Though the preferred path is not an actual barrier, it does function in the same way barriers do to modify the A* Navigator's node table. Instead of altering nodes by removing their directional indicators, the preferred path gives a greater weight to all of the nodes associated with the path. This Path Weight value can be changed through the A* Navigator's properties window. The path weight must be greater than 0 and less than 1 in order to function properly. Values above or below this can cause unintentional results. Preferred paths can be modified in the 3D view by dragging the end or middle points.

Preferred path's are one directional. Any number of points may be added to a preferred path.

FixedResource Barriers



FixedResource objects can be added to the A* Navigator as barriers. This can be down by holding the 'A' key down and clicking and dragging from the FixedResource object to the A* Navigator object, or by clicking on the A* Navigator object and dragging to the FixedResource object. If the Draw Mode is set to Show Members, a blue box will be drawn below the connected FixedResource objects.

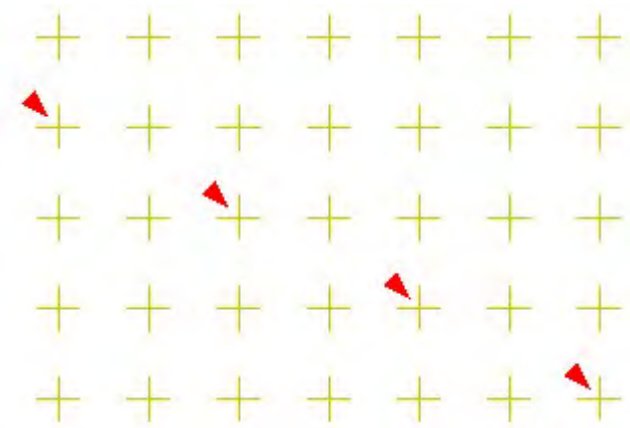
Arbitrary rotations of FixedResrouce objects will not result in a rotated barrier. The barrier associated with a FixedResource object will rotate to the nearest 90 degrees.

Offset Travel - Since offset travel by default does not use a navigator to calculate where the TaskExecuter moves to, TaskExecuter's may enter a FixedResource's barrier. To change this, open the TaskExecuter's properties window and change *Travel offsets for load/unload tasks* to *Use navigator for offset travel* or to completely remove offset travel change this to *Do not travel offsets for load/unload tasks*.

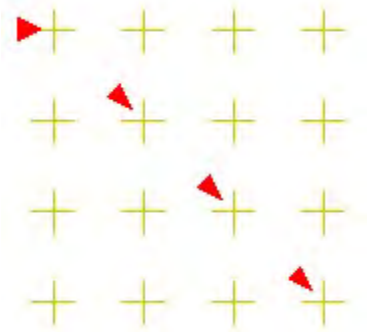
Note: The blue box is drawn just above the bottom of the object, so some objects may not show depending on their 3D shape.

Deep Search

Deep search allows travel members to take more direct paths between nodes and 'cut corners'. By default Deep Search is on. Deep search will cause the algorithm to check nodes that are two places away from the start node rather than just one. Below is an example of a travel member's path while deep search is on:

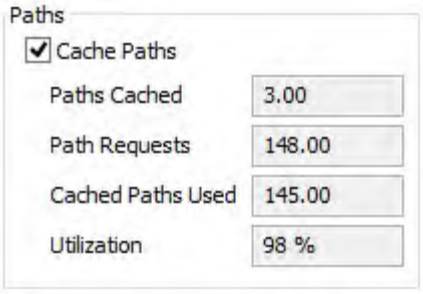


Alternatively, with deep search off, the travel member can only move in 90 degree or 45 degree angles, as shown below:



A downside of using the deep search option is that it is more processor intensive. It will take longer to calculate paths between points. Speeds can be improved by Caching Paths.

Cached Paths



Each time a travel request is made, the A* algorithm searches from the start node to find the most direct path to the end node. This is processor intensive and for large grids with frequent travel requests, can slow your model down. Caching paths can reduce processing time by reusing paths that have already been calculated rather than recalculated every time a travel request is made.

Paths Cached - The total number of unique paths that have been cached.

Path Requests - The total number of travel requests.

Cached Paths Used - The total number of travel requests that used a cached path.

Utilization - This is the calculated utilization of cached paths to path requests using the equation: *CachedPathsUsed / PathRequests * 100*.

Draw Modes

Draw modes specify which elements of the A* Navigator will be drawn in the 3D view.

Show Barriers - Draws solid barriers, dividers, one-way dividers and preferred paths.

Show Bounds - Draws the A* Navigator's bounding box that contains all barriers and nodes. The width of the bounding box is equal to the A* Navigator's Node Spacing.

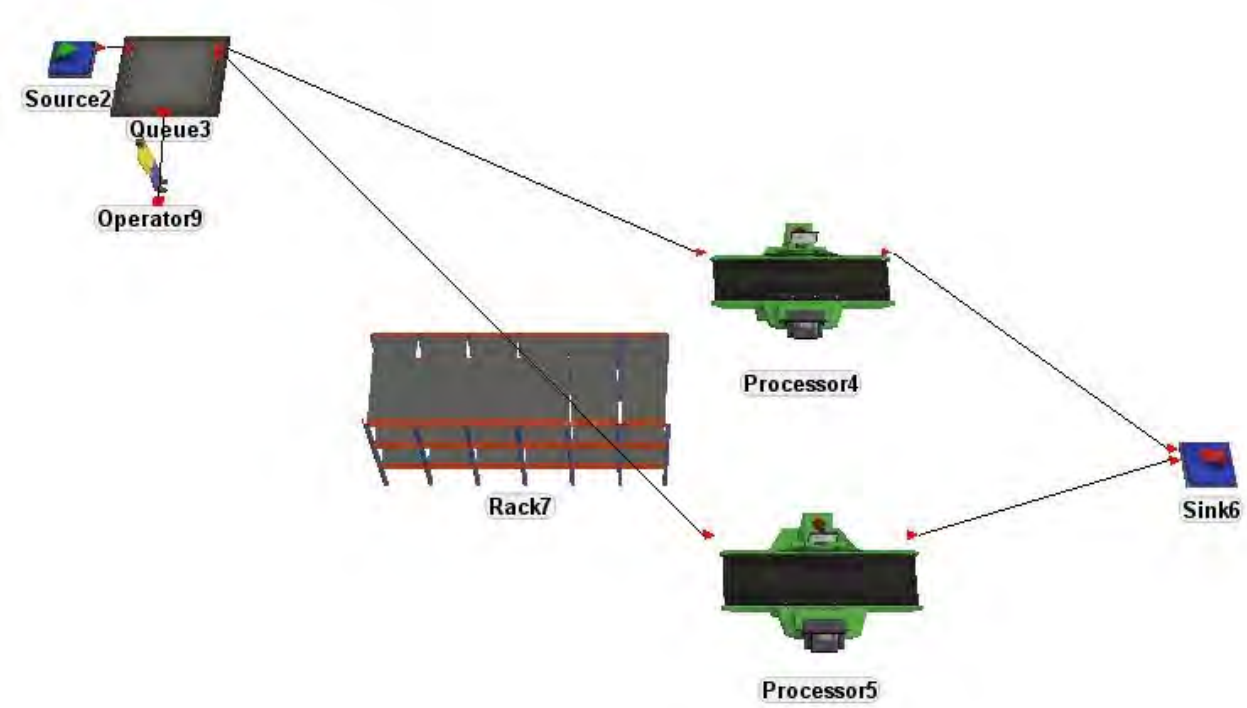
Show Grid - Draws the grid of nodes that can be travelled along by travel members. Note: Drawing the grid can drastically slow down your model if the grid is large.

Show Members - Draws circles under TaskExecuter objects and rectangles under FixedResource objects that are 'A' connected to the A* Navigator.

Show Traffic - Draws paths that have been recently travelled on by travel members. These paths will decay over time if they are not travelled on.

A* Navigator Example

The Model



We'll use the following model to show how the A* Navigator can be used in a simple model. The real power of the A* Module is to layout very large and complex models that would require large network node layouts to handle all of the TaskExecuters.

Create the Navigator

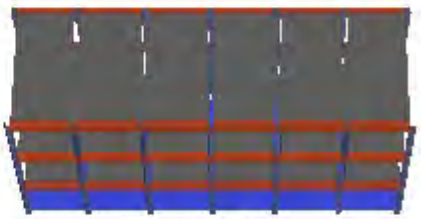
First, an AStarNavigator object must be added to the model. Only one AStarNavigator object is available per model.

- Create an AStarNavigator object by click and dragging from the Library to somewhere in the 3D view.

Connecting Members

In this model, there is only one TaskExecuter object, however, we also want to include the Rack as a barrier.

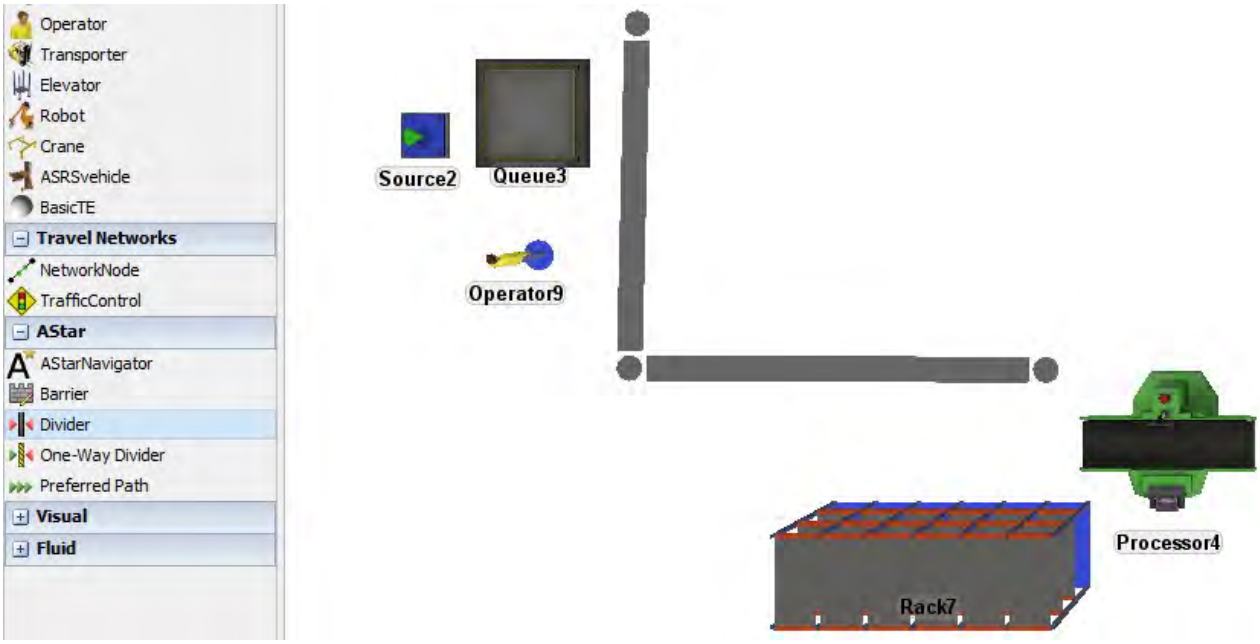
- Make an 'A' connection between the AStarNavigator and the Operator. This can be done by holding the 'A' key down and clicking and dragging from the AStarNavigator to the Operator, or from the Operator to the AStarNavigator. A blue circle will appear below the Operator.
- Make an 'A' connection between the AStarNavigator and the Rack. A blue rectangle should will below the Operator.



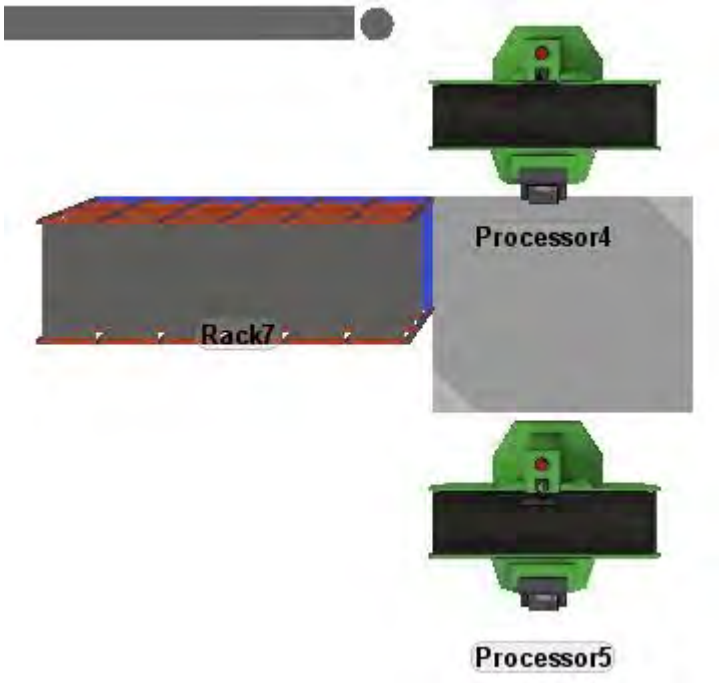
Creating Barriers

Along with the Rack, we will also create a couple of barriers to represent walls or obstacles that the operator cannot walk through.

- Click on the Divider icon in the Library to enter create mode.
- Click somewhere to the right of the Queue to create a new divider.
- Click twice more near the points shown in the image below to finish the divider.
- Right-click to exit create mode.

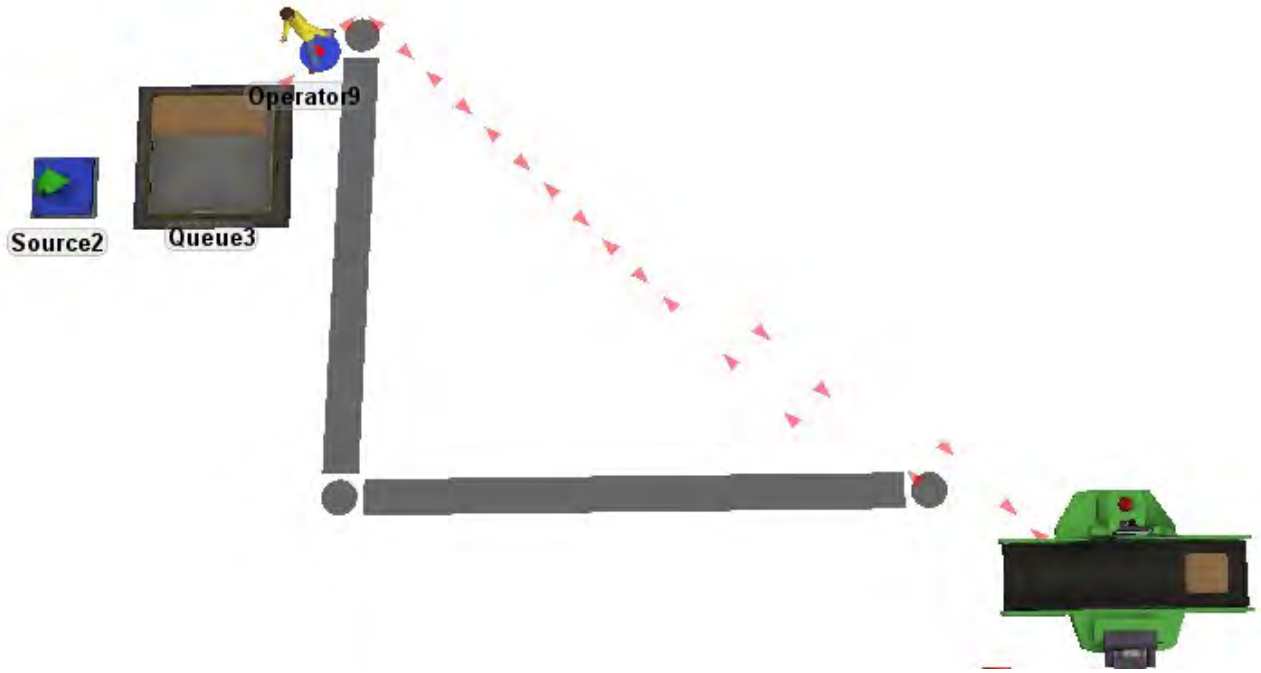


- Click on the Barrier icon in the Library to enter create mode.
- Click once where the lower left corner of the barrier will be as shown below.
- Click again in the upper right hand corner to complete the barrier.
- Right-click to exit create mode.



Run the Model

You can now reset and run the model to see how the operator performs. Remember, if you make any changes to your barriers the model must be reset before running.

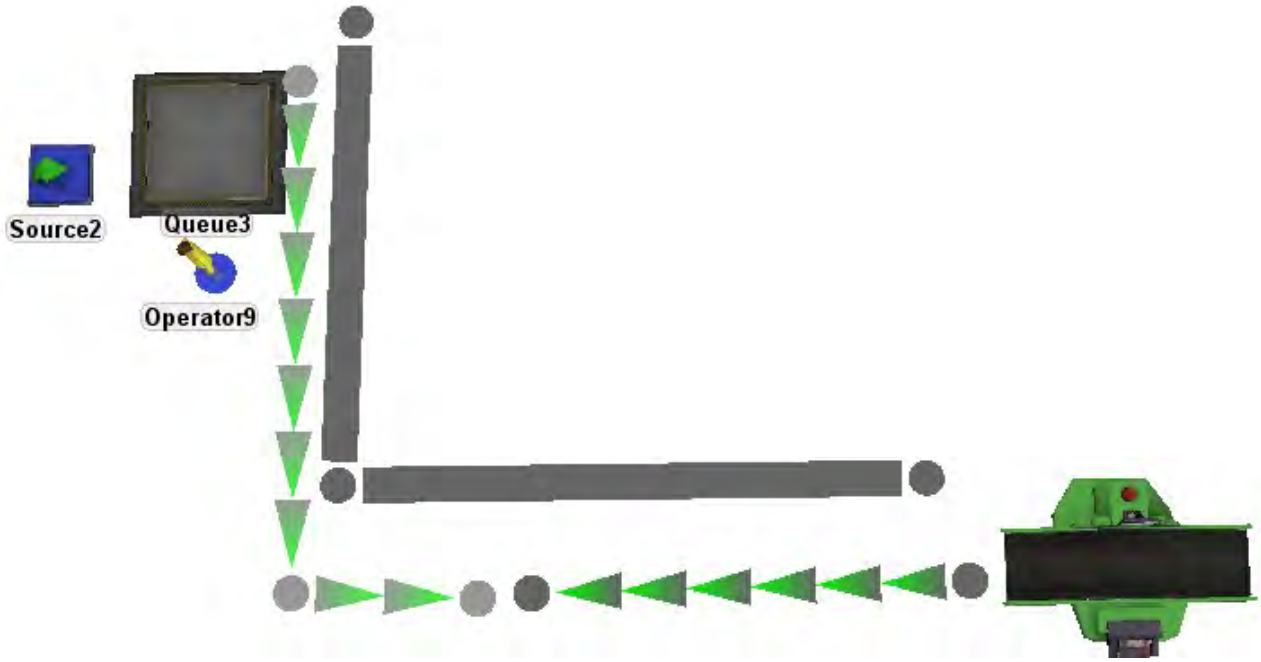


Notice that the operator found that the shortest path is to move up and over our divider. Though there are multiple ways to alter the operator's path (for instance making the divider extend out further) we will modify the operator's path through a Preferred Path.

Create Preferred Paths

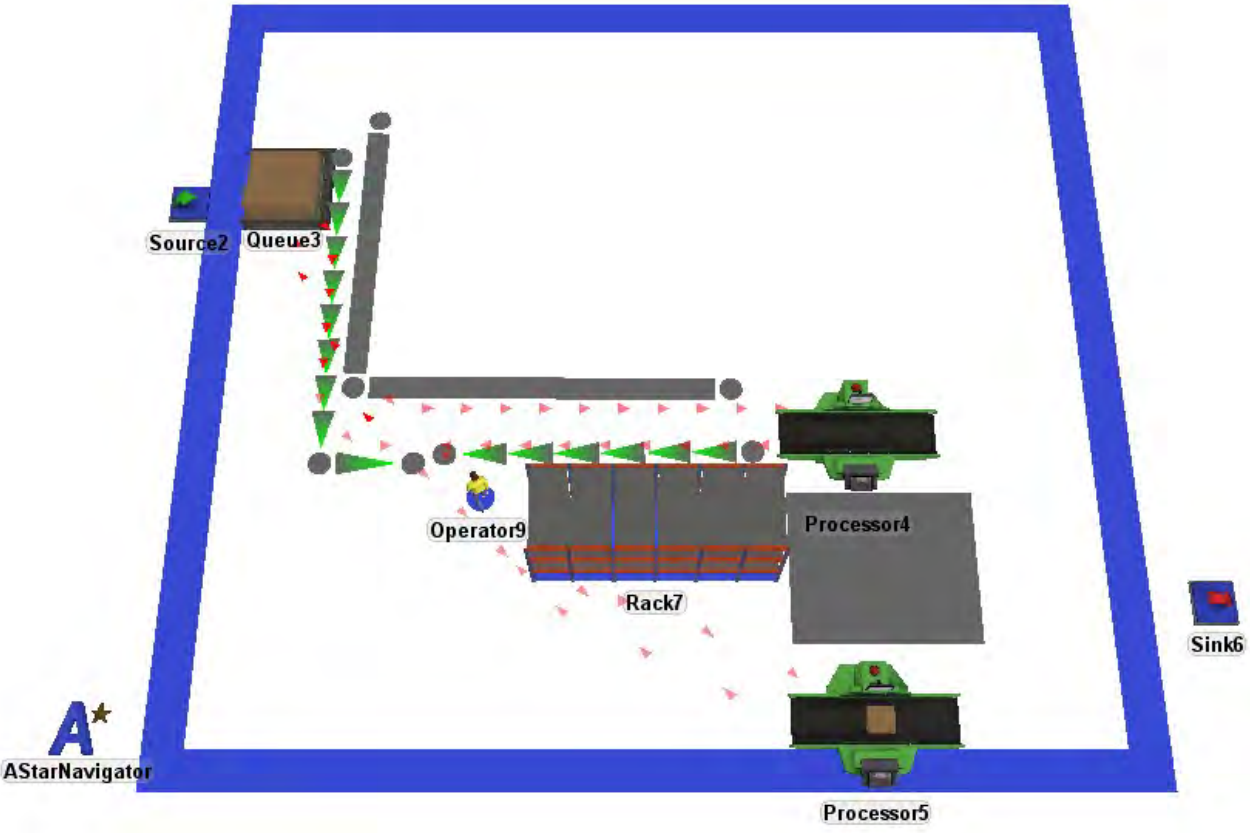
Preferred paths are not 'set' paths, rather they give the nodes associated with the preferred path a greater weight in the search algorithm. This will in affect, pull the operator toward the preferred path.

- Click on the Preferred Path icon in the Library to enter create mode.
- Create the following two preferred paths shown in the image below. Preferred paths are one-directional and thus must be laid out in the correct order.
- Right-click to exit create mode.

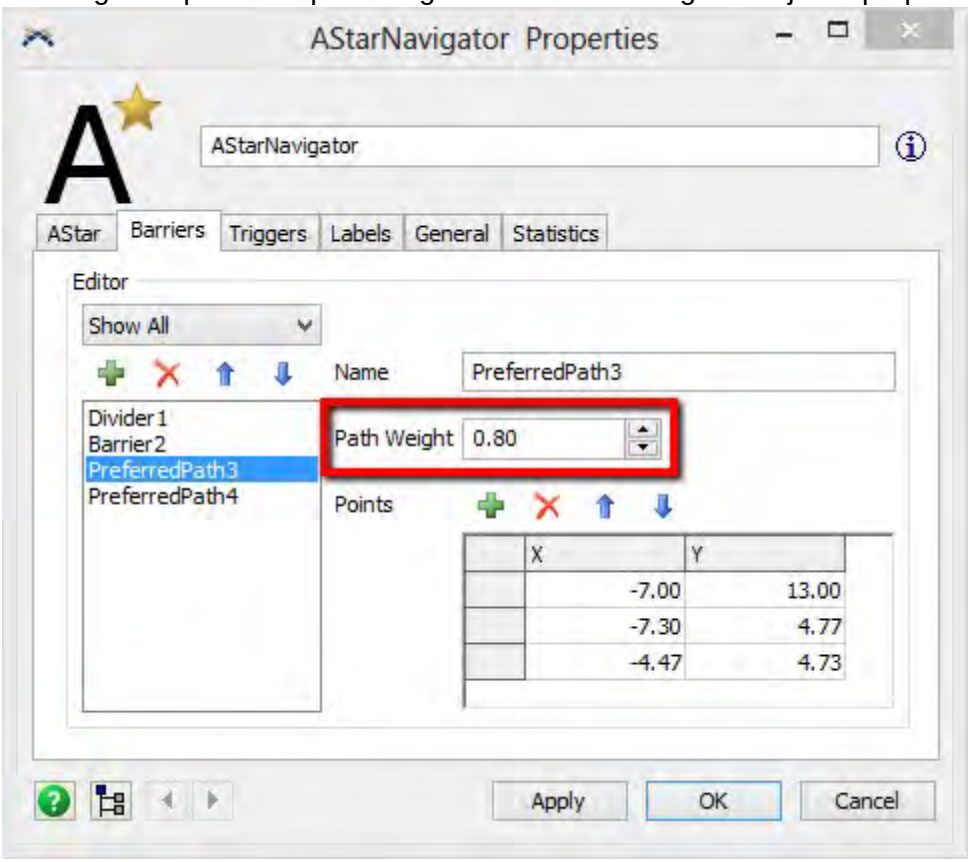


Run the Model

Reset and run the model.



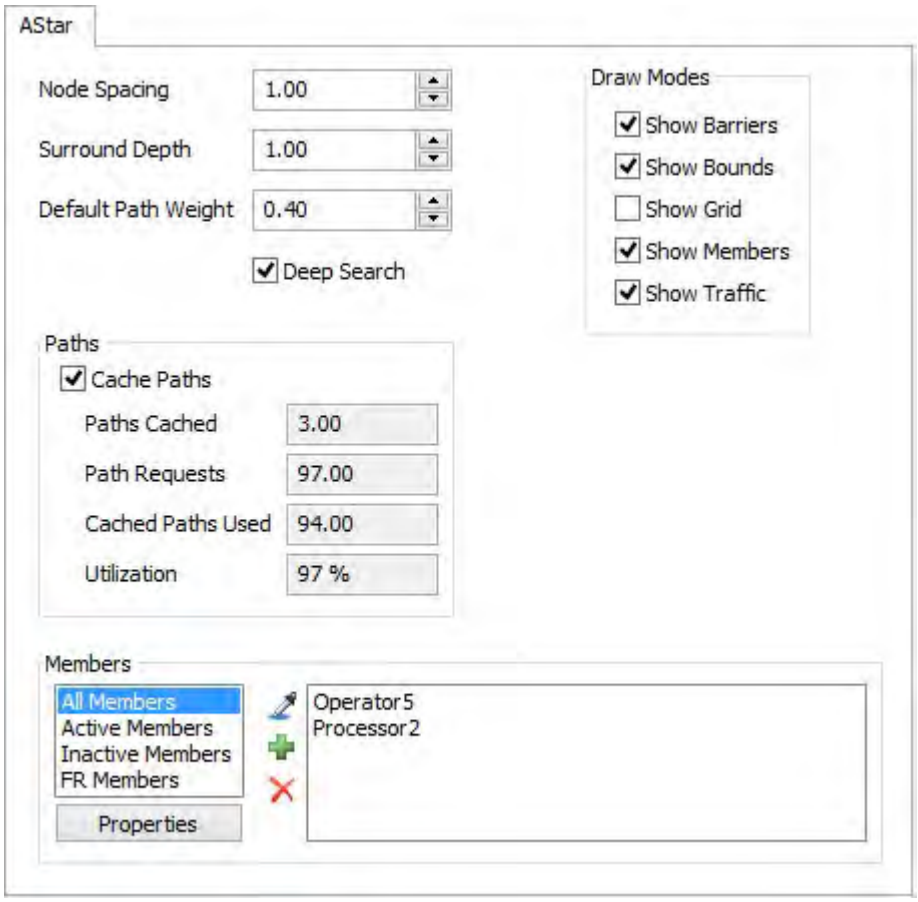
If your operator continues to move up and over the divider, you can either move the preferred path or you can change the preferred path weight in the AStarNavigator object's properties window.



Results of your model may vary from this example as placement of barriers and objects will make a difference in the calculated paths of the operator.

A* Navigator Reference

AStar Page



Node Spacing - This is the distance (in model units) between nodes in the A* Navigator's grid. Smaller spacing will allow travel members to move more smoothly and direct, however, more nodes will require more processing time to calculate paths.

Surround Depth - This number specifies the number of nodes that are placed around the outside of the navigator's outer most barriers. This number must be an integer and be greater than 0

Default Path Weight - This is the default path weight for preferred paths. Any newly created preferred paths will be given this value for their path weight.

Deep Search - When checked, the algorithm will perform a deep search when calculating travel paths. For more information on deep search see Concepts - Deep Search.

Draw Modes

Toggle draw modes on or off. For more information on draw modes see Concepts - Draw Modes.

Paths

Cache Paths - When checked, the A* Navigator will cache paths to be reused in order to save processing time.

For more information on cached paths see Concepts - Cached Paths.

Members

Status - Select which member lists to view.

Properties - Open the properties window of the selected member.

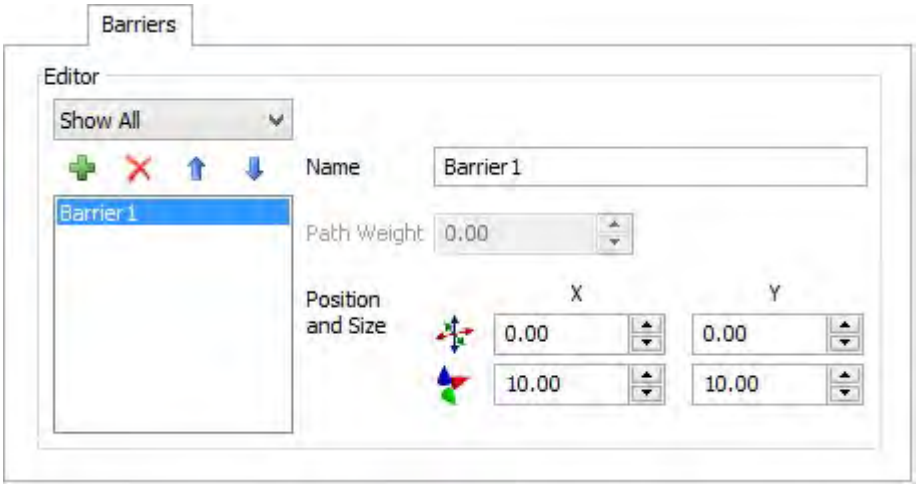
- Click to enter "Sample" mode then click on an object in the model to add as a member.

- Click to open an object selection window and choose objects to add as members.

- Remove the selected member.

Members List - Displays all of the objects at the currently selected status.

Barriers Page



Filter Barriers - Select a barrier type to filter the Barrier List.

- Add a barrier, divider, one -way divider or preferred path.

- Remove the selected barrier.

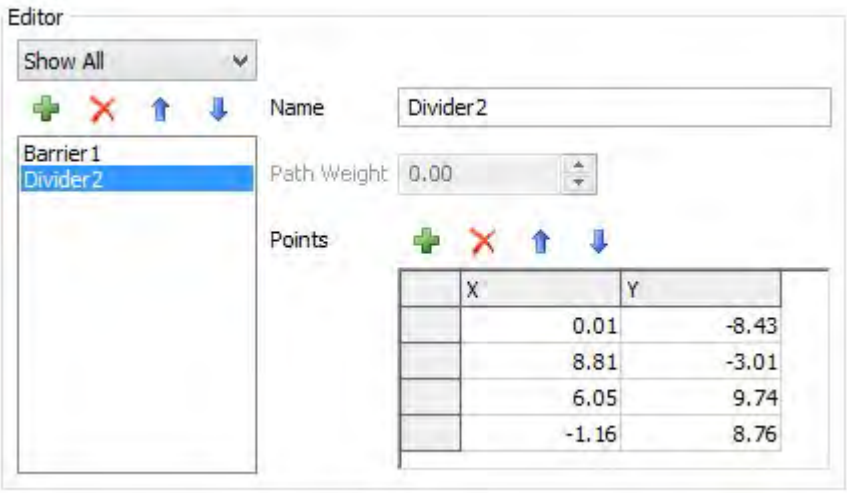
- Reorder the selected barrier up or do wn in the list.

Barriers List - Displays the A* Navigator's barriers.

Name - The name of the barrier. This has no affect on the model, but is for the modeller's convenience.

Path Weight - Preferred Path's only. This is the path weight added to the nodes associated with the path.

Position and Size (Barriers) - Specify the size and position of the barrier.



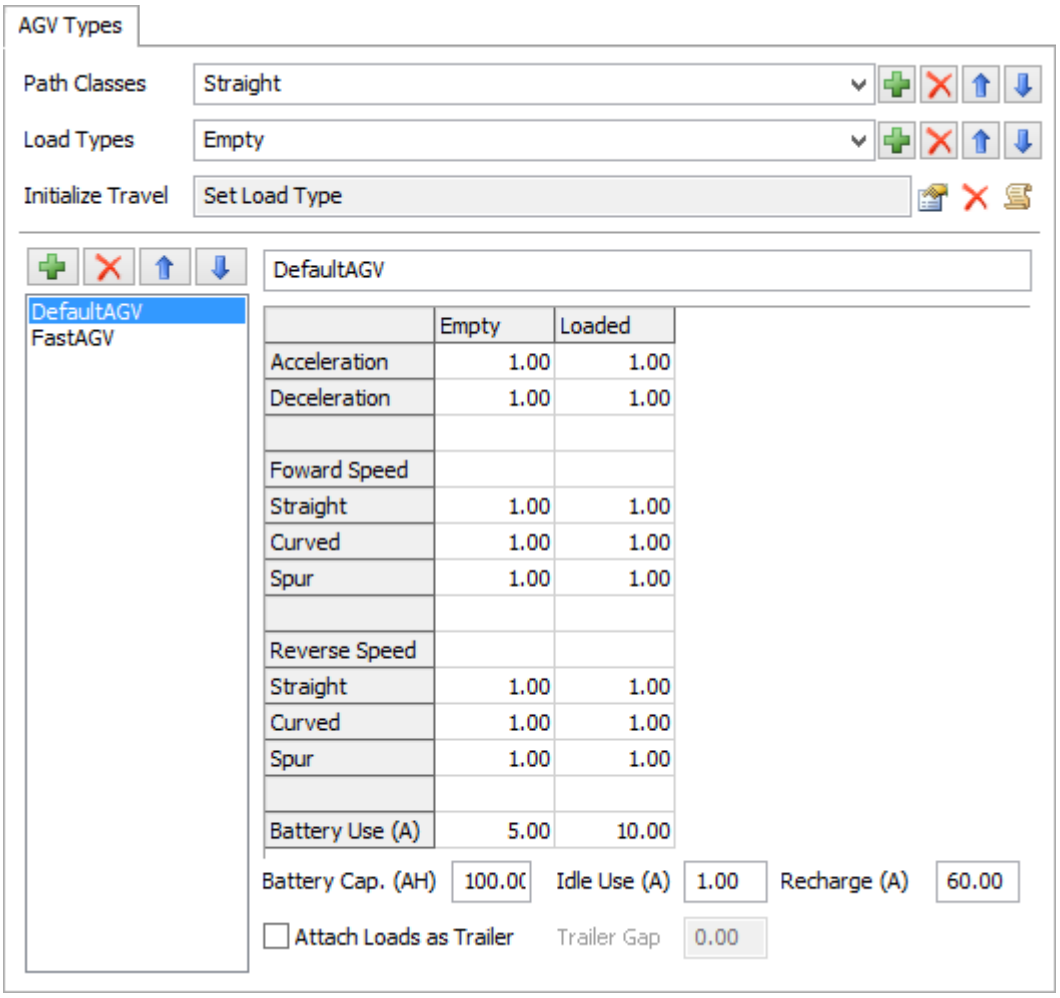
Points (Dividers/One-Way Dividers/Preferred Paths) - Define x,y positions of each point.

- Add a point to the end.

- Remove the selected point.

- Reorder the selected point up or down in the list.

AGV Types Page



You can get to the AGV Types page by right-clicking on a Path or Control Point and choosing AGV Network Properties.

A detailed conceptual overview of AGV Types can be found in Tutorial Lesson 3.

Path Classes - Here you can add, remove, re-order and rename the set of Path Classes for the model. Path Classes are specifically used for breaking out AGV speeds by path. When you add a Path Class, a new speed row associated with that Path Class will be added to each AGV Type table for both forward and reverse speed.

Load Types - Here you can add, remove, re-order and rename the set of Load Types for the model. Load Types are a user-defined list defining categories for what an AGV is carrying. This allows you to break out AGV speeds by the AGV's current load.

Initialize Travel - This is a trigger that is fired at the beginning of each travel operation. The primary objective of this trigger is to set the AGV's current Load Type.

AGV Types List - Here you can add, remove, re-order and rename the list of AGV Types for the model.

AGV Type Spec Table - In the AGV Type Spec Table you define max speeds by Path Class, Load Type and AGV direction, as well as acceleration, deceleration and non-idle battery usage. See Lesson 3 for a detailed description of these values.

Battery Levels and Usage

Each AGV Type has a defined Battery Capacity (in Amp Hours), Idle Battery Usage (in Amps), and Recharge Rate (in Amps). Additionally, each AGV Type has a non-idle Battery Usage (in Amps) broken out by Load Type. Each AGV starts the simulation at its maximum battery capacity, and then will track its battery usage over the course of the simulation. Whenever it is idle, its Idle Battery Usage applies. Whenever it is doing a travel operation, its battery usage is based on its current Load Type. If you set the AGV to recharge, it will recharge at its recharge rate until it is full or it starts its next travel operation, whichever comes first. To query battery level, start a recharge, or manually set the battery level, etc., see the documentation for the agvinfo() command.

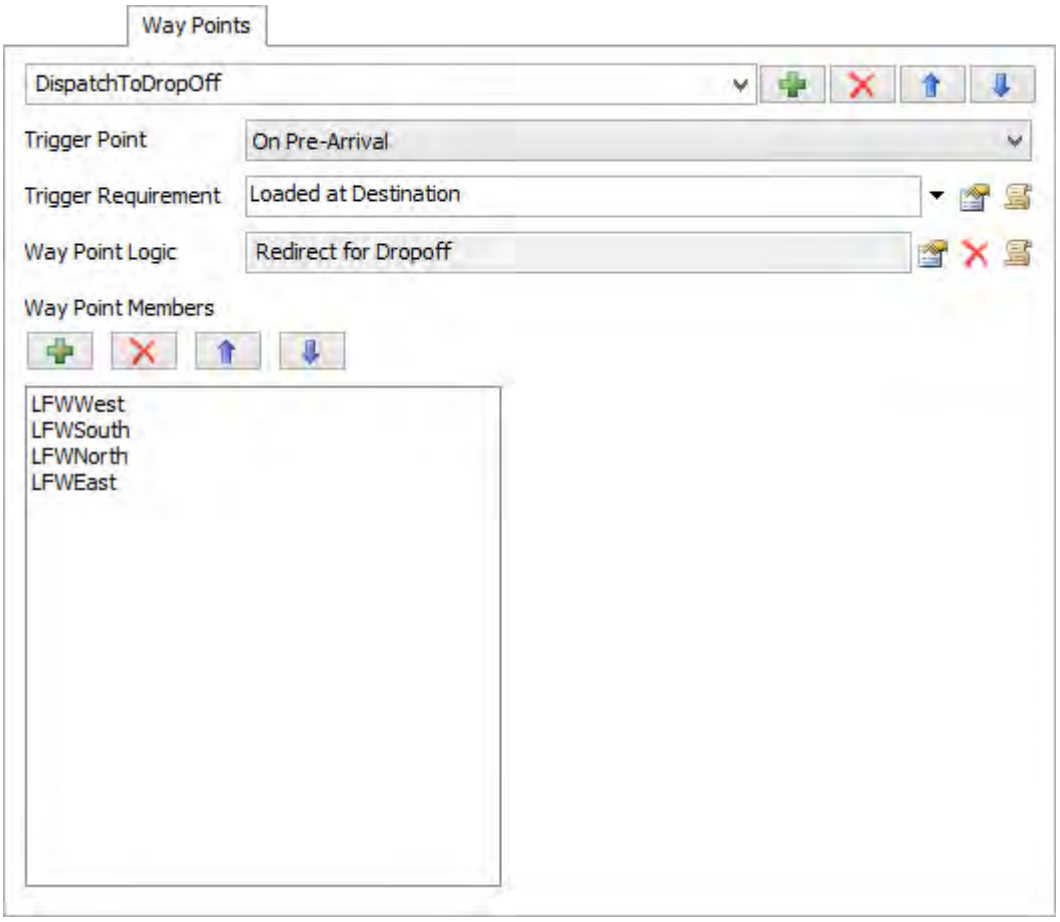
Attaching Trailers

You can also make the AGV attach loaded items as trailers to the AGV. This will make the loaded items trail behind the AGV on its path, instead of being carried on the AGV. Check the box Attach Loads as Trailer, and then define the Trailer Gap, which is the distance of the gap between the back of the AGV and the front of the trailing item.

Deceleration and End Speed

Usually when you give an AGV a task to travel to a destination, you will want the AGV to end that travel task fully decelerated to a stop. However, in some cases you may want to end a travel task while still moving, for example if you want to make dispatching decisions on-the-fly. In this case you can define a non-zero end speed for the travel task. Doing this will actually shift earlier the end time of the travel task, as well as the position of the AGV at the time it finishes. In other words, instead of ending on the destination control point while still moving, the travel task will end while the AGV is still approaching the control point, at the defined end speed. If you don't immediately give the AGV any subsequent travel tasks, then the AGV will decelerate down to stopped, arriving at the control point AFTER the travel task has finished. If, however, you immediately give the AGV a new travel task, it will continue to the new task, starting at the end speed of the previous task, traveling through the original destination control point.

Way Points



Way Points are used to define AGV control logic that will happen when an AGV passes over a control point. However, going forward we advise you to use process flow instead of Way Points for AGV control. FlexSim provides a template AGV control process flow, which can be used as a starting point for defining AGV control logic. This process flow is used in lesson 2.

You can get to the Way Points page by right-clicking on a Path or Control Point and choosing AGV Network Properties.

Way Points List - Here you can add, remove, re-order, and rename each Way Point. Trigger

Point - This defines when to fire the Way Point Logic. Options are:

- On Pre-Arrival - The Way Point Logic will be fired at the point when the AGV would otherwise start to decelerate to stop at the Way Point.
- On Arrival - The Way Point Logic will be fired when the AGV arrives at the Way Point. Note that if the Trigger Requirement is met and this Trigger Point is chosen, the AGV will slow to a stop at this Way Point and then fire the Way Point, even if the Way Point is not the AGV's final destination. Hence if you don't want the AGV to slow to a stop, you should use On Pre-Arrival.

Trigger Requirement - A field that should return 1 if the Way Point should be fired, 0 if not.

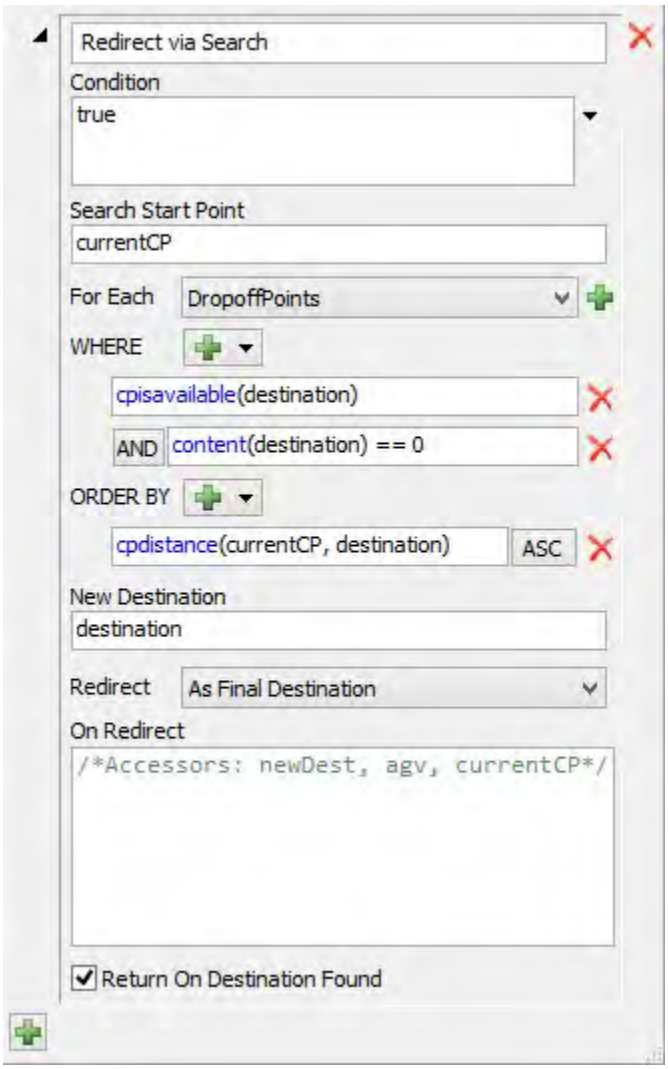
Way Point Logic - The code for the Way Point.

Way Point Members - The list of Control Points that are part of this Way Point. Here you can add, remove and re-order the members list.

Way Point Trigger Actions

Redirect via Search

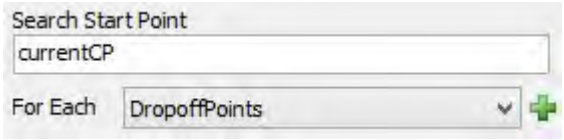
Redirect via Search will redirect the AGV by search through the Control Point Connections of the Control Point the AGV is arriving at.



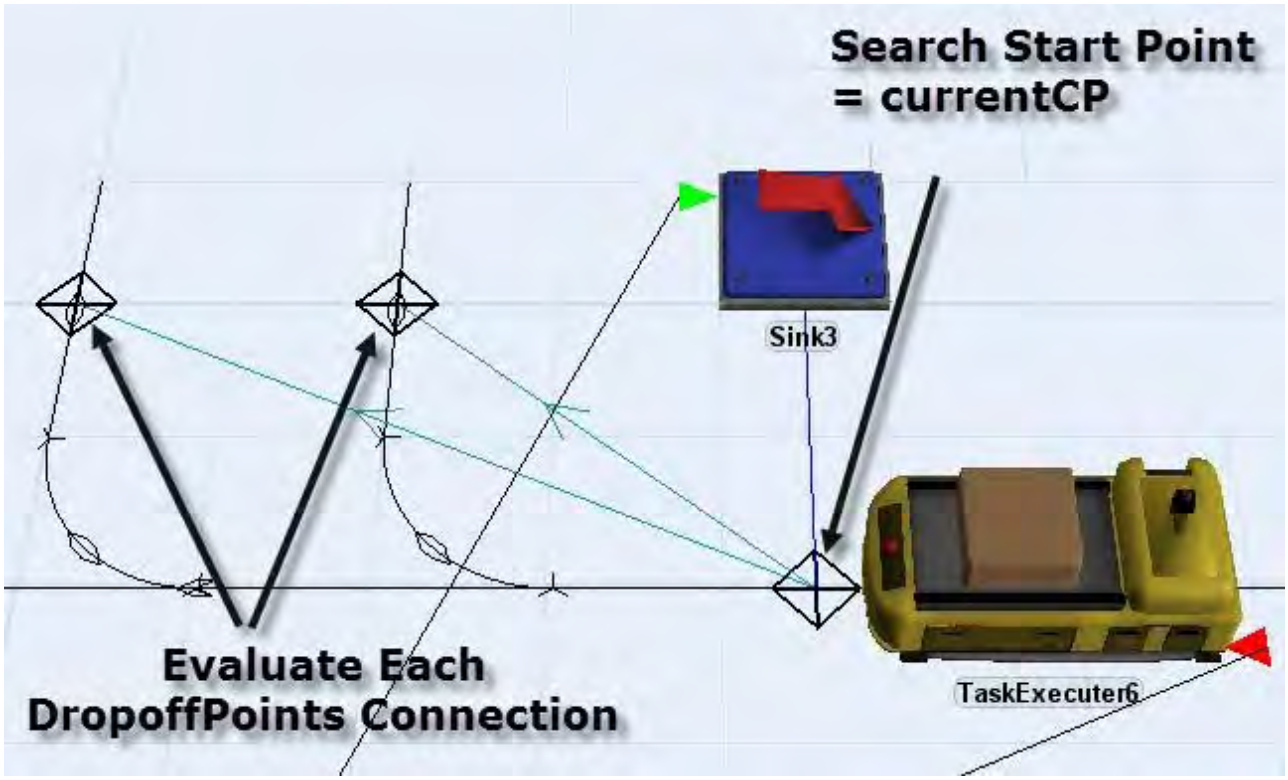
Condition

The condition field specifies when you should execute this action. The default (true) means always execute the action. You can use the drop-down button on the right to help you fill in this field.

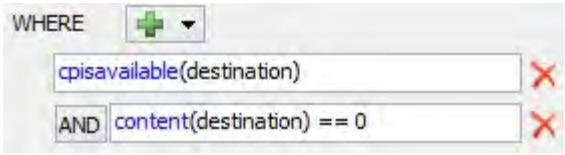
Search Scope



Here you define the scope of the search. This includes a Search Start Point, which is the Control Point from which to start your search. The default, currentCP, is the Control Point where the AGV has arrived. Next, you define the set of Control Point Connections to search. The default, DropoffPoints, means that it will search all of currentCP's DropoffPoints connections, evaluating whether to dispatch to each of the resulting Control Points.



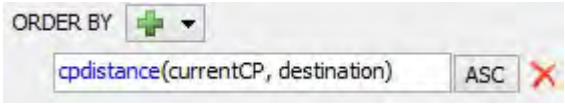
Filter (WHERE)



The WHERE filter defines the criteria for dispatching to a given Control Point in the search. The default is:

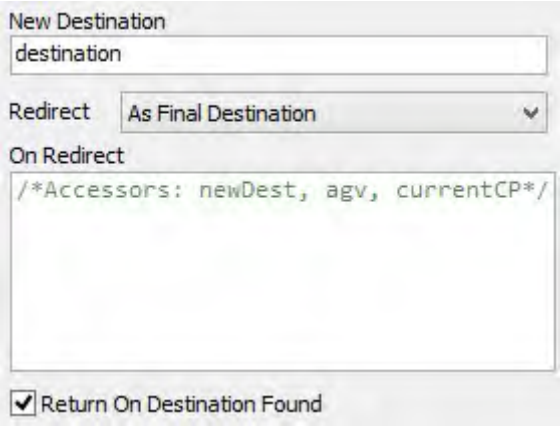
1. The Control Point is available, i.e. it is not claimed by another AGV.
2. The Control Point has nothing in it, i.e. there is not a flow item at that location waiting to be picked up.

Prioritization (ORDER BY)



The ORDER BY parameter defines which Control Points are "best" to dispatch to if there are multiple matches. The default is to dispatch to the Control Point that is closest to currentCP.

Other Parameters



New Destination - The new destination that you want to dispatch to. This is usually just the matched Control Point itself, i.e. destination.

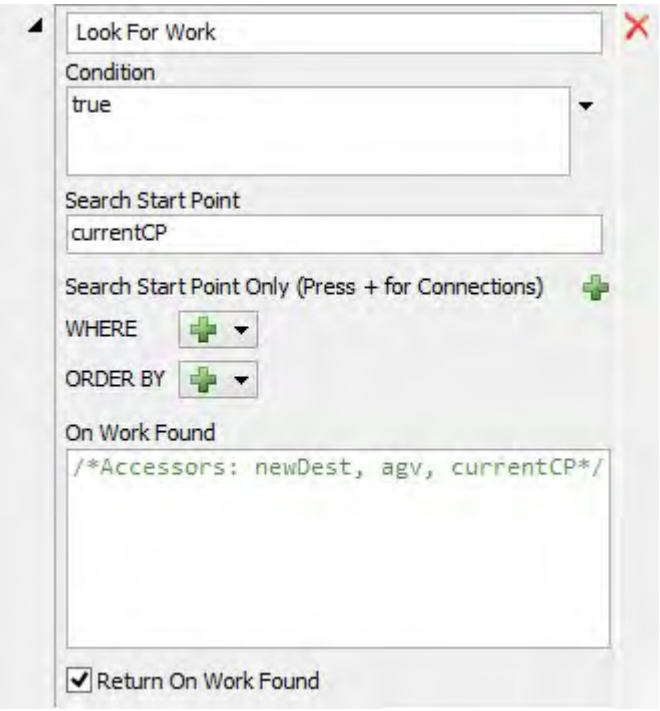
Redirect Type - This parameter defines how you want to redirect the AGV. The options are:

- Redirect as Final Destination - the AGV will finish the travel once arrive (as long as another Way Point doesn't redirect him again).
- Redirect and Wait - the AGV will travel to the destination and wait until it is redirected again
- Redirect and Continue - the AGV will travel to the destination and continue on to its original destination. If On-Arrival is chosen, the AGV will slow down to a stop at the intermediate destination before continuing. If On-Pre-Arrival is chosen, the AGV will just pass through the intermediate destination, not slowing down.

On Redirect - A piece of code to execute when a valid AGV is found.

Return On Destination Found - If checked, the Way Point trigger will discontinue execution if it finds an AGV to redirect to. The trigger will thus skip any subsequent logic. Sometimes you may daisy-chain multiple actions together, where you attempt to do one thing, and if you don't find anything to do, you try to do the next thing. If this is checked, then it will return when found, meaning you successfully found something to do for the AGV, and thus don't want to look for anything else.

Look For Work



This action will search for work to do and take up work when found.

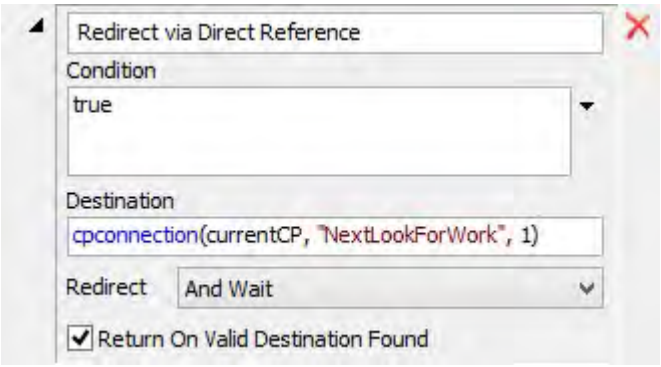
Condition - Defines the condition by which to look for work. Same as the Redirect Via Search condition. Search Start Point / For Each - Defines the search scope. Similar to Redirect Via Search scope, except here it is searching the task sequence queues of the objects/Control Points in the search. The default only searches the task sequence queue of the AGV's current Control Point. You will often use the LookForWork action in conjunction with a FixedResource's Use Transport > Move Item Via AGV action. In such cases, task sequences will often be stored on the Control Points themselves, so the task queue search will be focused on Control Points' task queues.

Filter (WHERE) - Defines criteria for taking up a given task sequence, i.e. loading/unloading a specific flow item. Similar to the Redirect Via Search filter, but here the focus of the filter is focused on a flow item, namely a task sequence's load/unload flow item.

Prioritization (ORDER BY) - Defines prioritization. Again, like the Redirect Via Search prioritization, but here the focus is a flow item.

Additional Parameters - Similar to Redirect Via Search additional parameters.

Redirect Via Direct Reference



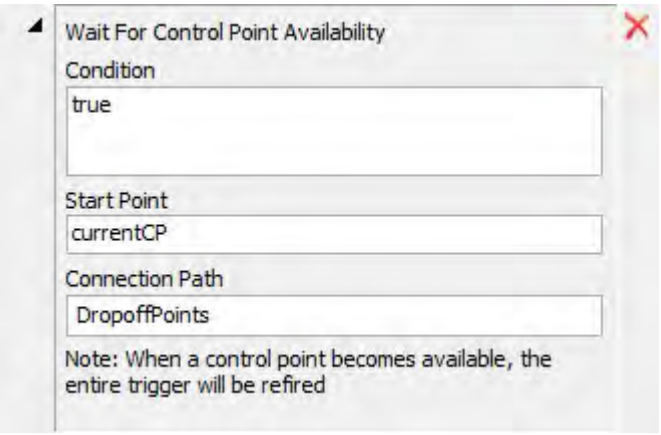
This action will redirect the AGV to another Control Point by a direct reference.

Condition - Same as Redirect Via Search condition.

Destination - The reference to the Control Point/object to redirect to.

Additional Parameters - Similar to Redirect Via Search additional parameters.

Wait For Control Point Availability



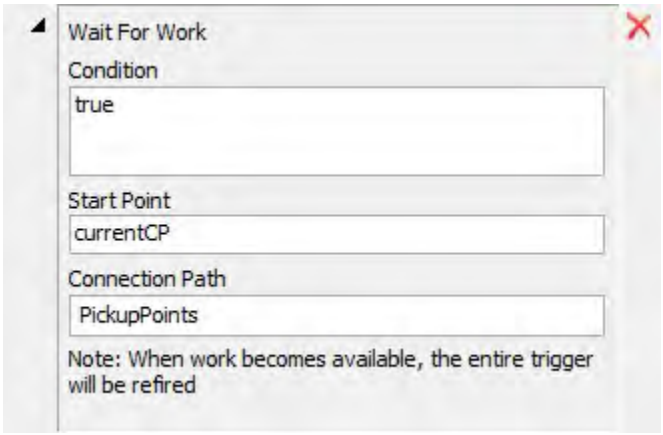
This action will wait for one of a set of Control Points to become available. When availability happens, all actions for the Way Point trigger will be refired. Usually you will place this action after a Redirect Via Search action, such that if nothing is found to redirect to, i.e. all Control Points are unavailable, you can wait for a Control Point to become available.

Condition - Same as Redirect Via Search condition.

Start Point / Connection Path - These fields define a set of Control Points to "listen to" for availability. The Start Point is a Control Point to start at, usually the AGV's current Control Point (currentCP). Connection Path defines the Control Point Connection(s) fanning out from the Start Point that enumerate the Control Points to listen to. The default is DropoffPoints. DropoffPoints would be used, for example, if you have a loaded AGV that needs to get into a spur to drop off its item, but currently all drop-off spurs are taken. So, you wait for a Control Point to become available, and you listen for availability on each of the drop-off points associated with the AGV's current Control Point. For Connection Path, you can define a single Control Point Connection, as in the default DropoffPoints, or you can define "chained" connections, such as "NextLookForWork>DropoffPoints". This would start at the Start Point, and fan out

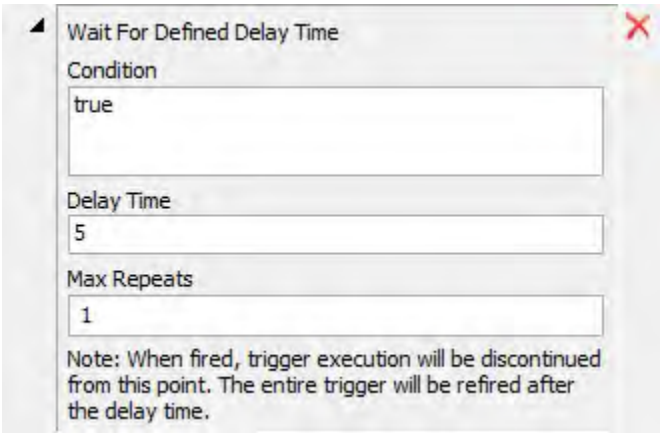
first to all of its NextLookForWork connections, and then for each of those it would enumerate all of the DropoffPoints connections.

Wait For Work



This action will wait for work to arrive at a defined set of Control Points. This should be used in conjunction with the Use Transport > Move Item Via AGV action. When that action is used, flow items will be placed in the Control Points for pick up. This Wait For Work action will listen for when items enter those Control Points, and then will refire all actions for the Way Point again. Usually you will add the Wait For Work after a Look For Work action, such that if no work is found, you can then wait for work to arrive.

Wait For Defined Delay Time



This action will wait for a defined delay time and then refire the Way Point trigger's actions again. Usually you put this at the start of a Way Point trigger. This can implement some delay associated with the real-life decision making process, i.e. if it takes the real-life system 5 seconds to process a dispatching decision, you first wait for 5 seconds, then fire the decision logic.

Control Point Connections

You can get to the Control Point Connections page by right-clicking on a Path or Control Point and choosing AGV Network Properties.

A detailed conceptual explanation of Control Point Connections can be found in Tutorial Lesson 2. For each Control Point Connection, you can define whether or not its visible in the 3D view, and its color. You can also add, remove, and rename connections as needed.

Using Lists

If you check the List box for a connection, all of those connections in the model will be added to a global partitioned list. This allows you to easily define AGV control logic through Process Flow. When an AGV arrives at a given destination, you can decide what to do by pulling from various list partitions associated with the AGV's current control point, such as, find a control point near here that has work that the AGV can do, or find a parking spot to go to, or continue to the next point to look for work. These operations can be done by pulling from the current control point's partition of the global list.

Connection Types

A Control Point Connection has one of three possible connection types:

Control Point One-Way - This connection type means the connection is a one-way connection from one Control Point to another Control Point. In other words, when you make one of these connections from Control Point A to Control Point B, Control Point B will not have a connection back to Control Point A.

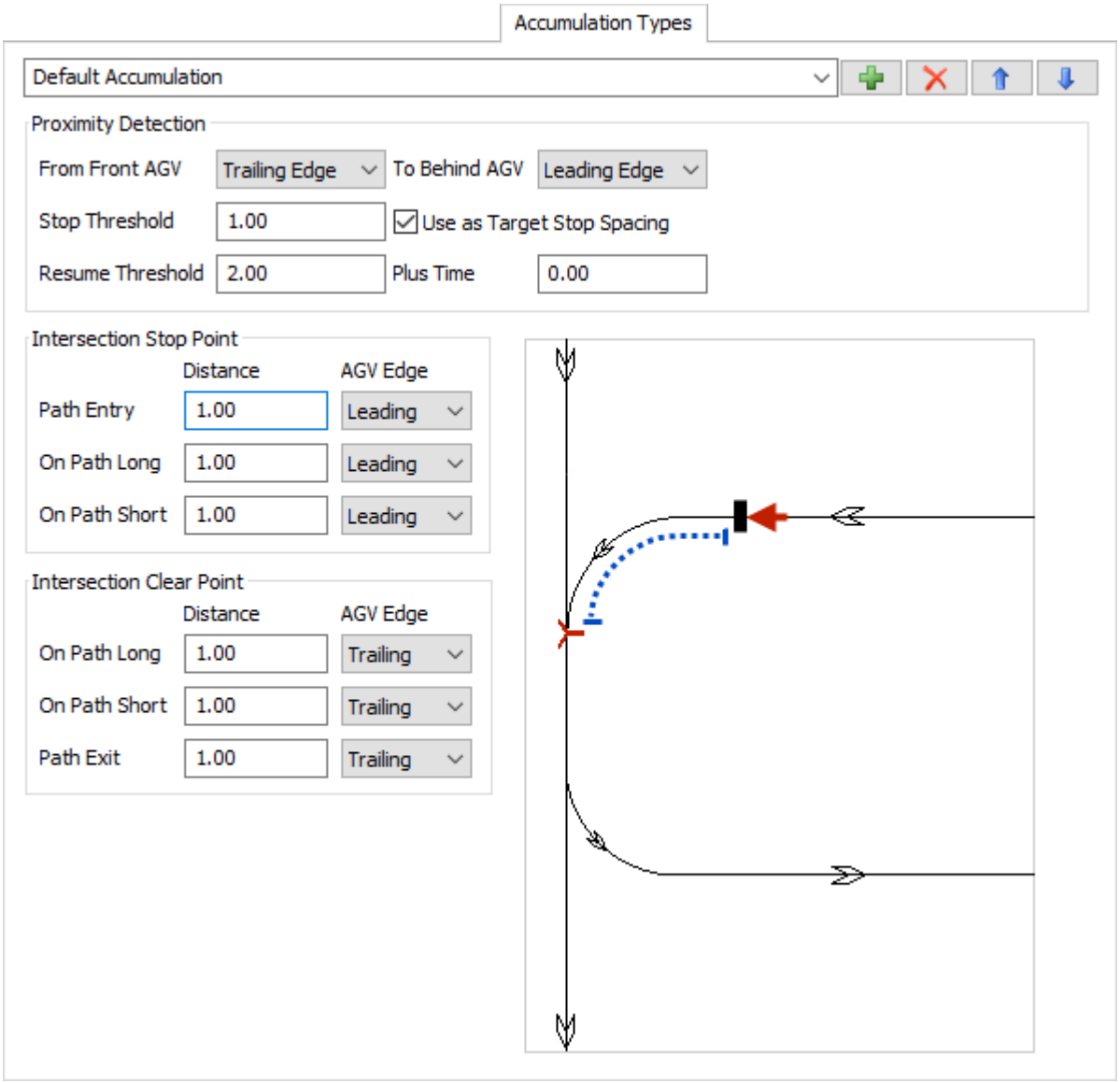
Control Point Two-Way - This connection type means the connection is a two-way connection from one Control Point to another Control Point. In other words, when you add one of these connections from Control Point A to Control Point B, a connection from Control Point B back to Control Point A will automatically be created as well. When you add one of these connection types, you can make it so that the reverse connection (B back to A) connects back using the same connection type (the To Self option), or using a new connection type. Technically the two-way connection doesn't give you any added capability over the one-way because you can do this same functionality using one-way connections by manually adding one-way connections in both directions. However, this can save time in creating the connections if needed.

Object Two-Way - This connection type represents a two-way connection from a Control Point to a model object. When you add one of these connections, the Control Point can accesss the object through that connection, and the object can access the control point through the same connection.

Using Connections

You can use Control Point connections through various trigger actions, such as Way Point trigger actions, or by manually accessing them with the cpconnection() command.

Accumulation Types Page



Accumulation Types let you define parameters for accumulating paths, i.e. paths on which AGVs will detect proximity and avoid collisions with each other.

You can get to the AGV Types page by right-clicking on a Path or Control Point and choosing AGV Network Properties.

Add, remove, rename and re-order Accumulation Types through the combobox and buttons at the top of the page.

To assign a path an Accumulation Type, click on the path, and in its Quick Properties on the right, choose the desired type from the Accumulation Type drop-down.

Note: AGVs will still use the standard Control Point allocation mechanism while on accumulating paths, i.e. they will still allocate ahead to their next Control Point. Thus if you want to allow more than one AGV between Control Points on an accumulating path, you should increase the Max Allocations value for the Control Points on that path.

Proximity Detection

These properties define how two AGVs will detect proximity with each other while on a common accumulating path.

From Front AGV Edge To Behind AGV Edge - This defines the AGV edges used to determine proximity. Usually this will be the default: Front AGV <Trailing Edge> to Behind AGV <Leading Edge>. This will evaluate the distance from the back of the AGV ahead to the front of the AGV behind. You could also choose something like Front AGV <Center> to Behind AGV <Center>, which would evaluate the distance from center to center.

Stop Threshold - This is the proximity distance at which an AGV will go into a "proximity stop" state, and slow down to a stop.

Use as Target Stop Spacing - If checked, then if an ahead AGV is stopped, the stop threshold defines the target stop spacing. Stopped AGVs will thus accumulate with this stop spacing. In other words, an AGV approaching a stopped AGV will begin decelerating to stop BEFORE the stop threshold (as a proximity distance) is breached. The AGV will instead start decelerating such that it comes to a stop with the stop threshold as its spacing behind the ahead AGV. In situations where both AGVs are still moving, the stop threshold will continue to be used in regular proximity detection.

Resume Threshold - The proximity distance at which an AGV can resume its travel after going into a "proximity stop" state. The resume threshold must be greater than the stop threshold.

Plus Time - An optional additional time, started when the Resume Threshold is reached, that the AGV will wait before resuming from a "proximity stop" state.

Intersections

When you define an Accumulation Type for a path, the AGV network will treat intersections on that path as allocations. Similar to the way AGVs must allocate Control Points and Control Areas, AGVs must allocate the intersection points on an accumulating path before proceeding past those intersection points. In the Accumulation Types page you define stop distances, which are distances before the intersection where the AGV must stop before an intersection point if the AGV cannot allocate it, as well as clear distances, which are distances after passing the intersection point where AGVs will release the intersection and allow other AGVs to claim it. Each of these distances are split out by whether the AGV is entering, exiting, or on the path. If already on a path or staying on a path, the distances are split out by the path geometry, i.e. whether or not the intersection branches out away from the AGV or toward it.

Intersection Stop Point

Here you define the stop distances for an intersection. For each distance, you define the distance itself as well as an AGV edge, which determine what part of the AGV should stop at the stop distance. Usually stop distances will be based on the leading edge of the AGV. When you click in a given field, the diagram on the right will display the distance / edge that you are defining, to help you figure out what exactly the field defines.

Path Entry - This defines the stop distance and agv edge associated with entering a path.

On Path Long - This defines the stop distance and agv edge associated with approaching an intersection when already on a path. It is the "long" stop distance because it will be applied to intersections that branch out toward the AGV, and thus require the AGV to stop farther away from the intersection point in order to provide room for merging AGVs.

On Path Short - This defines the stop distance and agv edge associated with approaching an intersection when already on a path. It is the "short" stop distance because it will be applied to intersections that branch out away from the AGV, hence the AGV can stop closer to the intersection without causing issues.

Intersection Clear Point

Here you define the clear distances for an intersection. Like with stop distance, you define the distance itself as well as an AGV edge. Usually clear distances will be based on the trailing edge of the AGV. When you click in a given field, the diagram on the right will display the distance / edge that you are defining, to help you figure out what exactly the field defines.

On Path Long - This defines the clear distance and agv edge associated with clearing an intersection when still on the path (not exiting). It is the "long" clear distance because it will be applied to intersections that branch out toward the clear point, and thus require the AGV to travel farther away from the intersection point before clearing it.

On Path Short - This defines the clear distance and agv edge associated with clearing an intersection when still on the path (not exiting). It is the "short" clear distance because it will be applied to intersections that branch out away from the clear point, hence the AGV can clear it closer to the intersection point without causing issues.

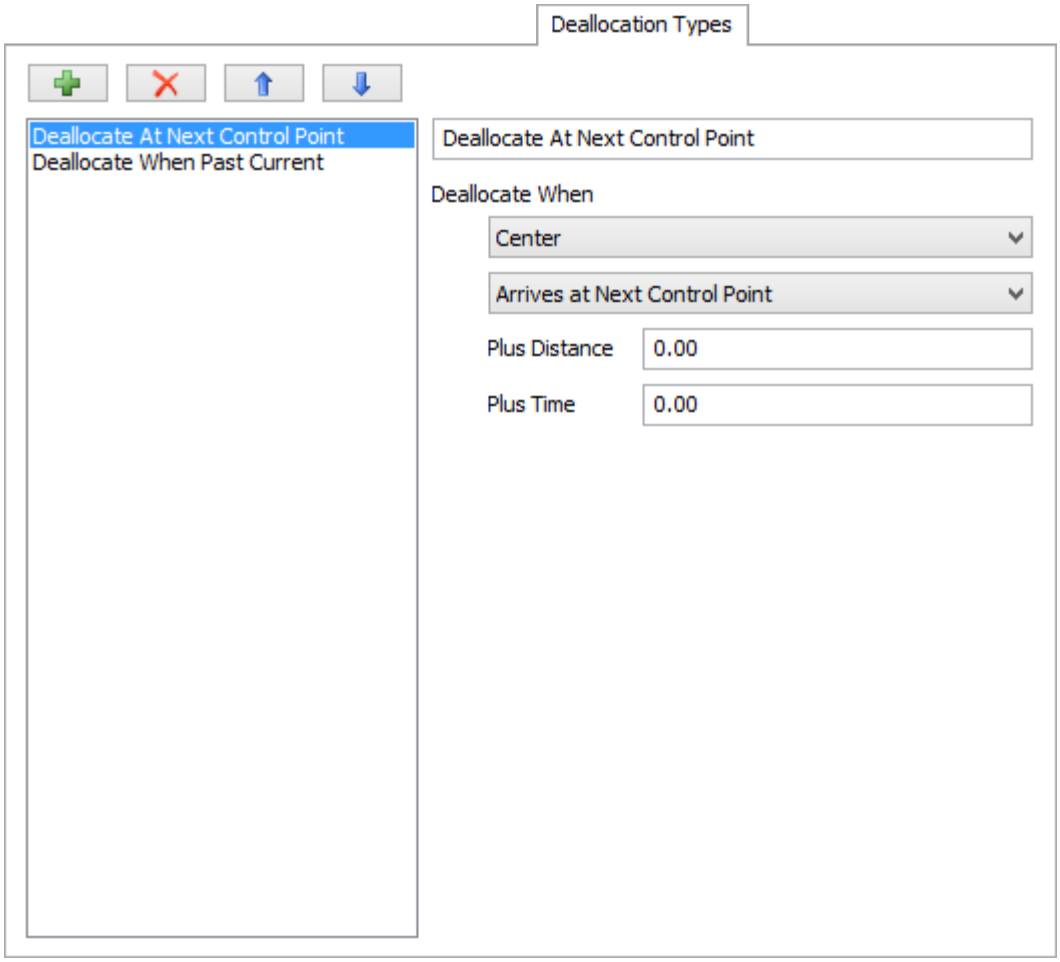
Path Exit - This defines the clear distance and agv edge associated with exiting a path.

Special Rules

There are a few special rules that apply to allocating intersection points.

Pure On-Path Allocations - The AGV network tries to make basic proximity detection take precedence over intersection stop and clear points. Thus, if an AGV is already on a path and already has an AGV ahead of it for which it is detecting proximity, and if it is not exiting the path at that intersection, then the AGV will be allowed to allocate the intersection point before the ahead AGV has cleared it. Since it is already detecting proximity on the ahead AGV, it is OK to have a simultaneous allocation of the intersection point because the basic proximity detection will already avoid proximity errors.

End-To-End Path Transfers - When transferring from the end of one path to the beginning of another path, the network still treats that as an "intersection" between two paths, i.e. it still requires an allocation of the intersection point. However, again if it can detect that, in transferring to the new path, the AGV will still be detecting proximity with the same AGV that it was detecting before the transfer, then it will go ahead and allow the intersection point to be allocated simultaneously by both AGVs, since the basic proximity detection will already prevent proximity errors. Note that this only applies to transfers between two paths of the same Accumulation Type. If the Accumulation Type is different, it will treat it like a regular intersection allocation.



You can get to the Deallocation Types page by right-clicking on a Path or Control Point and choosing AGV Network Properties.

A conceptual overview of Deallocation Types can be found in Tutorial Lesson 1.

Deallocation Types List - Here you can add, remove, re-order and rename each Deallocation Type.

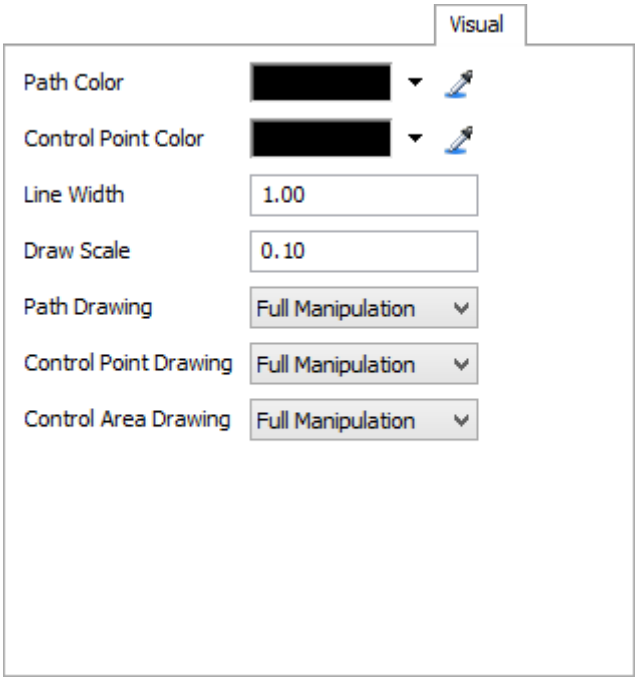
Edge Definition - This defines which "edge" of the AGV will determine the deallocation time. Options are:

- Center - Deallocation will be triggered when the AGV's center passes the given point.
- Trailing Edge - Deallocation will be triggered when the AGV's trailing edge passes the given point.
- Leading Edge - Deallocation will be triggered when the AGV's leading edge passes the given point.

Travel Point Definition - Defines the associated point on the path that determines deallocation time. Options are:

- Arrives at Next Control Point - Deallocation will be triggered when the AGV's defined edge arrives at the next Control Point. For Control Areas, this is the next Control Point after the path exits the Control Area.
- Passes Current Point - Deallocation will be triggered when the AGV's defined edge passes the current point. For Control Points this is the Control Point itself. For Control Areas, this is the point on the path where the AGV's defined edge exits the Control Area.

Plus Distance - This is an additional distance that will be added onto travel before deallocating the object.
Plus Time - This is an additional time to delay deallocation, after the defined edge has reached the travel point plus the distance.



You can get to the Visual page by right-clicking on a Path or Control Point and choosing AGV Network Properties.

Path Color - Defines the color by which Paths will be drawn in the 3D view. This is especially useful if you are overlaying your model with a CAD drawing, and want to delineate between AGV Network paths built in the model and CAD lines.

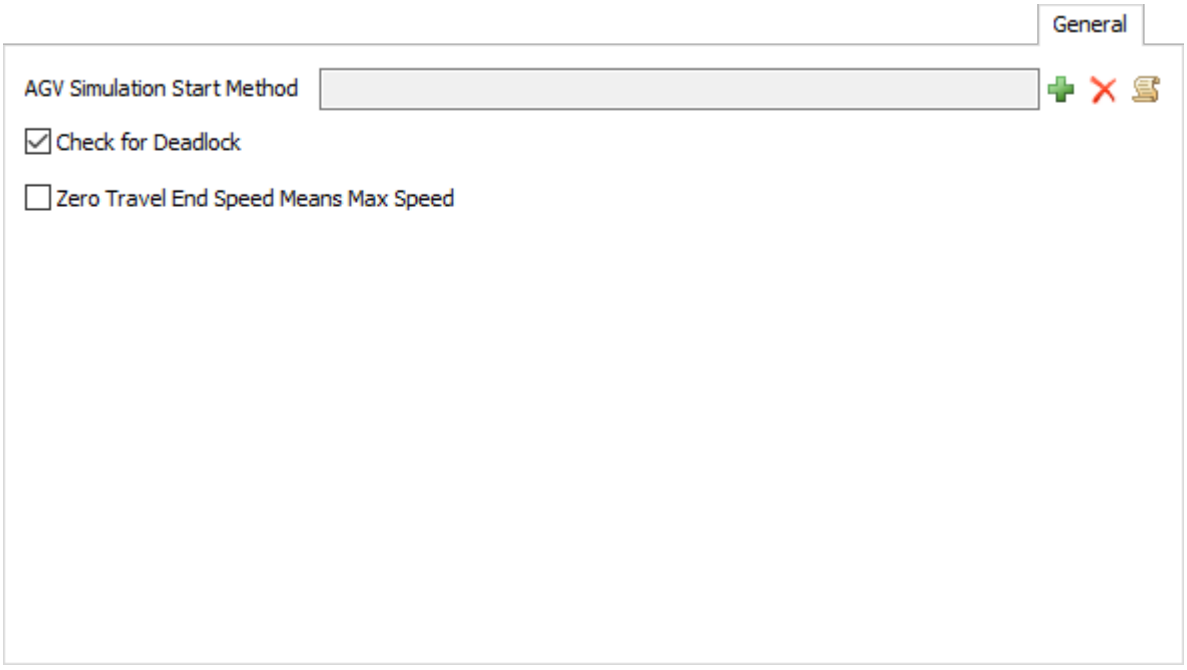
Control Point Color - Defines the color by which Control Points will be drawn in the 3D view.

Line Width - Defines a baseline width, in pixels, by which Paths and Control Points will be drawn in the model.

Draw Scale - Defines a baseline size by which to scale drawing of Control Points and Path direction arrows.

Path Drawing, Control Point Drawing, Control Area Drawing - Defines how the respective objects can be manipulated in the model. As you finish certain parts of your model you may want to restrict what you can change about the objects in it. Options are:

- Full Manipulation - You can click on and move these objects around as needed.
- Clickable Only - You can click on these object but you cannot move them around.
- Not Clickable - You can see the objects in the 3D view but you cannot click on them.
- Do Not Draw - You cannot see the objects in the 3D view.



You can get to the General page by right-clicking on a Path or Control Point and choosing AGV Network Properties.

AGV Simulation Start Method - A trigger that will fire for each AGV at the beginning of the model. Usually you will use this to fire the AGV's OnResourceAvailable, in order to start the AGV polling for work.

Check for Deadlock - If checked, the Control Point/Control Area allocation logic will continuously check for deadlock cycles. If it finds one, it will stop the model and notify you of the issue. Tutorial Lesson 1 includes a discussion of deadlock. Note that deadlock detection does require additional calculations and may slow your simulation down. You should hence turn it on while debugging, and once all deadlock issues are resolved, turn it back off.

Zero Travel End Speed Means Max Speed - If checked, AGV's will interpret travel tasks with end speed of 0 to mean: end at the AGV's max speed. This behavior is the default with other FlexSim travel mechanisms, such as standard travel networks that use network nodes. However, with AGVs this is not always the desired behavior, so it is an explicit setting that you define here.

The AGV module does not add its own AGV object type to the library. Instead you attach task executers to control points on an AGV network, and those task executers will travel using AGV-defined logic.

Events

Task executers who have been attached to an AGV network will have additional events that can be subscribed to with a process flow Wait For Event or Event-Triggered Source activity.

- OnStartTravel Fired when the AGV starts a travel task.
- OnFinishTravel Fired when the AGV finishes a travel task.
- OnPreAllocate Fired just before the AGV attempts to allocate forward. This will either be followed by one or more OnAllocate events or by an OnAllocationFailed event if it was not able to allocate forward. Allocation happens either on pre-arrival to a control point or when trying to allocate an intersection point on an accumulating path.
- OnAllocate Fired when a control point, control area, or accumulating path intersection point is allocated.
- OnAllocationFailed Fired when the AGV fails to allocate forward and hence must stop and wait.
- OnDeallocate Fired when the AGV deallocates a control point, control area, or accumulating path intersection point.
- OnAccumulationStop Fired when the AGV hits its proximity stop threshold on an accumulating path and must stop.
- OnAccumulationResume Fired when the AGV hits its proximity resume threshold on an accumulating path and can resume.
- OnPreArrival Fired at an AGV's pre-arrival to a control point, i.e. the point at which the AGV would start to decelerate to stop at the control point if needed. OnPreArrival is fired prior the AGV allocating further ahead, or when the AGV starts decelerating to its final destination.
- OnArrival Fired at an AGV's arrival at a control point, i.e. when the AGV has decelerated to stop at the control point, either because it could not allocate further ahead or if the control point is the final destination.

Statistics

AGVs can track the following statistics.

- BatteryLevel The AGV's battery level, as a percentage between 0 and 100.

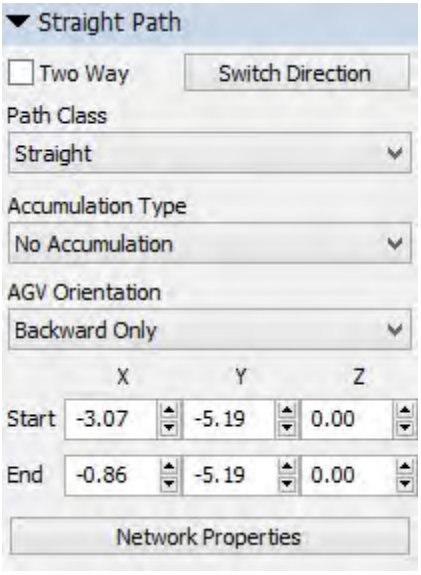
AGV Paths define the routes that AGVs take to get to their destinations on the AGV network. A detailed explanation of how to create paths can be found in Tutorial Lesson 1.

Path Properties

Paths do not have a traditional properties window. Instead you access their properties through Quick Properties.

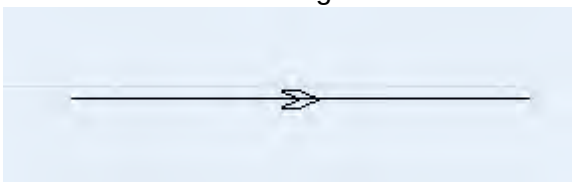
1. Click on the Path.

2. Edit the Path's properties in the Quick Properties window on the right.

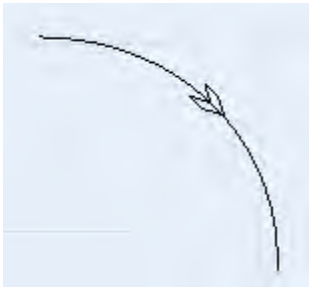


Straight vs Curved

A Path can either be a Straight Path

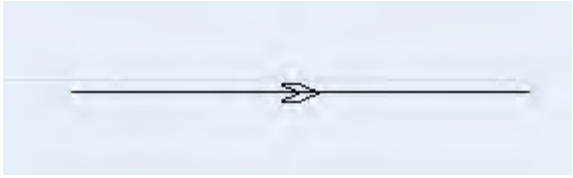


or a Curved Path

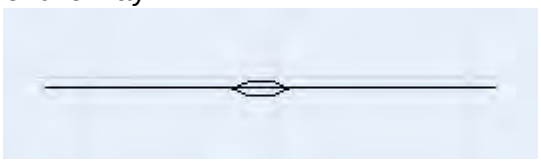


One-Way vs Two-Way

A Path can either be one-way

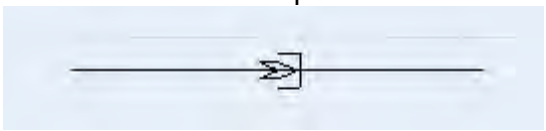


or two-way

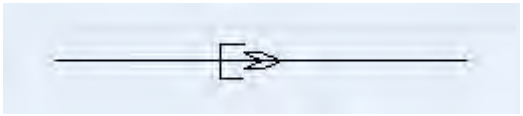


AGV Orientation

You can also force a single AGV orientation for a path. This defines how the AGV must face while on the path, irrespective of the AGV's travel direction. The default is Any, meaning the AGV may face any direction while on this path. You can define it as Forward-Only



or Backward-Only



Think of the half-square as an outline of the AGV's front end. Paths with a defined AGV orientation can be two-way, but the Forward/Backward-ness of the AGV Orientation is relative to the direction the Path would be if it were one-way.

Usually you can enforce proper AGV orientation simply by building the path network correctly. However, in some cases this is not possible, hence you have the option to enforce it explicitly. Note that choosing an AGV orientation on any path in your network will cause a non-trivial amount of extra memory to be allocated to build routing tables for each AGV orientation, and if you haven't built your network properly it may cause routing failures because the network can't find a path that will get the AGV to the destination with the proper orientation. Hence it is advisable to only use this option if it is explicitly needed.

Path Classes

Path Classes allow you to break out AGV speeds by path. A detailed conceptual explanation of Path Classes can be found in Tutorial Lesson 3.

Accumulation Types

Each path can have an Accumulation Type assigned to it. If defined (not No Accumulation), AGVs traveling on that path will detect proximity with each other and avoid collisions. You can define Accumulation Types in the AGV Network's Accumulation Types tab.

Control Point

Control Points are points on the AGV network where various decision logic happens.

A Control Point may be:

- A point where AGVs pick up and/or drop off flow items.
- A stopping point on the network where AGVs wait to enter an area or section of a path.
- A decision point on the network where the AGV looks for work to do.
- A decision point on the network where an AGV is dispatched to some other point on the network, dependent on network state/availability.

Control Point Allocation/Deallocation

Control Points act as allocation/deallocation points on the network. AGVs automatically look ahead to their next control point and must allocate that control point before proceeding on the network. A conceptual explanation of Control Point allocation/deallocation can be found in Tutorial Lesson 1.

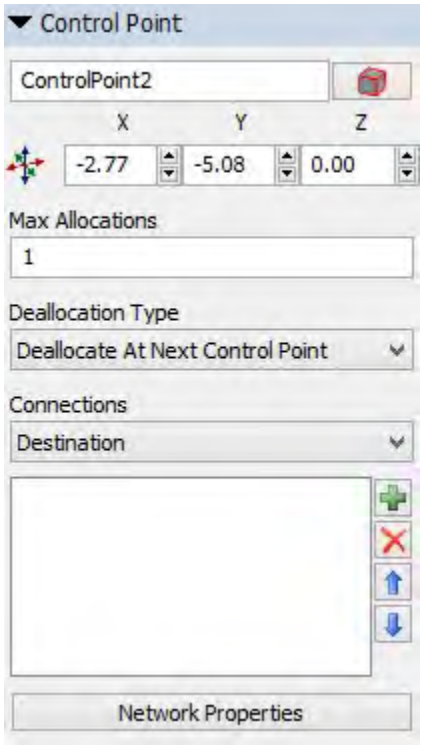
Way Points

Control Points can be members of Way Points. A conceptual explanation of Way Points can be found in Tutorial Lesson 2.

Control Point Properties

Control Points do not have a traditional properties window. Instead, properties are defined through the Quick Properties window.

1. Click on a Control Point in the model.
2. Edit its properties in the Quick Properties Control Point panel on the right.



- Max Allocations The maximum number of AGVs that can simultaneously claim the Control Point.
- Deallocation Type The Control Point's assigned Deallocation Type. See the Deallocation Types topic for more information.
- Connections Here you can view, add, remove, and re-order Control Point Connections. See the Control Point Connections topic for more information.

Events

You can subscribe to the following Control Point events, usually by using a process flow Wait For Event or Event-Triggered Source activity.

- OnPreArrival Fired at an AGV's pre-arrival to the control point, i.e. the point at which the AGV would start to decelerate to stop at the control point if needed. OnPreArrival is fired prior the AGV allocating further ahead, or when the AGV starts decelerating to its final destination.
- OnArrival Fired at an AGV's arrival at the control point, i.e. when the AGV has decelerated to stop at the control point, either because it could not allocate further ahead or if the control point is the final destination.
- OnAllocated Fired when the Control Point is allocated by an AGV.
- OnDeallocated Fired when the Control Point is deallocated by an AGV.
- OnEntry Fired when an object (usually a flow item) is moved into the control point.
- OnExit Fired when an object (usually a flow item) is moved out of the control point.

Statistics

The control point can track the following statistics.

- AllocationCount The total number of AGVs who have allocated the control point. Usually this will either be 0 or 1, unless you set the control point's Max Allocations to be greater than 1.

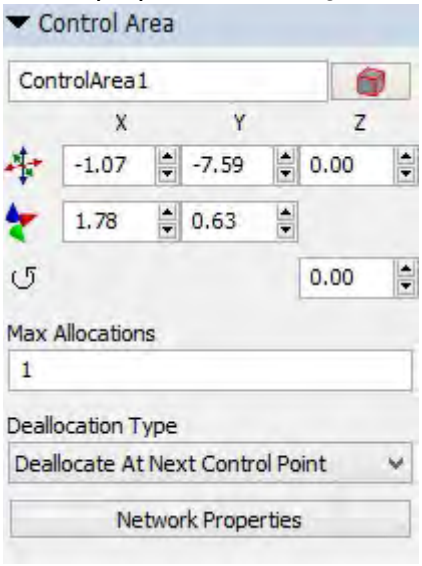
Control Area

A control area is an object that enforces mutual exclusion on one or more paths in the AGV network. A conceptual overview of control areas can be found in Tutorial Lesson 3.

Control Area Properties

Control areas do not have a traditional properties window. Instead, you edit their properties through the Quick Properties window.

1. Click on a control area.
2. Edit its properties in the Quick Properties Control Area panel on the right.



Max Allocations - The maximum number of AGVs that can simultaneously claim the control area. Deallocation Type - The control area's assigned deallocation type. See the Deallocation Types topic for more information.

Events

The control area shares some of the same events as the control point, including:

- OnAllocated
- OnDeallocated

Statistics

The control tracks the same statistics as the control point, including: •

AllocationCount

AGV-Specific Pick-List Actions

The AGV module includes several pick-list actions that are designed specifically to work with the AGV network.

Use Transport: Move Item via AGV

This can be applied in a FixedResource's Use Transport field. For an example of using the Move Item via AGV action, refer to Tutorial Lesson 3. Usually this will be used in conjunction with the Way Point's Look For Work and/or Wait For Work actions.

Search Start Point / For Each - These define the search scope for finding a Control Point to place the flow item for pickup. This is similar to a Way Point's Redirect Via Search scope, except that the base point is usually current, i.e. the FixedResource from which the flow item is being transported. The default starts at the origin FixedResource, and searches first the Control Points connected to it via the "Location" connection, and then from there enumerates all the Control Points in the "PickupPoints" connection.

Filter (WHERE) - Defines criteria for placing the flow item in a given Control Point. Similar to the Way Point's Redirect Via Search filter, but here you use controlPoint as the accessor instead of destination.

Prioritization (ORDER BY) - Defines prioritization. Again, this is similar to the Way Point's Redirect Via Search prioritization.

Task Queue Location - Defines where the generated task sequence will be stored. Usually this will be the "head" Control Point, i.e. the Control Point that has multiple PickupPoints connections.

Origin Control Point Transfer - Defines how the flow item should be moved to the origin Control Point. You can use a simple delay time, or you can transport it via another Operator/Transport.

Destination Transfer - Defines how the flow item should be moved to the final destination FixedResource once it's been dropped-off at the drop-off Control Point. You can use a simple delay, or have it be moved via an Operator/Transport, or just Do Nothing, in which case you would need to have some other trigger that initiates moving into the final destination.

OnResourceAvailable: Look For Work

The Look For Work action can be applied in a TaskExecuter/Dispatcher's OnResourceAvailable trigger. It is the same as a Way Point's Look For Work action.

OnResourceAvailable: Redirect Via Direct Reference

The Redirect Via Direct Reference action can be applied in a TaskExecuter/Dispatcher's OnResourceAvailable trigger. It is the same as a Way Point's Redirect Via Direct Reference action.

Index