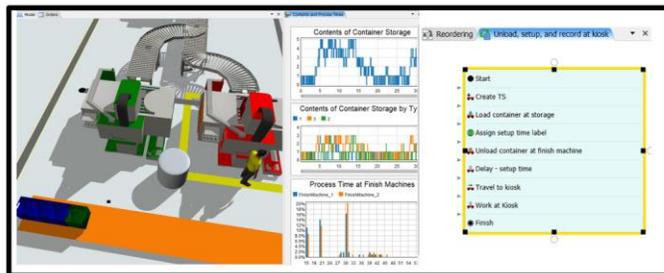
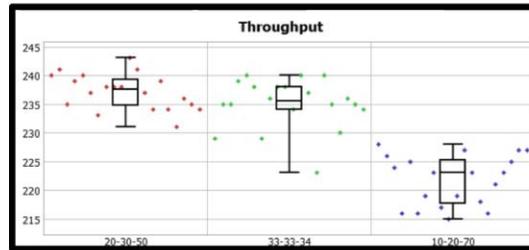
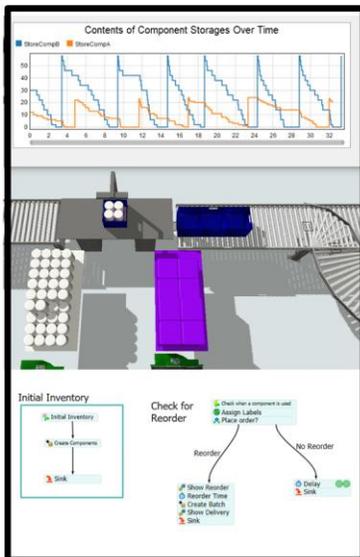
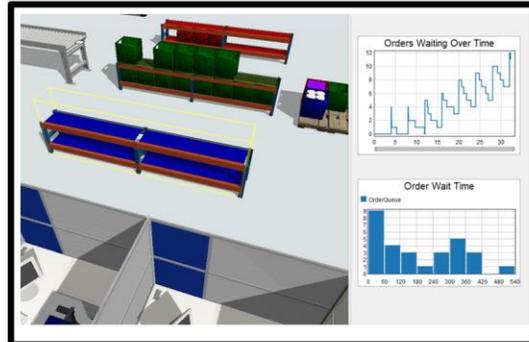




SIMULATION SOFTWARE PRIMER



Allen G Greenwood, Ph.D., P.E.

Simulation Education Specialist, FlexSim Software Products, Inc.

Professor Emeritus, Department of Industrial & Systems Engineering, Mississippi State University

Fourth Edition, 01 June 2020

Copyright © 2020 FlexSim Software Products, Inc. All rights reserved.

Published by FlexSim Software Products, Inc., Canyon Park Technology Center, Building A Suite 2300, Orem, UT 84097 USA.

FlexSim software and books may be purchased for educational, business, or sales promotional use. For more information, please contact our sales department: +1-801-224-2914 or sales@FlexSim.com.

The *FlexSim* logo is a registered trademark of FlexSim Software Products, Inc. Other registered trademarks belonging to third parties are used within this work. Where those designations appear in this book, and FlexSim Software Products, Inc. was aware of a claim, the designations have been printed in italics, caps, or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained here.

Version 1.1, for software version 2017 Update 2, May 2018

Version 2.0, for software version 2018 Update 2, September 2018

Version 3.0 for software version 2019 Update 2, December 2019

Version 4.0 for software version 2020 Update 1, June 2020

Supporting files and all *FlexSim* model files are available at <https://flexsim.com/primer-files>

ABSTRACT

This primer, using a single comprehensive example and sequential development process, provides detailed, step-by-step instructions for building and analyzing a comprehensive simulation model in the *FlexSim* simulation software. The system that is modeled is small, but not simple. While the focus is on basic concepts and constructs in *FlexSim*, the primer introduces some of the advanced features and capabilities that are available in *FlexSim*. The primer describes: (1) *FlexSim*'s basic fixed and mobile 3D objects (e.g., processors, conveyors, operators) that are powerful and yet easy to use, (2) the ability to build custom logic through the flowchart-like tool Process Flow, (3) other powerful tools such as tables, lists, reliability, and charts, (4) designing and running experiments with a simulation model to analyze the operational dynamics of a system, and (5) *FlexSim*'s array of modules for modeling AGVs, warehouses, people, etc. In addition to basic instructions, the primer provides insight and rationale for the described modeling actions and introduces good modeling and analysis practices.

The primer is not a software user manual that is primarily a reference document. The primer provides a means to learn the basics of *FlexSim* simulation software and to be introduced to its power and capabilities in a way that demonstrates how it relates to the simulation problem-solving process. The primer is primarily intended for someone who has little or no familiarity with *FlexSim* simulation software. However, in addition to new users, it is hoped that existing *FlexSim* users will find the primer useful to clarify some aspects of the software and to learn something new. While it is preferable that the reader is familiar with the basics of discrete-event simulation, that is not required.

Supporting files and all *FlexSim* model files are available at <https://flexsim.com/primer-files>

This page is intentionally blank.

PREFACE

This primer is based on many years of experience and my long-time passion for simulation. I started learning simulation in graduate school and applying it in practice in my first position as an industrial engineer in the mid-1970s. Simulation has been a major part of my professional life in a variety of settings and situations. These include teaching many simulation courses at the undergraduate and graduate levels at a number of universities, both in the US and other countries; developing and delivering short courses for practicing engineers in a variety of industries; writing, presenting, and publishing research papers on various aspects of simulation modeling and analysis; and, carrying out many simulation projects in a range of industries.

I consider simulation to be an applied technology that is an essential tool for effectively supporting a broad range of problem-solving and decision-making processes. It is especially valuable for the design and management of systems in many domains, such as manufacturing, material handling, warehousing, logistics, healthcare, mining, business processes, etc. All of these domains are naturally complex – the complexities are due to the systems having many disparate elements, the relationships among those elements, inherent variability, and intrinsic dynamics. Simulation facilitates and encourages the exploration and assessment of multiple ideas and alternatives in order to “optimize” system performance. All of this is done virtually without disturbing an existing system or before a system actually exists.

There are many good resources available to learn about simulation modeling and analysis in general and *FlexSim* in particular, including textbooks (such as *Applied Simulation Modeling and Analysis Using FlexSim*), training and teaching materials, user manuals, tutorials, blogs, videos, etc. Each resource has its own objectives and is devised to meet specific needs. However, none of the resources by themselves meet all of the primer’s objectives:

- provide detailed, step-by-step instructions for building a comprehensive simulation model,
- use a single comprehensive example and sequential development process to effectively move from introducing and applying simple concepts and methods to describing those that are more complex,
- provide insight and rationale for modeling actions, rather than just providing rote commands,
- introduce good modeling and analysis practices,
- introduce fundamental concepts of simulation; provide references for further reading, and
- provide an awareness of some of the more advanced features available in *FlexSim* without covering the details at that time.

This primer focuses on the modeling and analysis of operations systems in order to understand and analyze their dynamic behavior and performance (referred to as *operations dynamics*) using discrete-event simulation. In the broadest sense, operations systems transform input into output through a set of related activities and processes that require a variety of resources, such as equipment, material, people, and information. Transformations may either be tangible (e.g. machining, inspecting, or delivering material in manufacturing) or intangible (e.g. diagnosing or treating patients in healthcare).

Simulating operations dynamics requires an accurate representation of the system being considered, including both the physical aspects (locations, distances, sizes, speeds, etc.) and the salient underlying logic (e.g., what action to take, when and where to perform the action, using what resources). Visualization of the behaviors

resulting from the interaction of the physical and logical aspects greatly increases the acceptance of modeling and analysis by decision makers and those that are not experts in simulation technologies and methodologies. Visualization also enhances model validation and verification. *FlexSim* has rich visual and logical functionality and is easy to use. Of course, ease of use is relative, depending on a user's background and experience.

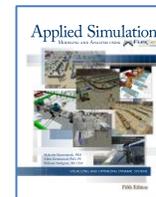
Since *FlexSim* is so comprehensive, it is not feasible to cover all of its capabilities in this introductory guide. Similarly, the field of simulation is extensive; therefore, the theory and methods of simulation modeling and analysis is beyond the scope of this basic primer. Therefore, for an introduction to simulation, the following primer is suggested. (The author of this primer is also author of the simulation primer.)

Greenwood, A. *Simulation Primer*, FlexSim Software Products, Inc., 2019.



For a more detailed discussion of simulation modeling and analysis concepts and practices, and how they can be implemented in *FlexSim*, the following textbook is suggested. (The author of this primer is a coauthor of the textbook.)

Beaverstock, M., Greenwood, A., and Nordgren, W. *Applied Simulation Modeling and Analysis Using FlexSim*, 5th Edition, FlexSim Software Products, Inc., 2017.



Other sources of information on *FlexSim* include: the *FlexSim User Manual*, in-software links to the manual, tutorials, and web resources, especially *FlexSim Answers*, which is a shared searchable knowledge base of questions and answers from FlexSim's worldwide community of users.

It is important to note that this primer is not another user manual for the software. User manuals are good reference resources to describe specific aspects of the software; whereas this primer is a means to learn both the basics of *FlexSim* simulation software and to be introduced to its power and capabilities in a way that demonstrates how it relates to the simulation problem-solving process.

It is suggested that this primer be covered carefully and methodically in the order it is written. The material builds from very simple aspects of simulation and modeling to the more complex. However, someone with more experience with simulation and the software may skim the early material, but it still should be considered since, at a minimum, it provides context. Later sections then may be considered in more detail to enhance understanding of a concept or software capability. Since all of the models described in the primer are available online (<https://flexsim.com/primer-files>), more experienced readers can start at a later section without having to build all of the previous models. Of course, providing all of the models also benefits those new to the software.

Allen G Greenwood

revised December 2019, Greensboro, NC USA
September 2018, Blowing Rock, NC USA

TABLE OF CONTENTS

ABSTRACT 3

PREFACE..... 5

TABLE OF CONTENTS 7

INTRODUCTION..... 11

 OBJECTIVES..... 11

 STRUCTURE 11

 APPROACH..... 12

TESTIMONIALS..... 13

 WHY *FLEXSIM* SIMULATION SOFTWARE? 13

 THE ROAD TO CHOOSING *FLEXSIM*..... 15

FLEXSIM SIMULATION SOFTWARE 17

PART 1 – THE BASICS OF 3D MODELING AND ANALYSIS 19

1 FUNDAMENTAL CONCEPTS 20

 1.1 SIMULATION MODELING AND ANALYSIS..... 20

 1.2 SIMULATION USING *FLEXSIM* 21

 1.3 OBJECT LIBRARY 21

 1.4 TOOLBOX..... 22

 1.5 ANALYSIS 22

 1.6 NOTATION AND FORMATTING CONVENTIONS 23

2 DESCRIPTION OF THE SYSTEM TO BE MODELED..... 25

 2.1 OVERVIEW..... 25

 2.2 MODELING APPROACH AND BASIC SYSTEM DESCRIPTION 26

3 FLEXSIM’S MODELING ENVIRONMENT 28

 3.1 TIME AND DISTANCE UNITS..... 29

 3.2 MAIN MENU AND MAIN TOOLBAR 30

 3.3 MODEL EXECUTION TOOLBAR..... 31

 3.4 3D MODEL VIEW WINDOW, MODELING SURFACE, AND MOUSE OPERATIONS 31

 3.5 OBJECT LIBRARY AND TOOLBOX 33

 3.6 OBJECT INTERFACE 33

 3.7 QUICK PROPERTIES PANEL..... 35

- 3.8 HELP..... 35
 - 3.8.1 *User Manual*..... 35
 - 3.8.2 *Context Help*..... 36
 - 3.8.3 *Answers*..... 36

- 4 GETTING STARTED WITH THE SIMPLEST MODEL..... 37

- 5 BASIC FIXED RESOURCE OBJECTS AND CUSTOMIZATION41
 - 5.1 BASIC OBJECT PROPERTIES AND STRUCTURE 41
 - 5.2 SOURCE OBJECT..... 44
 - 5.3 FLOWITEM BIN 47
 - 5.4 QUEUE OBJECT..... 48
 - 5.5 PROCESSOR OBJECT 48
 - 5.6 SINK OBJECT 50

- 6 BASIC MODEL OUTPUT.....51
 - 6.1 OBJECT STATISTICS AND QUICK PROPERTIES..... 52
 - 6.2 DASHBOARD..... 54

- PART 2 – NEXT STEPS IN 3D MODELING.....61

- 1 CHANGING 3D GRAPHICS..... 62
 - 1.1 IMPORTING A LAYOUT 62
 - 1.2 CHANGING OBJECT GRAPHICS 65

- 2 ALTERNATIVE ROUTINGS 69
 - 2.1 CHOOSING A ROUTE BASED ON AVAILABILITY 69
 - 2.2 CHOOSING A ROUTE BASED ON CURRENT SYSTEM CONDITIONS 70

- 3 TASK EXECUTERS.....71
 - 3.1 BASIC TASK EXECUTER CONCEPTS 71
 - 3.2 ADDING A FINISH OPERATOR TO THE MODEL 72
 - 3.3 CONTROLLING TASK EXECUTER TRAVEL WITH PATH NETWORKS 76

- 4 CONVEYORS..... 80
 - 4.1 STRAIGHT AND CURVED CONVEYOR SECTIONS 81
 - 4.2 CONNECTING CONVEYOR SECTIONS..... 83
 - 4.3 JOIN CONVEYORS OBJECT..... 84
 - 4.4 CONVEYOR TYPES..... 85

- 5 GLOBAL TABLES 87
 - 5.1 USING TABLES TO STORE PROCESS TIMES 88
 - 5.2 USING TABLES TO STORE PRODUCT MIX PERCENTAGES..... 89

6	DOWNTIME.....	91
6.1	TIME TABLES.....	91
6.2	RELIABILITY.....	95
6.3	CONSTANT MTTF/MTTR; MTBF BASED ON CLOCK TIME; NO RESOURCE FOR REPAIR.....	95
6.4	RANDOM MTTF/MTTR; MTBF BASED ON SYSTEM STATES; RESOURCE FOR REPAIR.....	97
6.5	CHARTING THE EFFECT OF DOWNTIME.....	99
 PART 3 – MORE 3D MODELING + ANALYSIS.....		103
1	COMBINING AND SEPARATING FLOWITEMS.....	104
1.1	SCHEDULED ARRIVALS OF COMPONENTS.....	104
1.2	MODELING THE PACKING OPERATION USING THE COMBINER OBJECT.....	108
1.3	DOWNTIME AT THE PACKING OPERATION.....	113
1.4	BATCHES OF COMPONENTS UNLOADED BY THE OPERATOR USING SEPARATOR OBJECTS.....	115
2	CONTROLLING TASK EXECUTER TRAVEL PATHS USING A*.....	123
2.1	THE BASICS OF THE A* ALGORITHM USING A SIMPLE STUDY MODEL.....	123
2.2	IMPLEMENTING THE A* ALGORITHM IN THE PRIMER MODEL.....	127
2.3	SAVING MODEL VIEWS.....	131
3	USING AN ITEM LIST FOR COMPLEXMODELING LOGIC.....	132
3.1	SIMPLE PRIORITY ROUTING USING AN OBJECT TRIGGER.....	132
3.2	MULTIPLE-CRITERIA ROUTING USING LISTS.....	133
4	EXPRIMENTATION IN <i>FLEXSIM</i>.....	139
4.1	EFFECT OF BUFFER SIZE ON PERFORMANCE.....	139
4.2	EFFECT OF PRODUCT MIX ON PERFORMANCE.....	143
5	SUMMARY OF THE PRIMER FOR <i>FLEXSIM3D</i>.....	147
 PART 4 – INTRODUCTION TO PROCESS FLOW.....		149
1	BASIC CONCEPTS.....	150
1.1	BASIC PROCESS FLOW CONCEPTS AND MODELING ENVIRONMENT.....	150
1.2	USING PROCESS FLOW TO MODEL INVENTORY POLICY.....	152
2	MODELING INITIAL INVENTORY.....	153
3	MODELING INVENTORY REORDER POLICY.....	158
3.1	CHANGES IN 3D OBJECTS FOR USE IN PROCESS FLOW.....	158
3.2	IMPLEMENTING REORDER-POINT INVENTORY LOGIC USING PROCESS FLOW.....	160
3.3.1	“CHECK FOR REORDER” LOGIC.....	160

3.3.2 “REORDER” LOGIC 162

3.3.3 “NO REORDER” LOGIC..... 164

4 CUSTOM TASK SEQUENCES IN PROCESS FLOW.....166

4.1 BASE MODEL MODIFICATIONS 166

4.2 SIMPLE STUDY MODEL OF TASK SEQUENCES IN PROCESS FLOW 168

4.3 CUSTOM TASK SEQUENCES IN THE PRIMER MODEL..... 174

PART 5 – WAREHOUSING, ORDER PROCESS AND ADDITIONAL FEATURES 181

1. INTRODUCTION TO WAREHOUSE MODULE.....182

2. ORDER PROCESSING SUBMODEL.....185

2.1 DEFINITION OF THE ORDER-FULFILLMENT PROCESS 185

2.2 IMPLEMENTATION OF THE ORDER-FULFILLMENT PROCESS..... 185

3 OTHER FLEXSIM OBJECTS AND CAPABILITIES194

3.1 MULTIPROCESSOR OBJECT 194

3.2 TRANSPORTER OBJECT..... 195

3.3 ROBOT AND CRANE OBJECTS 195

3.4 FLUIDS LIBRARY..... 196

4 SUMMARY OF THE PRIMER MODEL197

EPILOGUE200

APPENDIX - LIST OF MODELS.....202

APPENDIX - GLOSSARY207

ABOUT THE AUTHOR214

Supporting files and all *FlexSim* model files are available at <https://flexsim.com/primer-files>

INTRODUCTION

This section provides the primer's objectives, organization, and general approach.

Objectives

As stated in the Preface, the objectives of this primer are to:

- provide detailed, step-by-step instructions for building a comprehensive simulation model in *FlexSim*,
- use a single comprehensive example and sequential development process to effectively move from introducing and applying simple concepts and methods to describing those that are more complex,
- provide insight and rationale for modeling actions, rather than instructing rote commands,
- introduce good modeling and analysis practices,
- introduce fundamental concepts of simulation; provide references for further reading, and
- provide an awareness of some of the more advanced features available in *FlexSim* without covering the details at that time.

Structure

In order to meet these objectives, the primer is divided into the following five parts:

1. *The Basics of 3D Modeling and Analysis* that include a brief introduction to the following: fundamental simulation concepts, description of the system to be modeled, overview of *FlexSim*'s modeling environment, how to build a very simple model, fundamentals of *FlexSim*'s fixed resource objects and customization, and introduction to output from *FlexSim* models.
2. *Next Steps in 3D Modeling* continues through the following topics: changing 3D graphics, incorporating alternative routing rules, introduction to Task Executors or dynamic objects, basics of modeling conveyors, the use of Global Tables for data management, and incorporating planned and unplanned downtime into *FlexSim* models.
3. *More 3D Modeling and Analysis* discusses the following topics: combining and separating flowitems, controlling Task Executors using the A* algorithm, using Lists to represent complex routing logic, and running experiments in *FlexSim*.
4. *Introduction to Process Flow* begins with a discussion of the basic concepts of the logic builder and then provides two examples of building custom logic – one for creating inter-object logic to implement a reorder-point inventory policy and the second for creating a custom task sequence for a mobile resource (Task Executor).
5. *Order Fulfillment Sub-Model, Warehousing, and Additional FlexSim Features* introduces *FlexSim*'s Warehousing Module and adds logic to the primer model that processes incoming orders and completed items from production. This part also briefly introduces some of additional capabilities in *FlexSim*, including other 3D objects and the Fluids Library.

The main portion of the primer concludes with an *Epilogue* that provides a brief summary and possible next steps.

The primer also includes two Appendixes – a *List of Models* and a *Glossary*.

- The list of models provides a brief description of the concepts and software features that are used in each of the primer models, as well as a reference to the section in the primer where the model is discussed.
- The glossary provides definitions of key terms used in the primer.

The primer concludes with a brief biography of the author and his contact information.

In the following section, prior to the main parts of the primer, two common questions are addressed that often arise from those who are unfamiliar with *FlexSim* simulation software.

- Why should I use *FlexSim*? In order to help address this question, two long-time users of *FlexSim* provide testimonials as to why *FlexSim* is their simulation software of choice and their path to choosing *FlexSim*. One testimonial is from an academic perspective and the other is from an industry perspective.
- What is *FlexSim* and what are its basic capabilities? In order to help address this question, a brief overview of the software is provided along with information on how to acquire the software.

Approach

As mentioned in the objectives above, the primer introduces and explores *FlexSim* through a single model. The model is small, but not simple. Its complexity evolves throughout the primer as various aspects of the simulation software are presented.

Since the model of the example system evolves throughout the primer, it is recommended that each iteration of the model be saved. After each save, it is suggested that the current model file be copied and renamed so that one can always revert to a previous version of the model. This is just good modeling practice – models should always be developed in steps – build, test, and validate – and all models should start simple and then complexity is added as needed. That is, the final model evolves from a number of simpler models with increasing representation of the real system. One should never try to incorporate all capabilities at the beginning of the modeling process.

In order to facilitate learning, all of the models developed in the primer are available so that if a reader has a problem developing any one of the primer models, they can access and view the correct implementation and then proceed on with the primer. Having access to the primer models also allows a reader with some knowledge of *FlexSim* to skip the simpler parts of the model building process and start at any section of the primer. For example, if a reader is well versed in the 3D aspects of *FlexSim*, but is not familiar with Process Flow, then they can start the primer in Part 4 and begin with the model from the end of Part 3.

The models, along with supporting files (a layout file and two 3D representations of work areas in the example) are provided in a resources folder that is available at <https://flexs.im/primer-files>.

TESTIMONIALS

This section provides two testimonials, one from an academic perspective and one from an industry perspective, that discuss why *FlexSim* is their simulation software of choice and their path to choosing *FlexSim*. Each testimonial is from a long-time user of *FlexSim* and co-author of the textbook, *Applied Simulation Modeling and Analysis Using FlexSim*.

Why *FlexSim* simulation software?

Provided by: Allen G Greenwood, Ph.D., P.E.
Professor Emeritus, Mississippi State University
Simulation Education Specialist, FlexSim Software Products, Inc.

There are a number of simulation software products available in the marketplace; some have been around for many years, while others are quite new. A common question, both in industry and academe, is why should *FlexSim* be used, instead of any of the other software that are available?

I believe that the choice of a software tool should align with a person's general approach to modeling and simulation. Therefore, I briefly introduce my background in order to give context and perspective to my response to the posed question. I have been involved with simulation for over 40 years – using it to solve problems in a variety of industries and teaching numerous simulation courses, both in the U.S. and abroad. Over the years I have used numerous simulation software products in industry projects, to support research, and in the classroom. I started using *FlexSim* in 2006 for industry projects and then, a few years later, transitioned my simulation courses to use *FlexSim*. Since then, I have co-authored the textbook, *Applied Simulation Modeling and Analysis using FlexSim* (now in its fifth edition) and recently authored two primers, one focused on using *FlexSim* software and one on simulation in general. One of my long-term, foundational professional goals has been to enhance and increase the application of simulation, both in practice and in academe, to support problem solving and decision making.

While there are many good simulation products available in the marketplace, I find *FlexSim* to be the best for my approach to solving problems and for educating others. Hopefully this essay will articulate the reasons why *FlexSim* became my simulation software of choice.

FlexSim is a modern, comprehensive simulation modeling and analysis environment that supports problem solving and decision making in industry, as well as supports teaching and research in academe. However, it was *FlexSim* - more than any other software - that made it possible for me to focus on solving industrial problems or on teaching simulation concepts. It is a software tool that is extremely powerful, yet easy to use and easy to transition, as needed, from small, simple models to those that are large and complex. This is important in practice for incremental model building – starting simple and adding complexity as needed – as it allows all those involved in the simulation process to follow and understand the underlying approach. It is especially important in teaching as I am able to have students quickly build simple models that illustrate key concepts, and then easily add desired enhancements. *FlexSim* makes it possible for me, as well as for students, to

concentrate on solving problems or learning simulation concepts, rather than dealing with the idiosyncrasies and learning curve associated with software.

Being involved in various industries and trying to give my students a basic understanding of the applicability and power of simulation, I find that *FlexSim* can easily model a wide range of operational systems including, but not limited to, discrete-part and continuous (e.g. oil and gas, food processing) manufacturing, transportation, logistics, healthcare, mining, construction, business and service processes, etc. *FlexSim* also offers a variety of simulation modeling approaches. While the core technology is discrete-event simulation, continuous and hybrid systems can be modeled through a library of pre-defined, yet customizable, fluid objects. Agent-based simulation and Monte Carlo simulation can also be performed within *FlexSim*.

Standard discrete-event and continuous modeling functionality are implemented through easy-to-use 3D objects. Most importantly, they can be customized and have common and intuitive user interfaces. The processing logic and behavior, as well as the appearance, of the objects are modified through extensive drop-down menu lists and direct parameter specifications. An intuitive, flowchart-like logic builder is available for defining and refining inter-object and intra-object relationships and behaviors. While some software claim they do not require the use of detailed computer code, there are times when it is necessary. Many of the simulation projects that I have been involved with during my career required representing complex logic and behaviors. I found that, when needed, *FlexSim's* process logic builder (Process Flow) and inherent scripting language *Flexscript* (a subset of C++) are effective and convenient means for representing complex operations. Numerous commands and templates, i.e. prebuilt sets of code and logic, are available to facilitate customization and reduce the amount of programming that is required. *FlexSim's* open, object-oriented, hierarchical software architecture also makes it a powerful and effective modeling environment.

FlexSim models are built in a native 3D environment. For me, their effective animation facilitates model validation, enhances stakeholder communication and understanding, and increases confidence in modeling and analysis. From a student perspective, the models look realistic, which provides needed relevance and helps make simulation exciting. In addition to the objects, I regularly use the many powerful modeling tools that are available in *FlexSim*, such as dashboards, time tables for managing work schedules, reliability tables for managing downtimes, global data tables, global variables, lists, macros, animation editor, video and model flythrough support, etc.

The simulation of an operation is only as good as the model's representation of the system and the analyses that are performed using the model. I believe this is important fundamental for students to realize. One of the reasons that I like *FlexSim* is that it provides a comprehensive and easy-to-use set of analysis tools, such as an experimenter for running multiple scenarios and replications simultaneously in a designed experiment (while taking advantage of the number of cores on your computer), statistical measures of performance, extensive plotting and charting capabilities, etc.

Simulation is only one tool for problem solving; and thus, communication with other supporting software is critical. For example, *FlexSim* provides convenient means to transfer model data both to and from spreadsheets and databases. *ExpertFit* is available to support modeling randomness (selecting probability distributions). Also, *FlexSim* provides a seamless interface to *OptQuest* for intelligently searching for the best solution, i.e. the best set of operating conditions for the system being modeled.

Another very important characteristic of simulation software is the company behind the product. FlexSim Software Products, Inc. was founded in 1993 and the first version of the *FlexSim* software was released in 2003. Customer support – for both industrial and educational users – is excellent. They truly want to help everyone effectively solve problems through the use of simulation. The company continuously improves their software through new and improved modeling and analysis capabilities, more effective interfaces and methods, better graphics, etc. FlexSim strongly supports education through student and education licenses and a variety of learning materials.

I hope this essay provides a start for understanding why I use *FlexSim* and perhaps why you should use *FlexSim* to enhance problem solving and decision making, provide an effective foundation for teaching simulation, and support research. Of course, the best way to be convinced that *FlexSim* is the software for simulation is to give it a try. Therefore, visit flexsim.com and contact FlexSim about how to get started simulating using *FlexSim*.

The road to choosing *FlexSim*

Provided by: Mal Beaverstock, Ph.D.
Manger of Business Simulation, General Mills (retired 2008)

Over a span of 12 years, more than \$150 million in savings were attributed to simulation at General Mills, Inc. (GMI). Now retired, I look back and realize that it was achieved through maintaining a problem-solving approach to simulation and to the tool that made it possible – *FlexSim*.

Early in my career, I lost faith with modeling and simulation. Attempts to use this technology to solve problems met with frustration and little results. Computing technology in the late 1960's made detailed mathematical models expensive, cumbersome, time consuming, and widely inaccurate. Problems were actually solved sooner by the involved engineers before a modeling approach could gain any momentum.

The change started in the early 1990's when a co-worker at International Paper solved a problem by modeling the mechanics of a machine aided by software that greatly facilitated building differential equations. The speed and ease of using such a tool re-ignited my interest in modeling and simulation.

While managing a control and simulation group at General Mills in 1995, an *Extend* simulation showed a savings of over \$1M by eliminating a packing line in a new design. That got management's attention! However, the work was still slow going. Using simulation software meant spending more time working with their idiosyncrasies than thinking about the problem that had to be solved.

Despite the drawbacks, successful simulations still helped by: increasing the productivity of a complex conveyor system; resolving design issues when sizing surge and storage tanks; and improving a batch operation to meet new production levels without any capital expense. However, the use of simulation remained a specialty for a few individuals and not a general tool.

A search began for a software tool that: could be accepted and used by all levels of individuals interested in simulation; provided for easy communication of assumptions and results; allowed a focus on the process dynamics; was flexible enough to handle complex functionality; and was a fully supported commercial product.

Simulations were developed using a variety of software including: *Extend*, *Witness*, *ProModel*, *Arena*, and *Excel*. None matched the requirements.

In 2002 General Mills acquired Pillsbury where *Taylor ED* was used for simulation and through their contacts, I learned about *FlexSim*. This newly-developed software showed great potential. It was based on their incorporating object-oriented software constructs – an approach I immediately recognized from my work some years earlier as head of Systems Research at the Foxboro Company. Their use of functional objects combined with state-of-the-art graphics was exceptional. It was the first time I saw such techniques applied to building a simulation tool and it set them apart from the rest of the field.

Bill Nordgren, President of FlexSim Software Products, Inc., had a similar vision to mine for a simulation tool. The resulting symbiotic relationship resulted in a product that met our functional requirements. A graduate student majoring in art at the University of Minnesota spent a summer creating 3D objects based on actual pieces of manufacturing equipment as well as objects representing General Mills products. The object interfaces were modified to reflect GMI terminology and contain design information about equipment such as standard speed, costs, and reliability information. Stored in libraries, equipment could be inserted into a simulation model without any other programming. While looking like a custom tool, the simulation program was actually a library within standard *FlexSim* and could be changed back with a single click. Since *FlexSim* served as the functional base, complex logic and dynamic simulations were possible.

As a result of *FlexSim*, by 2004 all interested individuals, including engineers, managers, and production personnel on the plant floor, could interact and contribute to the building, running, and analysis of the simulations that were helping them. Because of the visualization, people could see **their** equipment making **their** products.

The interest in simulation increased exponentially. Simulations were built 30% faster with *FlexSim*. Capital projects using simulation came in on time and under budget. *FlexSim's* flexibility allowed for simulations to: quickly provide a feasibility estimate for new manufacturing concepts; help in the design of new equipment by analyzing operator actions; optimize material prep operations; study personnel assignments and requirements; determine yogurt scheduling approaches; resolve problems in delivery and shipping areas; review scheduling impacts on entire production lines, and many others. When more complex logic was required, the consistent *FlexSim* environment made the simulation effort easier to develop and understand.

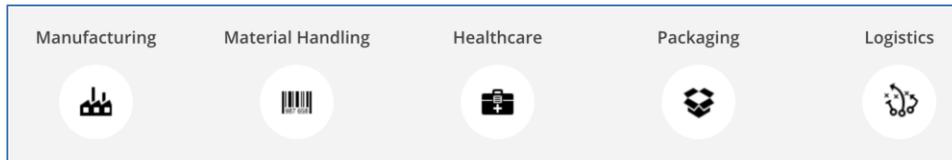
While the innovative software design, development execution, and tool-like approach to *FlexSim* was a major factor in my selection, the vision, responsiveness and commitment of the entire FlexSim staff sealed the deal. As far as I was concerned, the decision to use *FlexSim* was a “no-brainer” and proved itself in practice.



FLEXSIM SIMULATION SOFTWARE

FlexSim is a powerful, yet easy to use, environment for developing and analyzing simulation models of complex operations systems. Some of its features and capabilities include:

- comprehensive libraries of modeling objects and tools,
- modeling directly in 3D,
- complementary means to develop simulation models, such as customization of standard objects via consistent intuitive interfaces, developing complex process logic through a flowchart-like logic builder, and specifying behavior through custom coding with a scripting language that is a subset of C++,
- open, object-oriented, hierarchical architecture,
- links to spreadsheet, database, statistical, optimization, and graphics software,
- experimenter for designing, executing, and analyzing multiple scenarios and direct connection to *OptQuest* for optimization, and
- general modeling environment to support problem solving and decision making in diverse domains and applications.



- ...

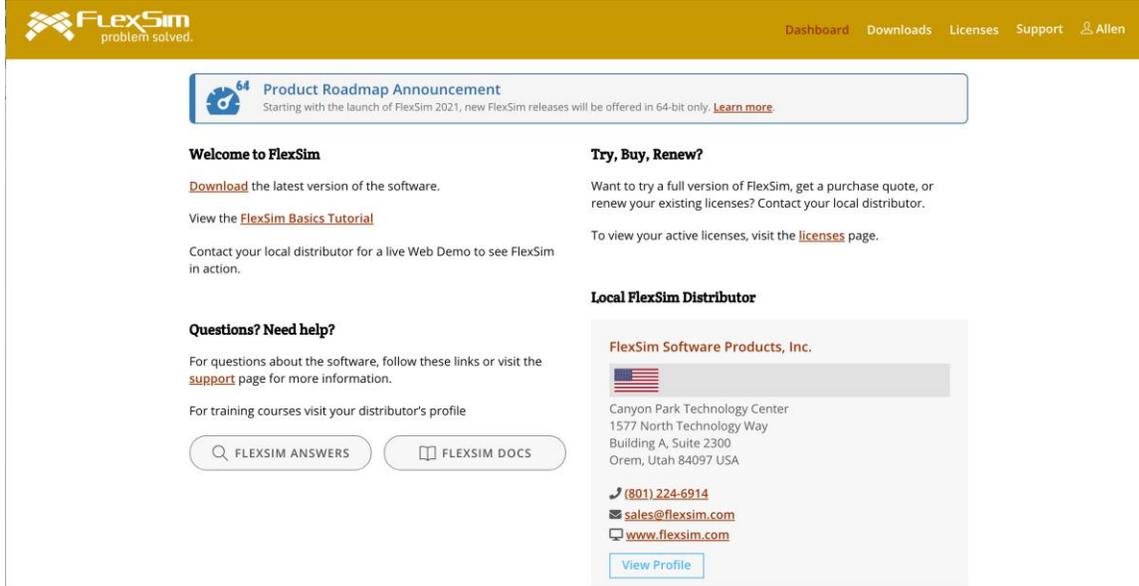
This primer is based on the *Enterprise* or *Education* versions of *FlexSim*, each of which requires a license. While *FlexSim Express* is available for free download from flexsim.com, it is primarily for evaluation purposes; and as a result, it is limited in the number of objects that can be used in a model and lacks key features of *FlexSim*, such as full customization and the Experimenter.

While FlexSim Software Products provides access to all versions of its software on its website, flexsim.com, it promotes two versions of *FlexSim* at any time - Current and LTS (Long-Term Support). At the time of writing this primer, Current is version 20.1.2 (release date 2020-05-08) and LTS is version 20.0.6 (release date 2020-05-08). A major release occurs at the end of the previous year, e.g. version 2020 (20.0.0) was released in late December 2019. Two updates with new features and capabilities occur twice a year, Update 1 is released in the March/April timeframe (e.g., 20.1.0) and Update 2 is typically released in the July/August timeframe (e.g., 20.2.0).

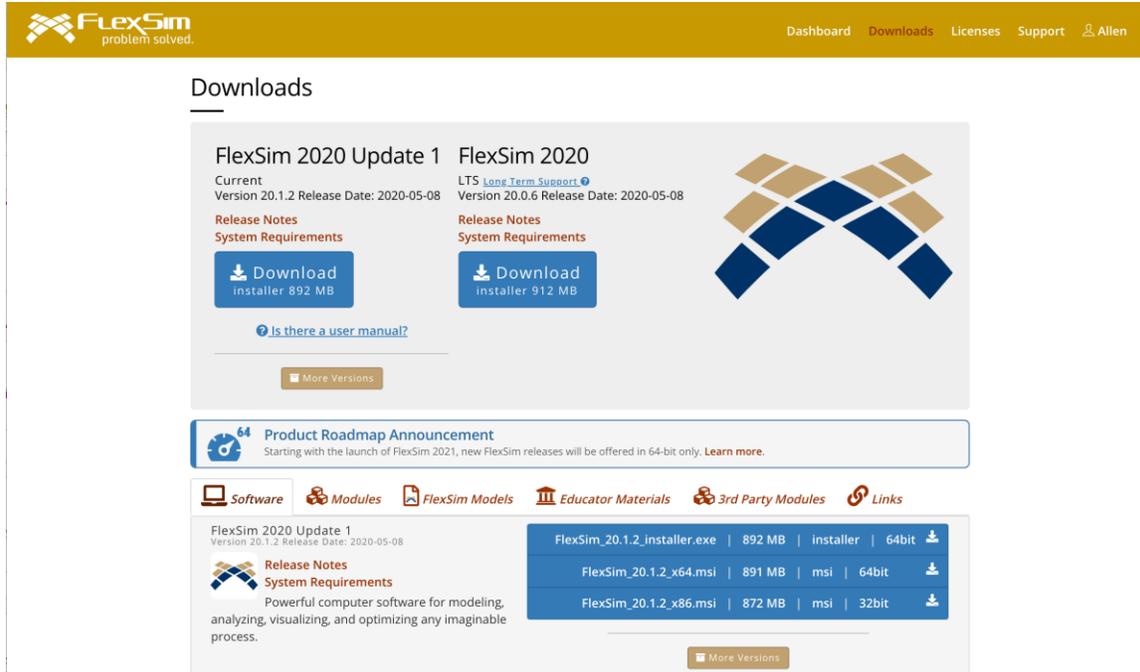
The current version of the textbook *Applied Simulation: Modeling and Analysis Using FlexSim* 5th Ed. (2017) is based on the version 2017.0.13.

This primer is based on the Current version - 2020 Update 1 (20.1.2, release date 2019-05-08). While many of the basic operations of the software and user interfaces are the same or very similar, there may be differences with other versions.

To obtain this version, log into your FlexSim account. The following dashboard will display once logged in.



Select **Downloads** from the main menu, resulting in the following interface.



To download the current version, select Downloads; for earlier, then click on **More Versions** (located just below the Download button for the Current version). The installer for all previous versions of *FlexSim* are available for download.

PART 1 – THE BASICS OF 3D MODELING AND ANALYSIS

The first part of the primer introduces:

- fundamental simulation modeling and analysis concepts,
- notation and format used in this primer,
- *FlexSim's*
 - overall structure
 - general modeling environment
 - basic fixed resource modeling objects
 - simple output from a simulation model
- description of the system being modeled. The same system is used throughout the primer and the model of that system evolves from a very simplistic representation of the system to one that more closely represents how the actual system might behave.

1 FUNDAMENTAL CONCEPTS

This primer begins with a short introduction to simulation modeling and analysis, an overview of *FlexSim*'s general approach to simulation, and the notation and formatting conventions used in this document.

1.1 Simulation modeling and analysis

Simulation is used for analyzing and solving problems and to support decision making. It is used to:

- understand a system's behavior, especially its dynamics,
- analyze and predict a system's performance,
- compare alternatives for improvement, and
- make the best decision for change.

Simulation is composed of two key parts, modeling and analysis:

- Simulation **modeling** is the ways and means for representing a system physically and logically in order to understand its behavior over space and time and to assess possible consequences of actions, virtually.
- Simulation **analysis** is the ways and means of using a simulation model to experiment with, and test, ideas and alternatives before deciding actions and committing resources.

While many things can be simulated, the focus of this primer, and the focus of *FlexSim*, is the simulation of operations systems – systems that transform input into output through a set of related activities and processes requiring a variety of resources, such as equipment, material, people, and information. Transformations may either be tangible (machining, inspecting, or delivering material in manufacturing) or intangible (e.g. diagnosing or treating patients in healthcare).

In order to simulate **operations systems**, three key aspects must be addressed: **interactions**, **variability**, and **dynamics** since these are all inherent in operations systems. There are complex relationships among a system's resources (material, equipment, people, and information) and the resources interact in numerous and complex ways. In even the simplest system, there are many sources of variability – arrival times and rates, process times, product mix, downtime, and quality level – to name a few. Some sources of variability are known, such as work schedules and possibly product mix (at least in the short term). Other types of variability are unknown, such as quality, process times, and breakdowns. Due to variability the resource interactions change over time, thus resulting in the system's dynamics.

A simulation must represent the basic actions that occur in an operations system, e.g. process, store, and transport items. The representation must consider physical aspects – e.g., size, distance, speed – and logical aspects – what, who, when, and where things are done, as well as how much and how long.

There are four basic types of simulation that are used to model and analyze operations systems: discrete-event simulation, Monte Carlo simulation, continuous simulation, and agent-based simulation. While all four of these types of simulation can be done using *FlexSim*, the most common, by far, is **discrete-event simulation** (DES). In DES the states of a system change at discrete points in time as the result of specific events, such as a customer arriving to a system or a work shift ending. A system **state** is a condition of a system or value of a system variable, such as whether a resource is busy or idle or how many customers are waiting for a service.

1.2 Simulation using *FlexSim*

FlexSim is a comprehensive simulation modeling and analysis environment and has extensive capabilities. Some, but not all, of the features and operations are noted below. Key terms are highlighted in bolded font.

1.3 Object Library

A library of modeling **objects** is available to rapidly represent key aspects of operations systems and create system dynamics. The objects are:

- pre-built, yet customizable, representations of actions commonly found in operations systems, for example:
 - planned and unplanned delays, such as process times, repair times, and wait for resource,
 - transportation by means of both fixed objects (e.g., conveyors or robots) and mobile objects (e.g., operators, fork trucks, cranes, and AGVs),
 - resource availability and reliability, and
 - combining and separating items.
 - ...
- dragged from a library, dropped into a 3D modeling view, and connected to create a representation of the operation of a system.
- customized to represent the characteristics of the system being studied by changing an object's **properties** or parameters.
 - The extensive set of object properties consider both **physical** aspects (size, location, capacity, speed, etc.) and **logical** aspects (processing and routing rules, sequence of activities, availability, etc.), all of which either depend upon or influence the current state of a system.
 - In addition to the many standard properties that are available in objects, user-defined properties (called **labels**) can be defined, used, and/or updated anywhere and at any time during a simulation.
 - Each object has a consistent and friendly **user interface**. Object properties are conveniently grouped by tabs on the interface. Many tabs are the same across all types of objects, thus making learning the software much easier. Properties are specified through direct value entry and drop-down menu options.
 - Properties may also be customized and controlled through *FlexSim*'s built-in logic builder, called **ProcessFlow**, or by writing computer code using *FlexSim*'s built-in scripting language, **FlexScript**, a subset of C++.
- inherently 3D; thus, simulation models are created in and run in a three-dimensional environment.
 - Any object's 3D shape can easily be customized. For example, 3D shapes created in *AC3D*, *3ds Max*, *SketchUp*, etc. can be directly imported into *FlexSim*, as can objects from the extensive, online, open-source *3D Warehouse*.
 - Capabilities to “fly through” a model and to create videos of a model in action are built into the software.

1.4 Toolbox

The **Toolbox** is a library of tools that supports modeling building through, but not limited to:

- **Global Tables** store data and easily provide input to a model or obtain output from a model.
- **Lists** dynamically store and process information during a simulation.
- **Time Tables** specify deterministic resource availability, such as shift schedules, break times, periodic inspections, etc.
- **MTBF MTTR** provide means to specify downtime by:
 - randomly making resources unavailable for a duration that is also oftentimes a random variable,
 - basing downtime on model states (e.g. processing and traveling) rather than clock time, and
 - managing multiple or competing downtimes on the same object.
- **Groups** denote objects that are similar or are acted upon in a similar manner.
- **Process Flow** builds complex inter-object and intra-object logic.
- **Dashboards** display model dynamics through charts, e.g., pie, histogram, time series.
- **Excel Import/Export** provides direct links between *FlexSim* and *MSExcel* for inputting and outputting model data.
- Other types of tools for: creating special events and actions, tracking and collecting information on a model variable, creating global variables and macros, developing a video of a model running, etc.

1.5 Analysis

Simulation models are built in order to perform analyses. *FlexSim* provides many tools for analytics, including:

- statistics tracking for:
 - automatically tracking many common measures of performance, such as throughput, content, utilization, etc.
 - defining and tracking custom, system-specific measures.
 - charting to observe system performance on a dashboard as a simulation runs or to export results for reports and presentations.
- experimentation and optimization.
 - easy-to-use, built-in **Experimenter** for creating and comparing changes to sets of model parameters (**scenarios**), running multiple **replications** of a model, providing confidence interval estimates and other types of statistics, etc.
 - direct interface to a leading **optimization** engine, *OptQuest*, in order to effectively search for the “best” set of model parameters to meet specified system objectives.
- export to analysis software – data can be exported to spreadsheets, databases, and graphing and special analysis software for further investigation.
- data import – simulation models in *FlexSim* can be driven from:
 - data stored in spreadsheets and databases.
 - probability distributions selected by using Averil Law’s *ExpertFit* software. (*ExpertFit* is included with a *FlexSim* license.)

All model data are stored in an open, accessible hierarchical data structure referred to as the **Tree**.

Many of these features will at least be introduced in this primer. However, a full discussion of them is well beyond the scope of this introductory document.

The basic operation of *FlexSim* involves the creation and execution of **events** that are based on the logic specified in a model. The events generate actions and activities that occur over time. As a result of the events occurring and/or the current state of one or more objects, items move or flow from object to object. In *FlexSim*, the items that flow through a model are called **flowitems**. The items typically move between resources, either **fixed resources** (e.g., machines, conveyors, and storage areas) or mobile resources (e.g. operators, trucks, and AGVs). Items move into and out of objects via **ports** and contain user-defined characteristics, called **labels**. Mobile resources in *FlexSim* are called **task executers** since they execute a sequence of tasks such as travel, load, unload, etc.

Over the duration of a simulation, information on the conditions (or states) of a system are gathered, summarized, and used for analysis. Typically the summary information, such as average state values (average utilization, average number waiting for service, etc.), are used to compare the performance of alternative systems.

1.6 Notation and formatting conventions

In order to enhance the readability of this primer, certain formatting conventions and notation are used throughout the document.

As introduced above, *FlexSim* includes many objects and tools to facilitate simulation model building and analysis. When referring to these entities, for readability their names are capitalized, e.g., objects such as Processor and Source and tools such as Dashboard and Time Table. This is consistent with the way these entities are identified on the *FlexSim* user interface.

Key words are bolded the first time they are introduced.

A significant part of modeling is customizing general objects, such as a Source or a Processor, to represent the system being considered. In *FlexSim* this involves specifying properties or parameters on the objects through a user interface, which primarily involves selecting from drop-down menus or directly specifying values. Therefore, for readability:

- Names of the properties or parameters, such as **Process Time**, **Send To Port**, and **OnEntry** are denoted in bolded font.
- The values of the properties, e.g. **1000.0**, **First Available**, **item.Type**, are also bolded.
- Selection options within a drop-down menu, such as **Statistical Distribution**, **First Available**, and **Data > Set Label**, are shown in bolded font.
- Names of object tabs, such as General and Flow, are not bolded.
- Instructions to the reader for carrying out specific modeling actions are highlighted by separating them from the text and preceding them by the ➤ symbol.

In addition to providing step-by-step instructions on building example models, this primer provides general tips on modeling and analysis and on effective ways to use *FlexSim*. These are denoted by the following symbols, presented in a smaller font, and indented from the main text.



General modeling tips or best practices are denoted by the M symbol.



General analysis tips or best practices are denoted by the A symbol.



FlexSim tips are denoted by the FlexSim logo.



[section or chapter reference] denotes where a topic is discussed in Beaverstock, Greenwood, & Nordgren, *Applied Simulation Modeling and Analysis Using FlexSim*, 5th Edition, 2017.



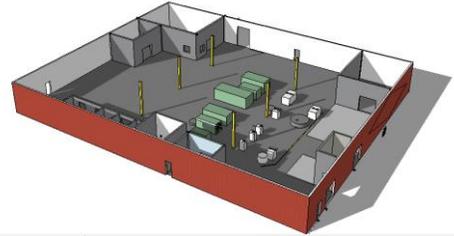
Suggested time to save a model.

2 DESCRIPTION OF THE SYSTEM TO BE MODELED

This primer uses a single model to introduce the various features and capabilities of *FlexSim*. and to provide step-by-step instructions for building a simple, yet comprehensive, simulation model.

2.1 Overview

Dobre Products Limited (DPL) is planning to reuse an area in one of its production facilities to finish and pack containers for distribution. The existing facility is shown to the right, but all of the equipment will be moved out so that the space can be used to support the new production area.



The new production area will finish various types of containers and then the containers will be packed, where the contents depend on the type of container. All process times – in both finishing and packing depend on the container type. Containers are expected to enter and leave the production area on conveyors.

At this stage of the design, DPL is not sure how many different types of containers they will produce, the demand for each type, and the components that will be packed into each container. However, they want to optimize operations and use simulation to help them design the facility and operations. Therefore, the simulation model will initially be built so that it reflects the general operations, yet contains numerous assumptions that will be modified as the project evolves. This approach is essential so that the project can be done in a timely manner - the simulation work must start well in advance of when all of the information is known. In fact, the simulation will be used to help make some of the design decisions and establish the system's characteristics and capabilities.



This is a very wise approach – *simulation is most effective when applied early in the design process* since things can easily be changed early in a project, before too many resources are committed, too many costs are incurred, and too many decisions are made.

The modeling process begins by identifying and representing the basic components of production in the finishing and packing areas (equipment, material, operators, transporters, etc.) and defining the relationships among the components. Later this is expanded, or scaled up, so that the model aligns with the company's production plans.



Modeling should be done sequentially – start small and simple and add complexity as needed. The best model is not the one that is the most complex – *the best model is the one that has the minimum amount of detail needed to answer the questions being addressed*. Remember, no model can replicate the real system – the real system is too complex. Any model is a representation, albeit a simplification, of the system being considered.

2.2 Modeling approach and basic system description

As with any simulation project, the modeling and analysis of the aforementioned system is done in steps, moving from the simplest representation to the more complex. After each step, it important to test and validate the model. The validated model is saved in a file with a different name so that if a subsequent modeling effort encounters a problem, modeling can “rollback” to the last saved and validated version.



One of the most difficult aspects of modeling is deciding how much detail should be incorporated into a model. This is oftentimes referred to as **model fidelity**. It is very tempting to model as much detail as possible, but in this case more is not better. The more complex a model becomes, the harder it is to validate, verify, and maintain. *Models should be as detailed as needed in order to answer the posed questions.*

Of course, this is easier said than done; determining the right fidelity comes with practice and experience. In any case, modeling should start simple and complexity should be added as needed. Start modeling with an extensive list of assumptions and a very simple model. Then, iteratively decide which assumptions need to be removed and remove them by adding features to the model that address the assumptions.

Be careful not to fall into the trap where a model incorporates features and detail because it *can* be done and not because it *should* be done.



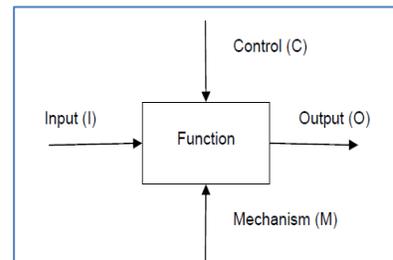
Another good modeling practice is to look for similarities within the system and model them as a single instance; i.e., *create a template*. Test and validate the instance. Once validated, copy and paste the template as many times as needed. For example, if a system has multiple packing stations, model, test, and validate one station, then duplicate as many as needed in the model.



Another good modeling practice is to use small *study models* to develop and test concepts before implementation in the main model; i.e., develop and test in isolation, removed from the complexity of the main model. This is applicable to developing logic within a single object or a set of objects.

A good approach to modeling is to *sketch* the system first. Focus on how the system *operates* and not on how the model will be *built*. Use the sketch to discuss the system operations with domain experts. One diagrammatic methodology for representing dynamic operations systems is the Object Flow Diagram (OFD).

An OFD uses symbols to represent various functions found in a simulation model, such as process, transport, convey, etc. and the edge of each symbol has a specific meaning, as shown in the figure to the right. The Input, Output, Control, and Mechanism notation is derived from the IDEF0 methodology. An OFD for the example model that is used in this primer is provided in the next section.



[Section 17.3] **Prepare a conceptual model.**

The following is a very brief description of the system that will be modeled throughout this primer. It is referred to as the Initial System Narrative and provides a high-level description of the way the system operates. As with many simulation projects, modeling can start with assumed logic and placeholder values until they can be estimated or determined.

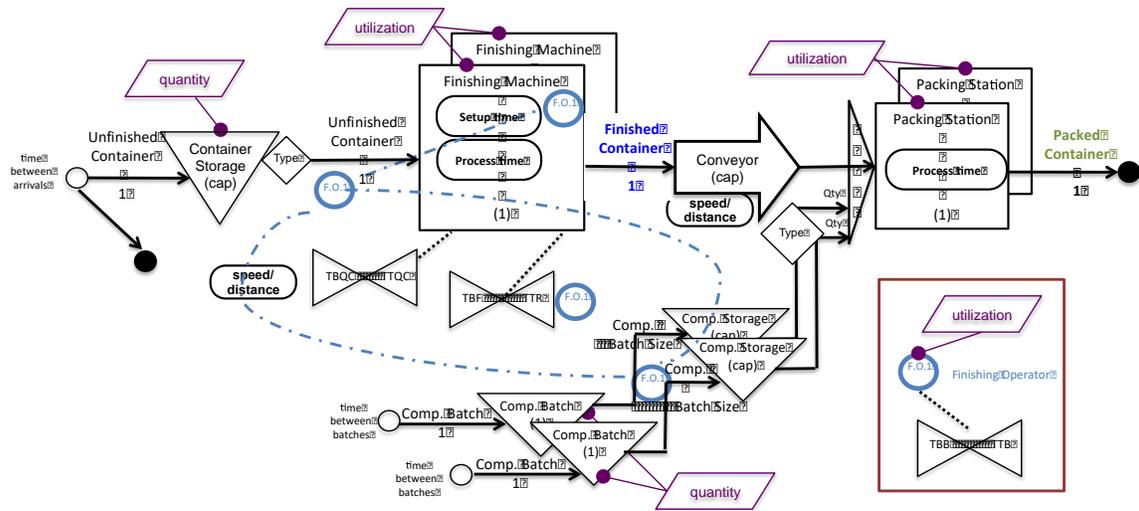
Initial System Narrative

Containers arrive to the finishing area at a yet to be determined rate. The number of types of containers and their properties are also not yet known. The arriving containers are loaded onto finishing machines by a finishing operator. If no machines are available, the containers wait in a buffer before being loaded. The size of the buffer and how the operator decides which container to process next needs to be studied. In order to maximize throughput, DPL is considering using the shortest-processing-time rule to load the finish machines, but is concerned some container types may wait too long for processing if it uses this approach.

The finishing process is automated; i.e., it does not need an operator. The finishing times, which will depend on the type of container, need to be determined. It is expected that a setup operation will be required on the finish machine if the current container type differs from the previous one produced on the machine. It is expected that the finish operator will do the setup. The number of finishing machines that are needed to meet forecasted demand and product mixes is also yet to be determined – the simulation will help determine this. Each finishing machine needs to run a quality check periodically and is subject to breakdowns that require repair. The quality check activity does not require any other resources; it primarily uploads data to a server. However, the repairs that result from a breakdown are expected to be performed by a finish operator.

After finishing, it is planned that containers move to a packing area via conveyor. At packing each container is loaded with a mix of components that depends on the container type. Components are delivered in batches and the finish operator will unpack the batches and make them ready for packing. After packing, packed containers move to the next process, storage waiting picking to fulfill customer orders.

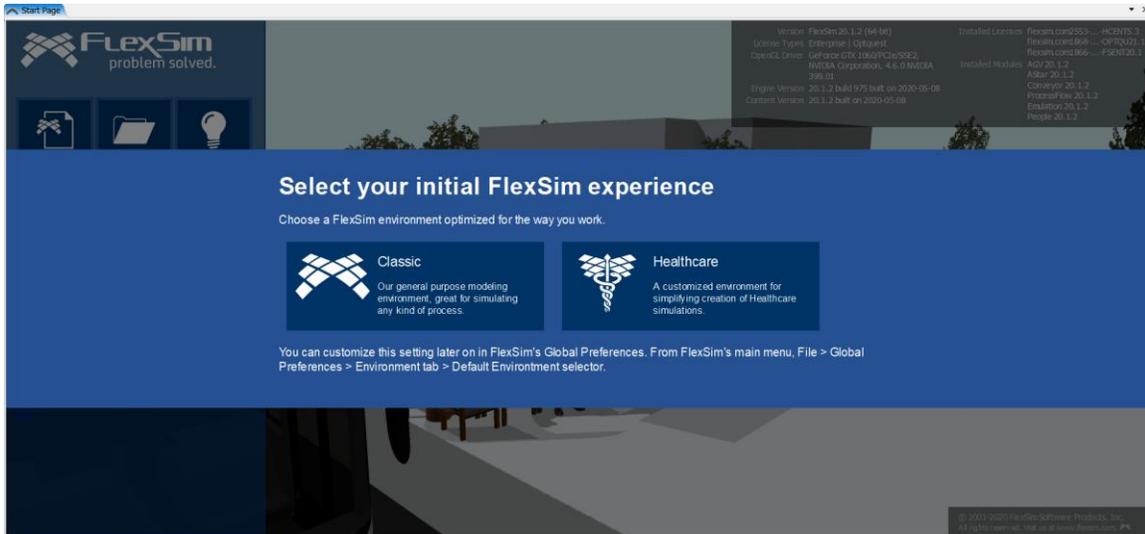
The narrative information is represented symbolically in the following OFD.



Note that the key performance measures of interest – denoted by the “slanted rectangle” symbol are the contents of the storage buffers for the containers and components (denoted as quantity), utilization of the finish machines, finish operator(s), and packing station. Also, the basic item flowing through the model changes, or is transformed by the operations, from a unfinished container to a finished container to a packed container. The finish operator moves material from the buffer storage to finish machines, performs the setup and repairs on the finish machine, and unloads batches of components in the packing area.

3 FLEXSIM'S MODELING ENVIRONMENT

When *FlexSim* is launched, by double clicking the *FlexSim* icon, the first choice is the desired modeling environment, as shown below, either **Classic** or **Healthcare**. This primer only addresses the Classic environment.



Classic FlexSim's Start Page is shown below. The main interface is in the upper left-hand portion of the page, where buttons are provided for starting a new model, opening an existing model, setting preferences, checking licensing information, and accessing the *User Manual*. Below the buttons are a list of recently-accessed model files. A model file can be launched by clicking on its name in the list.



The upper right-hand portion of the interface provides detailed information on the software version.

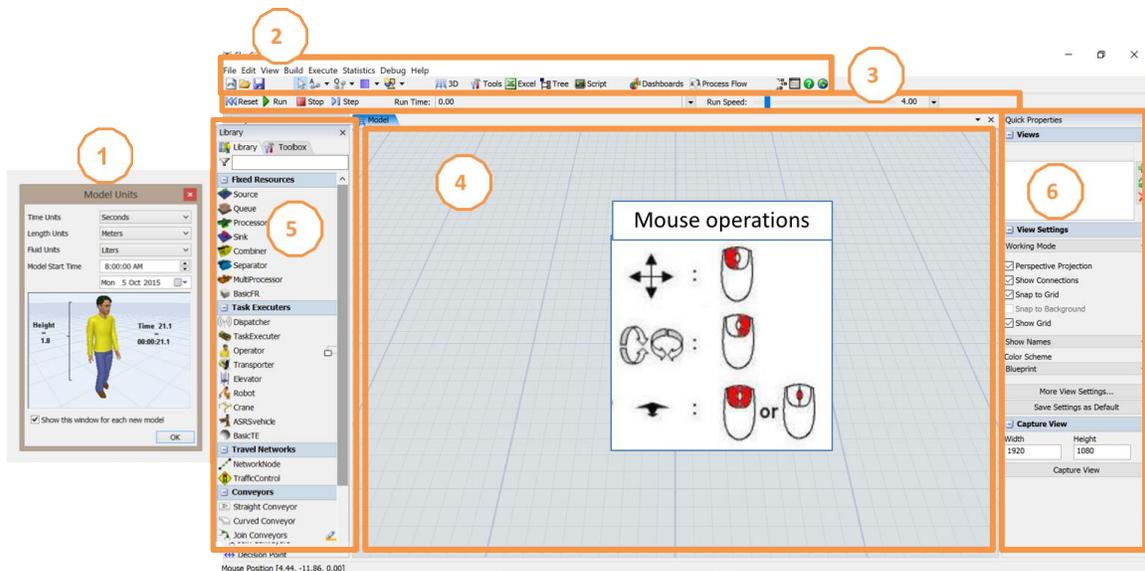
The tiled images in the main portion of the interface are links to FlexSim's website for more information on the software. If you are not connected to the internet, these links will not be displayed.

The 3D model image in the background, in this case the model represents a restaurant, changes from launch to launch, thus providing examples of application areas for where *FlexSim* can be used.

Once a model has been created, it can be accessed directly, without going through the Start Page, by double clicking on the model file.

The file extension for *FlexSim* models is fsm. *FlexSim* automatically creates a backup of the current model through an autosave feature. The default time between saves is 10 minutes, but this can be changed through Global Preferences. The autosaved file is named <your_filename>_autosave.fsm. Note that the model is only saved if it is not running; i.e., if a scheduled save occurs when a model is running, then the save does not occur.

The following figure shows the basic user interfaces in *FlexSim*. The numbers refer to the sections below where each interface is briefly described.



3.1 Time and distance units

When starting a new model, the first information that is needed is the model’s units of measure. *FlexSim* is actually unit-less – i.e., simulations are conducted using general time units and distance units. Therefore, it is up to the modeler to specify the units appropriate for the system being modeled. A number of different model units are available through a drop-down menu for each unit of measure. Note that the default units are metric; i.e. length (distance) is in meters, time is in seconds, and fluid (volume) is in liters.

This first step involves a very important decision, since the specified **units cannot be easily changed in a *FlexSim* model once they are initially set.** However, there is Measure/Convert tool within *FlexSim* to help with conversions.

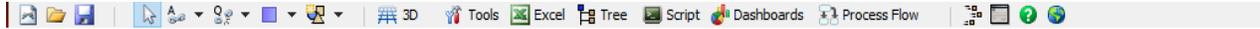
Model Start Time can be changed at any time, but in most cases, its value can be ignored when starting a model.

3.2 Main Menu and Main Toolbar

The Main Menu functions the same way as in most *Windows* applications. File, Edit, View, etc. – provide drop-down menu options for basic functions, such as opening and saving files or setting global preferences (File), undoing modeling actions or setting/viewing model parameters (Edit), viewing various user interfaces (View), executing a model (Execute), running experiments (Experimenter), and accessing the User Manual and license information (Help). The menu option Debug is for advanced users and is not discussed here.

Global Preferences is an important option under the File menu, especially for setting graphics card compatibility and customizing the tool bar.

The Main Toolbar is located below the Main Menu and is composed of a set of icons that provide shortcuts to basic *FlexSim* operations and capabilities, such as saving a model, opening additional 3D views, accessing the Toolbox, linking to *MS Excel*, creating or accessing Process Flow logic, etc.



3.3 Model execution toolbar

As the name indicates, this toolbar, located below the Main Toolbar, controls a model's execution. **Reset** suspends model execution, removes all flowitems from the model, resets all statistics, and sets the simulation clock back to 0. **Run** starts the execution of a model. **Stop** pauses a model run; the run can be continued by pressing the Run button again. **Step** advances a model one event at a time.



Run Time shows the current simulation clock time and its drop-down menu is used to set the planned stop time. There is no default planned stop time so unless suspended through Stop or Reset, a model will run indefinitely. The duration of a simulation run is in simulated time, not real time. Time in *FlexSim* is unit-less; it is given context through the user's specification of time units (e.g., seconds, minutes, days). If the user specifies the model units as seconds, then a Run Time of 10,000 is 10,000 seconds of simulated time (about 2.8 hours). Of course, it will not take 10,000 seconds to run because simulations can run much faster than real time. The execution time depends on the Run Speed.

Run Speed is like *FlexSim*'s "gas pedal" – it controls how fast a simulation runs. It can be set with the slider bar or via its drop-down menu. If a model's units are seconds, then a speed of 1.0 means the simulation is running in real time, i.e., one second of simulation time is one second of real time. Similarly, if the speed is set to 100, the simulation is running 100 times faster than real time. Therefore, if a model's Run Time is 100,000 seconds (about 28 hours) and Run Speed is set to 100, then the 100,000-second (28-hour) simulation will take 1,000 seconds (about 17 minutes) to run. The Maximum option in the Run Speed drop-down menu is the fastest a simulation will run based on the host computer's capability. *A simulation's speed can be adjusted as it runs to move quickly ahead in time in order to get to an interesting point in the model execution or slow down to carefully watch a certain behavior.*

The **Experimenter** provides another means to run simulation models and is discussed later in the primer.

3.4 3D model view window, modeling surface, and mouse operations

The model view window is *FlexSim*'s primary interface since this where models are constructed in 3D. The modeling surface is a gridded infinite plane in the x and y directions and located at 0 in the z direction. The x direction is to the left and right, the y direction is forward and backward, and the z direction is up and down.

The black cross on the grid denotes the origin, where $x=0$, $y=0$, and $z=0$.

The grid units reflect the length units that are defined when starting a new model.

As discussed in the next section, *FlexSim* objects are dragged into the model view and placed on the modeling surface. By default, object are placed *on* the modeling surface, but can be located above or below the surface based on the z-location parameter.

A mouse is used to navigate in the 3D model view. As shown in the figure above, the mouse movements are as follows: (1) the left mouse button moves the view left and right and forward and backward, (2) the right

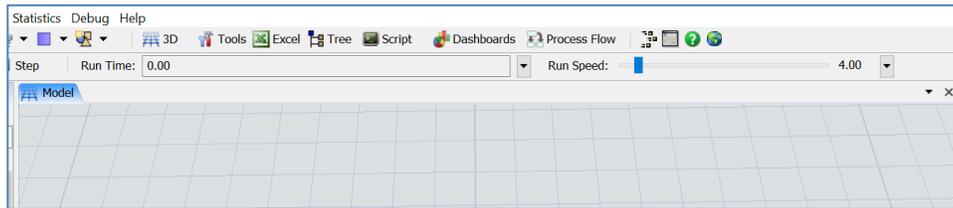
mouse button rotates the view, and (3) using both mouse buttons, or the scroll wheel, zooms the view in and out.



To reset the view of a model so that it is centered at the origin and is in a flat, two-dimensional plane: (1) Right-click anywhere on the modeling surface, then (2) Select View >, then Reset View. This two-dimensional view is useful for placing objects in specific locations on the grid, connecting objects, and when the model appears to be lost somewhere in 3D space.



As shown below, the modeling surface is actually a view of the 3D model and is a tabbed interface in the modeling environment. As will be seen later, other things, such as dashboards and tables, can be a tabbed part of the modeling environment as well. The tabbed views can be removed by clicking on the X button in the top-right portion of the view. In the case of the Model view, it can be reopened by pressing the 3D button in the Main Toolbar. Therefore, *if you close the Model view, you only lose the view of the model, not the model itself!* Also, as discussed later, model views can be saved so that the saved view can be quickly recalled at any time.



A model view can also be created through the drop-down menu options on the View option on the Main Menu. Other views can be created from there as well, such as the Quick Properties, Drag-Drop Library, Toolbox, etc., which are described below.



Multiple model views can be created so that the model can be viewed from various perspectives. Again, views are created via the 3D button on the Main Toolbar. However, having too many views open can slow down model execution since the graphics on all of the views must be updated as a model runs. Another approach for having multiple model views is discussed in a later portion of the primer. Again, closing views do not affect the model. Even if all views are closed, the 3D model is still there, it is just not displayed.



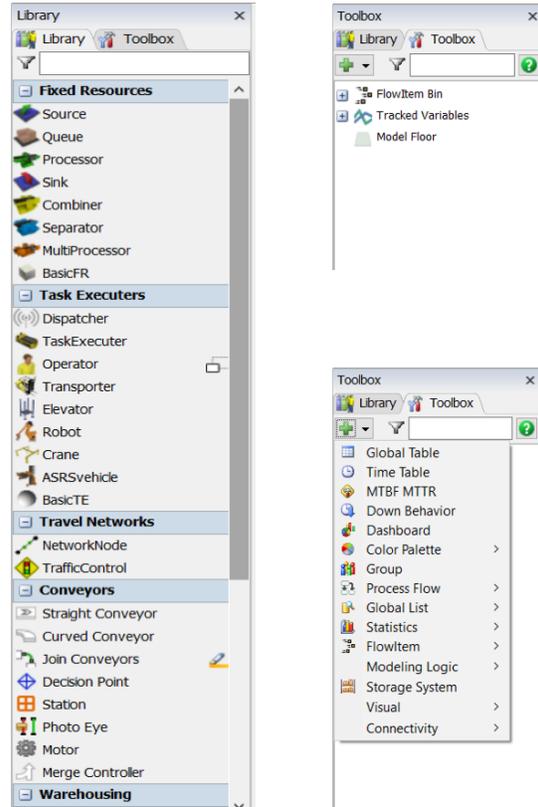
[section 4.2] **Simulation environment**

3.5 Object Library and Toolbox

As mentioned earlier, models are created in 3D by dragging *FlexSim* objects from the Object Library to the model view window and dropping them onto the modeling surface in the 3D view. The Object Library (also referred to as the Drag-Drop Library in the View drop-down on the Main Menu) groups objects by major categories, such as Fixed Resources, Task Executors, Conveyors, etc. A portion of the Object Library is shown in the figure to the right.

Other types of object libraries can be used in a model, such as special-purpose and user-developed libraries of custom objects. However, this is a more advanced feature and is only mentioned here to introduce a *FlexSim* capability.

The tab next to the Object Library is the tool library, referred to as the Toolbox, that provides access to a variety of modeling aides, such as data tables, time tables, dashboards, etc. An example is shown to the far right. The Toolbox shown at the top is the default list of tools that are used in a model, i.e., in a new model. The view of the Toolbox below the default is a list of all of the tolls that are available. This list is accessed through the + icon. Many of these are discussed throughout the primer.



The various modeling objects and tools are introduced as the modeling examples are built. While *FlexSim* contains a wide range of objects and tools to facilitate modeling very diverse and complex systems, this primer only considers a subset of the capabilities, the key aspects that are needed to get started with *FlexSim*.

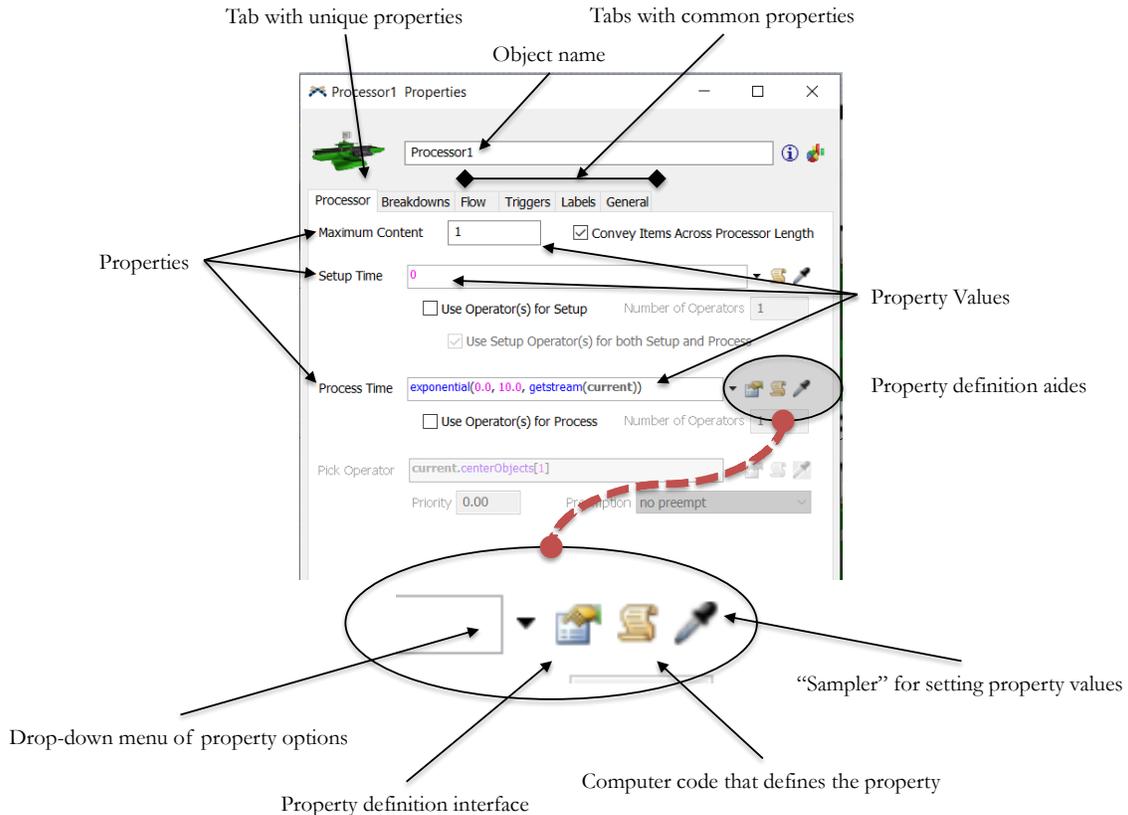
3.6 Object interface

As described earlier a significant part of model building, at least for basic models, is to drag and drop pre-built 3D objects to a modeling surface. The objects are selected, arranged and connected so that they represent the behavior of the system being simulated.

Each object represents a different functionality and contains numerous properties that are available to customize the object’s behavior during a simulation. The properties are grouped by **tabs** on the object’s user interface. Many of the tabs are the same across all objects since all objects can be customized by size and shape, means for routing items to and from the object, defining custom attributes or characteristics (called labels in *FlexSim*), defining special actions that occur within an object, etc. Having these common tabs greatly enhance learning and using the software.

Each object has one or more tabs that contain properties that are unique to its functionality.

The following figure, using the Processor object as an example, illustrates some of the common aspects of 3D object interfaces. For the Processor there is one unique property tab named Processor and the common tabs Flow, Triggers, Labels, and General. The Processor object has numerous **properties**, such as: Maximum Content, Setup Time, Process Time. The properties' corresponding *values* are in the text boxes; for the aforementioned properties, the values are 1, 0, and `exponential(0, 10, getstream(current))`, respectively. The values of the properties are what can be customized by the user/modeler.



To help provide values for the properties, *property definition aides* are found to the right of many property text boxes. One such set of aides is highlighted by the gray oval in the figure above. The aides are called out and identified at the bottom of the figure and briefly defined below.

- Drop-down menu of property options. Property values can be set by many means, such as: specifying a numeric constant, sampling from a probability distribution, looking up a value in a table, and following logic that is based on one or more states of the system. The drop-down menu is used to provide a list of means that are available for setting the property value.
- Property definition user interface. Once a means for setting a property value is selected, it likely requires specifying some parameters in order to be implemented, such as specifying the mean of a probability distribution or identifying which table to use for looking up a value. For each type of means, *FlexSim* provides a user interface that contains the required parameters that need to be specified.
- Computer code of the property. For more advanced applications, the predefined means for setting a property value may not be adequate. One way to further customize the model is to modify the underlying logic. Clicking on the scroll-shaped icon provides access to the underlying computer code, which may be modified using *FlexScript*, a subset of C++.

- **Sampler.** The eye-dropper-looking tool provides a convenient way to access properties of 3D objects by just clicking on the object with the Sampler and then selecting from the resulting set of applicable property values. This tool is illustrated later in the primer. It is especially useful when accessing information from 3D objects while modeling in Process Flow, *FlexSim's* custom logic builder.

3.7 Quick Properties panel

All of an object's properties are accessed by double clicking the object. However, a subset of commonly accessed properties is provided in the Quick Properties window. The window also includes some basic statistics that are updated as a model runs.

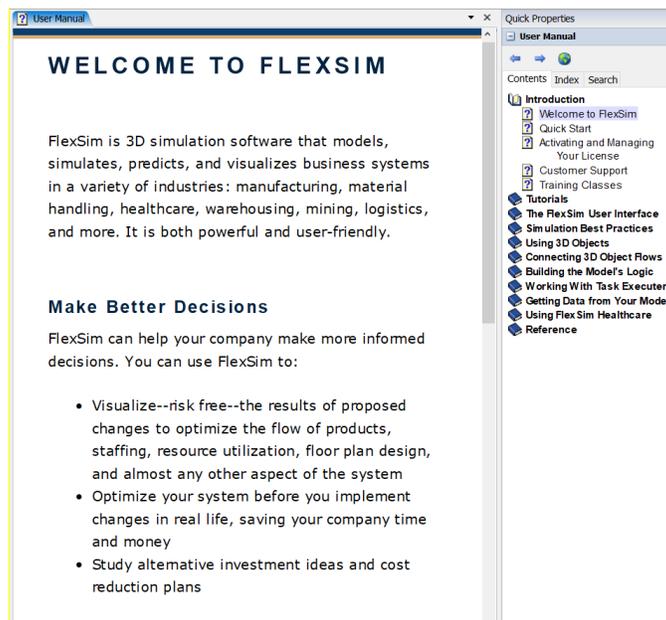
Clicking anywhere on the modeling surface brings up some of the viewing properties on the Quick Properties panel. For example, the Views section is used to create specific views of a model, such as an overall model view or a close-up of an individual object, and to easily switch between views. Also, in the View Settings section, the default Working Mode is used for model building, but Presentation Mode is useful for demonstrating a model since it hides connections, labels, etc.

3.8 Help

FlexSim offers many ways to obtain clarifications and more detail while using the software. Three of the primary means are introduced here.

3.8.1 User Manual

The first means to obtain additional information is via the Main Menu option **Help**. There are many resources available here, but the key one for new users is the **User Manual**. When this menu item is selected, the window shown below opens.



The left pane of the window provides the textual description of a topic.

The right pane, in the Quick Properties panel, contains three tabs. The first tab provides the table of contents for the full user manual. The second tab provides an extensive list, which can be searched, of indexed topics that are covered in the manual. The third tab provides the ability to search the User Manual using key words.

3.8.2 Context Help

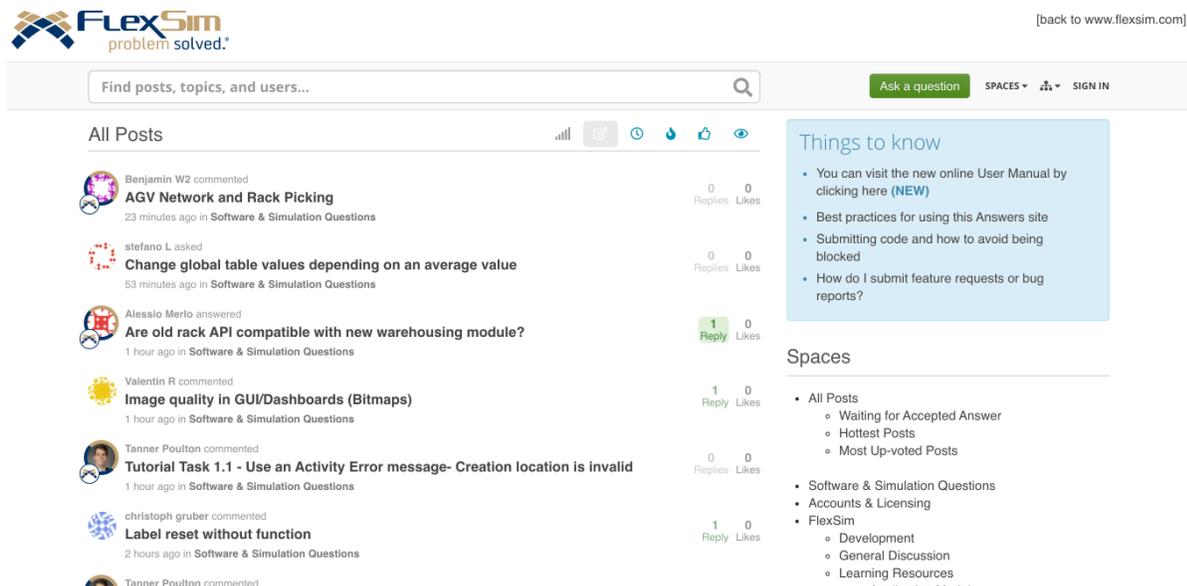
The second means to get help and further information is via the Help icon.



This icon can be found throughout the software. When the icon is clicked it goes to the relevant part of the User Manual depending on the context. One place where the icon is prevalent is in the lower-left corner of each tab of each object user interface. Clicking on it opens the User Manual with a description of that tab's functionality and an explanation of each property and parameter on the interface.

3.8.3 Answers

The third means for obtaining further information is the online help area that is referred to as **Answers**. It is a part of FlexSim's website and can be accessed at <https://answers.flexsim.com/index.html> . As shown in the snapshot of the site below, users can search for posts and topics and pose their own questions. This is a very active site and the posts are extensive, but the search capability makes it extremely useful.



Prior to using the site, one should review the best practice for using the forum.

<https://answers.flexsim.com/articles/22192/best-practices-for-using-this-answers-site.html>

All of these resources are very valuable for both new users and experienced users. Obviously, there are many means beyond this primer to learn about the vast capabilities of *FlexSim*.

4 GETTING STARTED WITH THE SIMPLEST MODEL

To start building a model of the system described in Section 2, start with the most simple model. This model is used to introduce the basic 3D modeling objects in *FlexSim* in terms of their functionality, structure, and properties, as well as obtaining basic output from the simulation. Once these fundamentals are explained, along with a step-by-step guide to customizing the objects and building the model, the primer describes how the basic model is expanded to better represent the system being considered and introduce more of *FlexSim*'s functionality and capability.

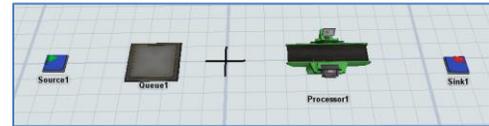


[section 4.3] Simulation components

A new model is started by double clicking the *Flexsim* program icon and selecting New Model from the Start Page.

For this case, use the default units, except set time to minutes; i.e., the model units should be meters, minutes, and liters.

Start with the most simple model by dragging the following objects from the Object Library to the 3D modeling view: Source, Queue, Processor, and Sink. The objects can be located anywhere, but arrange them similar to the figure to the right.



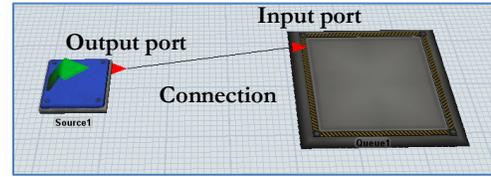
These objects represent the first part of the production process, the finishing operation, and provide the basic functionality for the simulation model. The Source generates flowitems; in this case it generates the containers that are to be finished. The Processor represents the finishing operation and the Queue is used to retain items that have arrived, but cannot access the finishing machine because it is busy. By default, a Processor can only process one item at a time. The Sink represents the next process and that is not modeled at this time. Items should move to the packing area, but that will be modeled later – we are starting small and simple.

While the four objects define the model's functional parts, the model is not yet complete - the relationship among the objects still must be defined. In this simple case, the objects are related by the sequential flow of the items between the objects. The item flow is defined by connecting the objects and, as with most aspects of modeling, there are several ways to do this. In *FlexSim*, there are several types of connections between objects; the item flow connection is referred to as an A-connection.

One approach for connecting objects for item flow, an A-connect, is to follow the sequence:

- Press the A key (notice the cursor changes from an arrow to links in a chain).
- Keeping the A-key pressed, click on the “from” object, where the flowitem comes from (the object becomes highlighted with a yellow box).
- Keeping the A-key pressed, drag toward the “to” object, where the flowitem is going (notice a yellow line emanating from the highlighted object).
- Click on the “to” object.
- Release the A key.

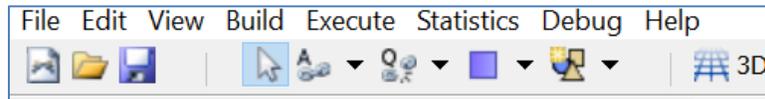
The result of an A-connect is that the two objects are connected with a thin black line as shown in the figure to the right. Also, two small triangles appear on the edge of the two objects. The one on the right-hand side of the Source is its **output port**, from where flowitems leave the object. The one on the left-hand side of the Queue is its **input port**, where flowitems enter the object. Every A-connect results in the automatic creation of two ports – an output port on the “from” object and an input port on the “to” object.



Be careful with the direction of the object connections since they affect the flow direction. The first object selected is the object *from* which items flow and the second object selected is where items flow *to*. Of course, this can be checked by looking at the ports connections as shown in the figure above.

If a connection is to be removed, follow the same steps as defined above for the A-connection, but hold down the Q key instead of the A key. Notice the cursor changes from an arrow to links in a chain with one link broken.

An alternative connection approach is to use the **Connect Objects (A)** icon on the main toolbar, next to the arrow icon and to the right of the new, open, and save icons, as shown below. With this method, the A key is not held down and objects are just clicked in the order that they need to be connected. The ESC key must be clicked once all connections have been made in order to return to Standard Mode, where the cursor appears as an arrow.



Similarly, the **Disconnect Objects (Q)** icon can be used to disconnect objects.

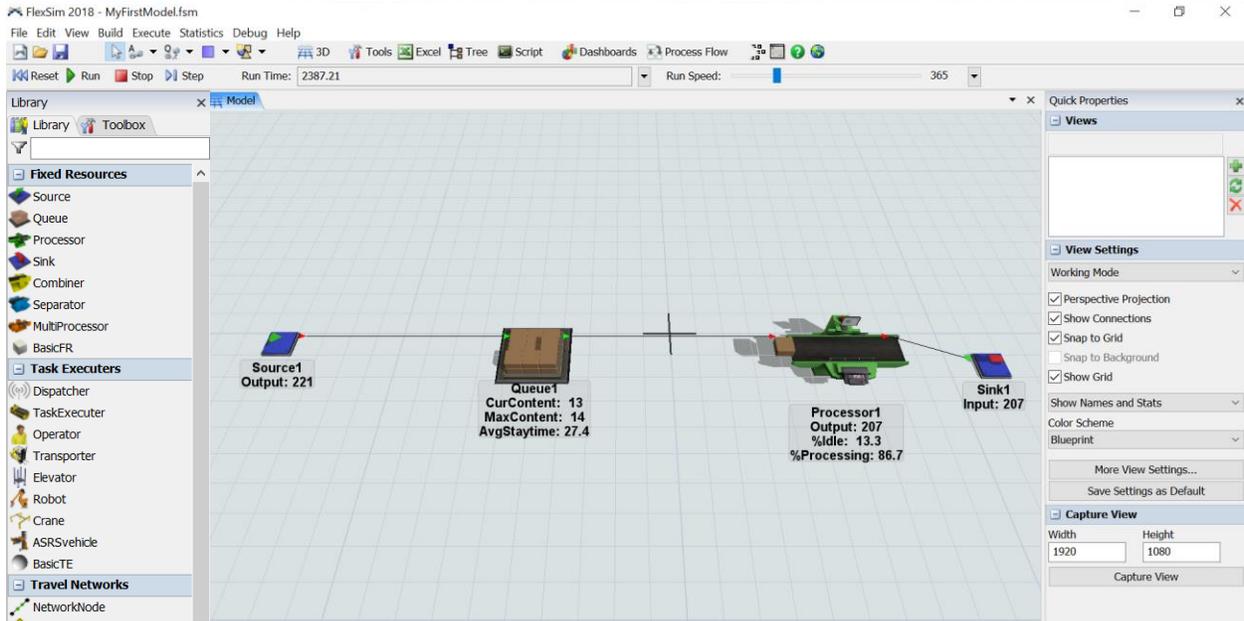


[section 4.7] **Making connections**



If a mistake is made in a model, the action can be undone, by using Undo command located under Edit in the Main menu.

Connect the objects, as shown in the figure below, then **Reset** and **Run** the model. It should look similar to the following figure. Items (boxes) should move across the object named Processor1 and should be accumulating in the object Queue1. If so, congratulations – you’ve built your first *FlexSim* model! Hopefully, this will be first of many helpful models for improving system performance.



Use the **Save** option in the **File** menu to save the current model. Use whatever file naming convention you prefer, e.g. MyFirstModel. *FlexSim* models have the filename extension *.fsm*.

By default *FlexSim* displays each object's name in the modeling view. This is the default **Show Names** setting in the Quick Properties panel (accessed by clicking anywhere on the modeling surface). This can be changed to either **Show Names and Stats** or **Show Nothing**. The **Show Names and Stats** option, as shown in the figure above, displays and updates a few basic statistics about each object as the model runs. As with most things in *FlexSim*, this can be customized.

A few things to note in the figure above, which is a snapshot of the model as it is running.

- The simulation time of the snapshot is 2387.21 minutes (**Run Time** in the Execution Toolbar) after the simulation started. This is nearly 40 hours of simulated time, yet the model only requires about a second or so of real time to run (at a Run Speed of 365, as shown in the figure).
- The Processor has been busy 86.7% of the simulated time, the current value of the **%Processing** statistic.
- Since the simulation began, 221 items have entered the model through the Source (its **Output** statistic) and 207 have left through the Sink (its **Input** statistic).
- The current number of items awaiting processing is 13 (**CurContent**). The maximum number waiting at any time since the simulation began is 14 (**MaxContent**). The average time an item spends waiting is 27.4 minutes (**Staytime**), which may or may not be acceptable system performance; it depends on the system. In a human-based system, it may not be acceptable to wait nearly a half hour for 10 minutes of service. However, in a manufacturing or logistics system, the wait time may be acceptable.

This model is running with all default parameters. While the basic flow logic is correct, the model is not using data that represents characteristics of the example system, such as inter-arrival times and process times. The next section begins the customization process.



Use the **Save Model As** option in the File menu to make a copy of the existing model so that it can be customized beginning in the next section. Again, use any file name, but in the primer, it is referred to as Primer_1-1.



Whenever a model is saved, *FlexSim* creates a second file with the same filename, but with the extension .fsm!. This is a backup file so that if the basic model file is corrupted or lost this file can be enabled by deleting the ! at the end of the extension.



FlexSim also automatically saves the model in a separate file every 10 minutes (the default time which can be changed on the Environment tab of the **Global Preferences** located in the **File** menu. The name of the backup file is the filename plus _autosave, e.g. Primer_1-1_autosave.fsm.



It is good practice to save a model frequently and to save the model as a new version, with a new name, whenever significant changes are to be made. This permits a roll back to a model that has been tested and works as desired if a mistake is made while making the change or if the change does not provide the desired result.

5 BASIC FIXED RESOURCE OBJECTS AND CUSTOMIZATION

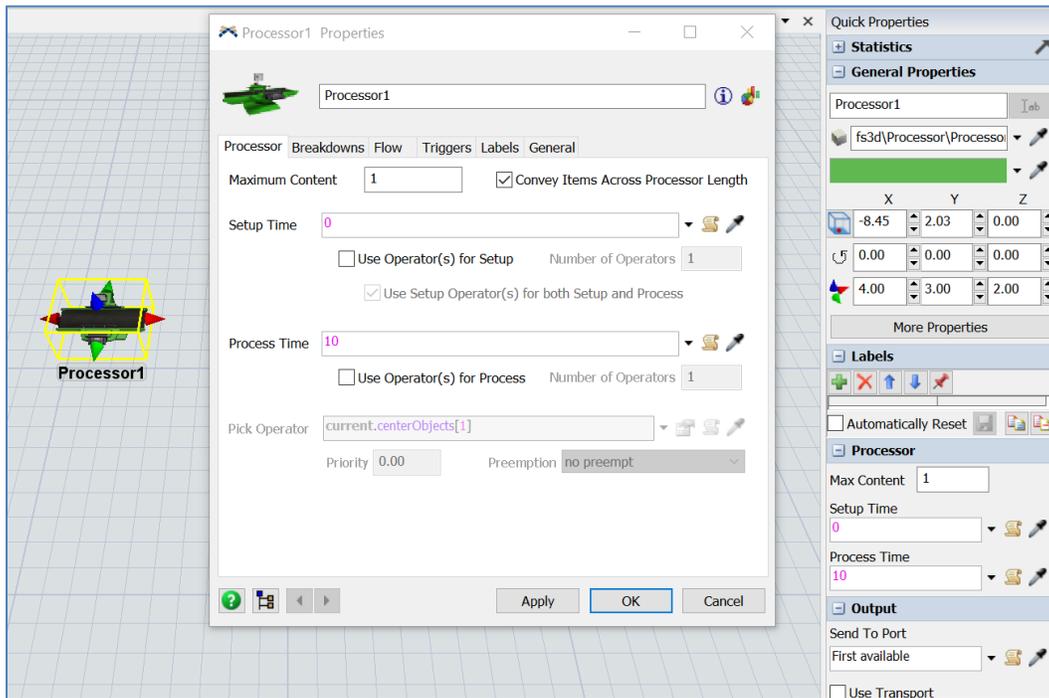
Each of the basic modeling objects that are used in the simple model are explored, and in doing so, the objects are modified so that the simple model becomes a better representation of the real system that is considered in the primer.



[sections 4.4, 4.7] Fixed resources, Making connections

5.1 Basic object properties and structure

The objects are customized by modifying their properties, which are accessed by double clicking on the object in the model view or through the Quick Properties panel when an object is selected in the model view. The figure below shows both interfaces – the Quick Properties are available in the panel to the right and the main or detailed properties interface is shown in the center of the figure.



The Quick Properties interface provides access to a commonly-used subset of an object’s properties. It is accessed by selecting an object, i.e., just by clicking or selecting it and without opening.

However, in this primer, most of the properties will be changed through the main, detailed interface that is accessed by double clicking the object. All of the properties of an object are organized on tabs. For the Processor, as shown above, all of its properties are contained on six tabs.

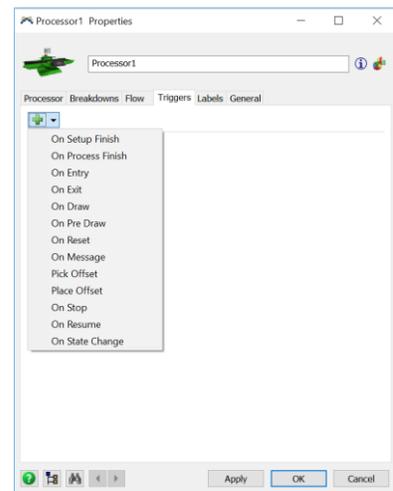
The number of object properties in *FlexSim* is extensive, thus providing a great amount of modeling flexibility and capability. Since the purpose of this primer is to introduce the basics of how *FlexSim* works and some of its capabilities, only the most basic and salient features and properties are presented. Therefore, it is well beyond

the scope of this primer to discuss every property on an object interface and every object and feature in *FlexSim*. With the foundation provided in this primer, the User Manual can be used to understand other properties and parameters.

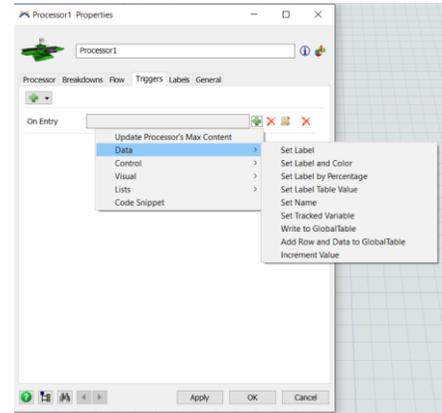
It is important to note the buttons at the bottom of the object interface. **Apply** saves any changes, but does not close the window. **OK** saves any changes and closes the window. **Cancel** does not save any changes and closes the window. Avoid closing the object windows with the **X** in the upper-right corner of the window – it functions the same as **Cancel**, thus not saving any changes made in the properties in the window.

One thing to note as the various objects are explored is that the basic structure of the different types of objects is the same. Object properties are grouped using tabs. All objects have the tabs defined below and one or more tabs that contain properties unique to that type of object. Having a common user interface for object properties greatly enhances learning *FlexSim* and its overall ease of use.

- General tab – basic object information, including:
 - Appearance for specifying 3D shape, color, position, size, etc.
 - Flags for controlling what is displayed and the object’s accessibility.
 - Ports for displaying and controlling what objects are connected to the object’s ports.
- Labels tab for defining custom attributes that are used or updated during a simulation; there are multiple types of labels, numeric, string, pointers, arrays, etc.
- Triggers tab for invoking optional actions that can occur as a model runs.
 - Triggers have many and diverse uses and are grouped both by *when* the action occurs and by *what* type of action occurs. The type of trigger varies by the object, but many of the triggers are common across all of the objects.
 - When an action specified by a trigger is executed, it is said that the trigger “fires.”
 - Triggers are first specified by *when* they occur, such as **On Reset**, **On Stop**, when an item enters or exits the object or both (**On Entry** and **On Exit**), etc. The figure to the right shows when triggers can fire in a Processor. For example, when a setup operation is completed (**On Setup Finish**), when the basic process is completed (**On Process Finish**), when a flowitem enters the Processor (**On Entry**), etc.



- Once *when* a trigger fires is defined, then *what* action is to occur is specified. The figure to the right shows *what* actions are possible when an **On Entry** trigger fires. The actions are grouped by the type of actions: **Data**, **Control** (e.g., open or close object ports), **Visual** (e.g., set color or size), **List** (e.g., push to or pull from lists), and **Code Snippet**. In the case of the figure to the right, for an **On Entry** trigger on a Processor, a variety of **Data** actions are possible, such as **Set Label**, **Set Label Table**, **Set Name**, etc.



- All objects have **OnReset**, **OnMessage**, and **Custom Draw** triggers.
- Note, all Fixed Resource objects have both **OnEntry** and **OnExit** triggers, except the Source only has an **OnExit** trigger (nothing enters a Source); similarly, the Sink only has an **OnEntry** trigger (nothing exits a Sink).
- All Task Executors have **OnLoad** and **OnUnload** triggers; these are analogous to the **OnEntry** and **OnExit** triggers for Fixed Resources.
- All objects have triggers related to their type, e.g., **OnCreation** in a Source object, **OnProcessFinish** in a Processor, or **OnResourceAvailable** in an Operator object.

All Fixed Resources have a Flow tab that has three major functions:

- Control *to which output port a flowitem should be sent* when it leaves an object, e.g. based on the downstream object(s) availabilities or queue size, an item’s attribute or label value, or a table value.
- Specify *whether an item uses a transporter* to reach the next object. If a transporter is used, it is requested based on specified criteria. The item waits in the current object until a transporter arrives and loads the item. Therefore, if an object’s capacity is one item, then a new item is not moved into the object until the current object exits via a transporter (if required).

Note that if no transport is used, then an item moves between two Fixed Resource objects in zero simulation time regardless of the distance between the two objects.

- Enable *pulling items into the object*. Normally items move through a model by being “pushed” from one object to another; i.e., the current object determines to which subsequent object an item moves. Pulling enables an object to determine which item it processes next based on specified criteria.

Each object has a name. Since *FlexSim* must ensure each object has a unique name, it automatically assigns a name when an object is created by selecting it from the library and dropping it into the 3D modeling view. When a model has multiple instances of a type of object, even with a relatively small number of objects, it becomes difficult to recall which object does what in a model, e.g. the difference between Source1 and Source2. Therefore, each object should have a meaningful name that reflects its role in the model; e.g. ContainersArrive is a much more meaningful name than Source1. The naming is for the modeler’s benefit, and others who access the model, not for the software.



Begin with the basic model from the previous section, named Primer_1-1, and **Save As** Primer_1-2. Basically make a copy of the model so that it can be customized and the original model is retained.

5.2 Source object

The Source is a modeling boundary, a starting point for what is being considered in the system. For this example, what happens to the containers prior to arriving to the finish area is not a concern at this time.

Objects should be named in a meaningful way. Therefore, change the object’s name from Source1.

- Name the Source object **ContainersArrive**.

Information about the flowitem being created is specified on the Source tab. Flowitems are created in one of three ways, based on: (1) inter-arrival times, (2) schedule, or (3) sequence. The default value for the **Arrival Style** property is **Inter-Arrival Time** and the default is used in this example. The **Arrival Schedule** option for the **Arrival Style** property is discussed later in the primer.

The time between arrivals is defined by the **Inter-Arrivaltime** property. The default is:

exponential(0, 10, getstream(current))

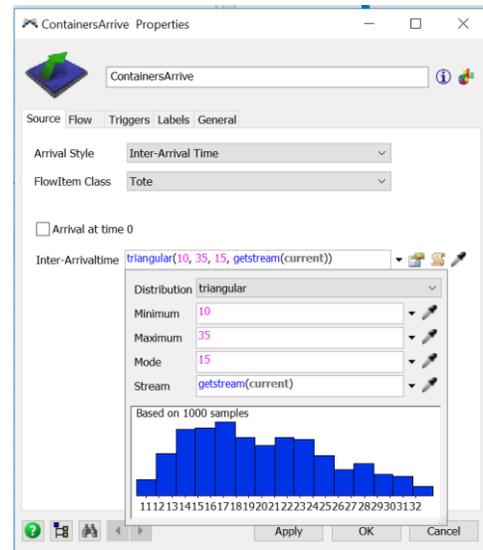
i.e., each inter-arrival time is obtained by a random sample from an exponential probability distribution with a mean of 10 time units, 10 minutes in this example.

The property’s value can be set from a variety of drop-down menu options. For this example, and as shown in the figure to the right, change the inter-arrival time distribution.

- For the **Inter-Arrival Time** property, select the **Statistical Distribution** option and then the **Triangular** option in the submenu.

Note that there are many distributions to choose from in *FlexSim*. Each distribution has a different set of parameters. The triangular requires three parameters - the minimum, maximum, and most-likely (mode) times between arrivals.

- As shown in the figure to the right, enter the triangular distribution’s parameters for this example. The minimum, maximum, and most-likely (mode) values are 10, 35, 15, respectively.



In order to implement a distribution, the user interface provides the specified values to a *FlexSim* command; so that it can be executed; in this case, it translates the interface values to:

triangular(10, 35, 15, getstream(current))

These specified parameters result in an average time between arrivals of 20 minutes, the sum of the three parameters divided by three. In general, the mean of a triangular distribution is the simple average of its three parameters.

An average time between arrivals of 20 minutes corresponds to an average arrival rate of 3 customers/containers per hour. Rates are the reciprocal of time values; in this case, the average arrival rate is

60 minutes/hour \div 20 minutes/customer = 3 customers/hour. Simulation software typically use average times rather than average rates.

The last parameter in specifying any probability distribution in *FlexSim* is the number of the random number stream to be used for this source of variability. This is implemented through the command, `getstream(current)`. Discussion of random number streams is beyond the scope of the primer. Also, commands are discussed later in the primer. However, it is fine to use the default for the **Stream** property. It is good general practice that stream values be unique for each source of variability in the model – this is handled automatically by *FlexSim* through the `getstream(current)` command.



Discussion of selecting probability distributions for interarrival times, process times, failure times, etc. is beyond the scope of this primer. *ExpertFit*, which comes with a *FlexSim* license, is a good tool for fitting data to distributions.



Also, it is generally a good idea to use bounded distributions so that the model does not use unrealistic, extreme values that can result from such distributions as the normal, exponential, etc.



[Chapter 8] Modeling Randomness

The type of flowitem that is created by the Source is specified by the **FlowItem Class** property. The default type is **Box**, but there are other options in the drop-down menu. Also, it is easy to create custom flowitems. This is done later in the primer.

➤ For this example, select **Tote** as the **Flowitem Class** value.

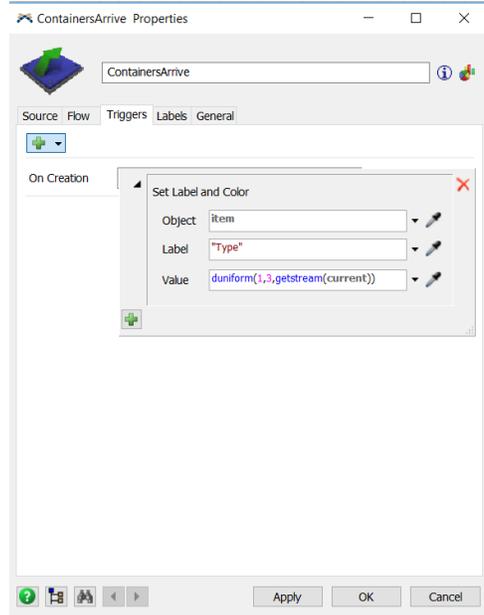
In this example, there are multiple types of containers, all of the same class; i.e., they all use the same flowitem container, **Tote**. For now, assume three equally-likely types; this will be changed later in the primer. To easily distinguish the types in the model they will each have a different color. To model this in *FlexSim*,

- In the Source object, use the **OnCreation** trigger and select **Data** and then select the **Set Label and Color** option. Use the default properties as shown in the figure to the right.

As mentioned earlier, attributes in *FlexSim* are called *labels*. The **OnCreation** trigger that is shown to the right creates a label named Type for each item that is generated at the Source. In this case, the value of Type is the result of a random sample from a discrete-uniform distribution with possible values of 1, 2, or 3, with each value equally likely. This action is accomplished by the *FlexSim* command

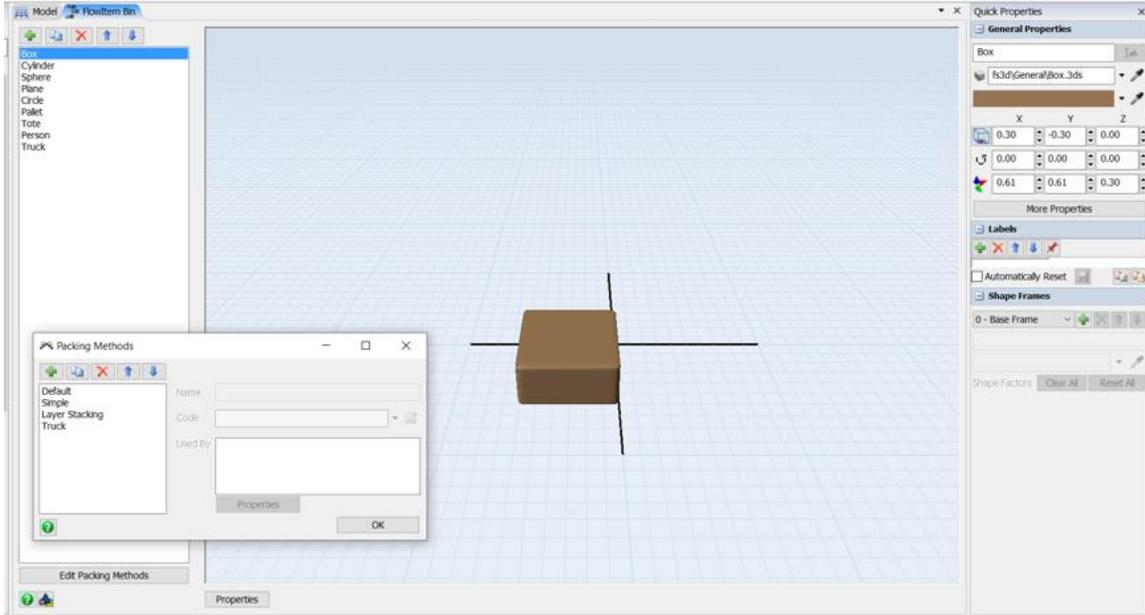
duniform(1, 3, getstream(current))

This trigger also sets the item's color based on its Type value. The default color scheme in *FlexSim* is 1=red, 2=green, 3=blue, 4=yellow, etc. Of course, a custom color scheme can be defined.



5.3 Flowitem Bin

Flowitems are customized through the **FlowItem Bin**, which is accessed either via the ToolBox or its icon located on the right end of the Main Toolbar. An example of the FlowItem Bin is shown in the figure below. The default classes of flowitems are shown in the left-hand portion of the interface. The lower-left corner shows the packing methods that are available for the container flowitems., as defined below. The right-hand panel in the figure below shows the Quick Properties of the selected flowitem.



There are two general categories of flowitems – Basic and Container.

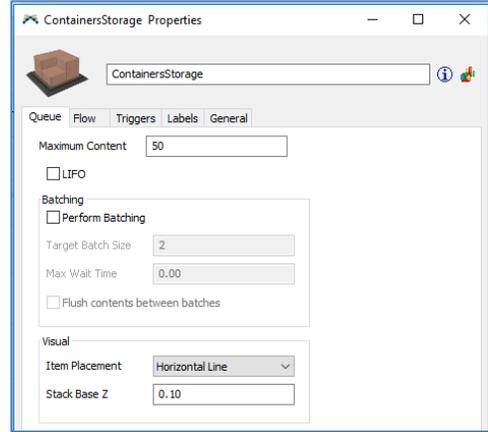
- Basic flowitems are simple shapes, such as **Box**, **Cylinder**, **Sphere**, **Plane**, **Circle**, and **Person**. Their visual appearances – such as size, shape, rotation, color - can be customized.
- Container flowitems – **Pallet**, **Tote**, and **Truck** – are special in that they can contain basic flowitems and can be packed in various ways.
 - The **Default** packing method is used by the **Pallet**. Items are stacked on top of the container filling up as much available space as possible before moving to the next level. This is similar to the way items are stacked in a Queue by default.
 - The **Simple** packing method is used by the **Tote**. This is similar to the **Default** method except flowitems are packed *into* and not *onto* the container.
 - The **Layer Stacking** method considers the uniformity of flowitem sizes. When flowitems have a consistent height (z direction), they are put on one layer of the container. As soon as the height of a flow item differs from the others, it is placed on a new layer. The width (x direction) or depth (y direction) can be checked instead of z.
 - The **Truck** packing method is used by the **Truck** class of flowitem. Within the Truck item, other flowitems are first stacked on top of each other in the forward end of the truck and the stacking moves toward the back.
 - Each of the packing methods can be customized through *FlexScript*.

5.4 Queue object

The Queue object delays an item if no downstream object is available to process an item. In this case, if a container arrives to the system and the Processor is busy, it waits in the Queue.

For this example, the Queue properties are changed as follows and as shown in the figure to the right.

- Name the Queue object **ContainerStorage**
- Change the **Maximum Content** property from the default value of 1000 to 50. By default, the queue size is assumed large (1000), but in this example, it is assumed that the maximum realistic quantity is 50. This will likely change as the system is designed and the space available for storage becomes clearer.
- Change the **Item Placement** property from the default value of **Stack inside Queue** to **Horizontal Line**. By default, items are stacked in the queue by filling up as much available space as possible before moving to the next level. *The quantity in a layer depends on the physical size of the Queue and of the flowitems.*



5.5 Processor object

The function of a Processor is to invoke a planned delay on items flowing through a model.

- Name the object **FinishMach_1**; it is anticipated that more than one finish machine will be needed.

Each finish machine processes one container at a time, which is the default value for the **Maximum Content** property. However, note that a Processor can process multiple flowitems concurrently by changing this property's value.

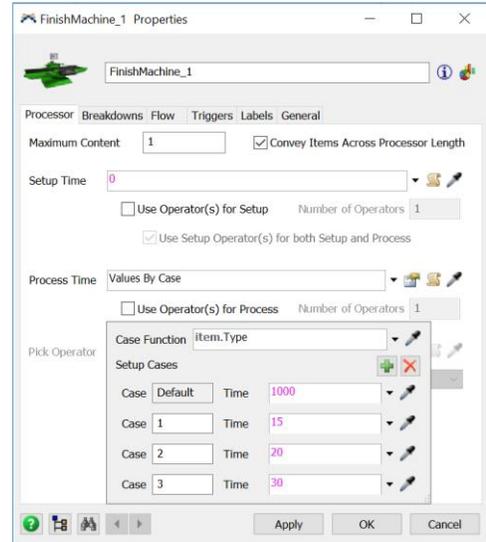


Using the **Maximum Content** property is a quick and convenient way to represent multiple resources by using only a single object. This is handy when building a simulation model to obtain a rough estimate of system requirements. However, modeling multiple resources through a single object precludes modeling certain system aspects, such as downtime on individual resources, operator travel to specific resources, processing specific types of items on specific resources, etc.

The planned delay is specified by the **Process Time** property. The default is a constant 10 time units. In this example, the finish time depends on the type of container; i.e. the finish time for Type 1 is 15.0, Type 2 is 20.0, and Type 3 is 30.0. This is implemented through the **Values By Case** option. In this example, each case corresponds to a type; therefore, three cases are created.

To model the situation where process time depends on product type, perform the following steps:

- Select the **Process Time** property, then select the **Values By Case** option from the drop-down menu.
- Create 3 cases, in addition to the default, by using the + button; then set the values as shown in the figure to the right.



The logic works as follows. When an item enters this Processor and needs a process time, *FlexSim* checks the value of the label Type and it becomes the case number. Therefore, if an item's Type value is 2, the time specified for Case 2, 20.0, is used for the process time. The Default Case value is used if an item has a Type value other than 1, 2, or 3.

The notation in the **Case Function** property is the way *FlexSim* refers to a label. In general, it is referred to as **dot notation** because a dot or “.” separates parts of the notation, i.e. object.LabelName. In the case of item.Type, the object is the flowitem, referred to as item, and the name of the label that is being considered is Type.



The default process time in the **Values By Case** logic is set to 1000, or any large number. This helps the modeler to remember to add a Case if a new type of container is added to the model. This way, if a fourth type is added in this example and this property is not updated with a new Case, then the item will be delayed a long time at the Processor and things will back up in the model. This should prompt an investigation as to the problem and hopefully the failure to update the process time is discovered and remedied. If a small value is used for the default, this oversight may not be caught and the new container type will use an erroneously low value, thus affecting the results.

Since the three types of containers are equally likely, the average processing time is 21.67 minutes (average of 15.0, 20.0, and 30.0). Recall, the average time between arrivals from the source is 20 minutes. This means the single finish machine will be overloaded since the planned utilization is greater than 100%, on the average ($21.67/20.0 = 1.083$ or 108.3%). The situation will actually be worse since there is variability in the system due to the type of container, time between arrivals, and the finish machine not being available 100% of the time due to downtime. One way to remedy the large queue contents is to use another finish machine, which is done in Part 2 of the primer.

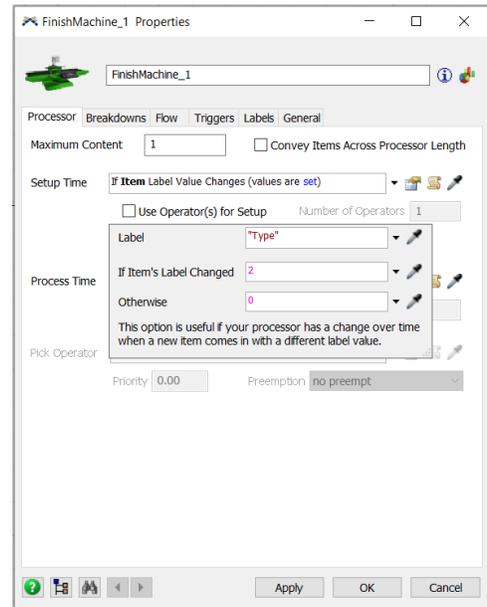
In addition to the process time, there is a *setup* operation. The setup operation is separated from the process time for two reasons:

- (1) An operator will perform the setup operation, but the machine will run on its own once it is set up. (This will be incorporated into the model in the next part of the primer.)
- (2) The setup operation only needs to be performed when the type of item (container) changes.

The time to perform the setup operation, when it needs to be performed, is assumed to be a fixed two minutes.

The setup logic is implemented into the *FlexSim* model as follows and as shown in the figure to the right:

- Select the **Setup Time** property on the Processor object's Process tab, then select the drop-down menu option **If Item Label Value Changes**.
- Update the resulting menu's property values as shown in the figure to the right and as described below.
 - **Label:** no change, use the default **Type**.
 - **If Item's Label Changed:** Set this property's value to 2.
 - For the property **Otherwise**, change the default value 5 to 0.



5.6 Sink object

The Sink object removes flowitems from a model; therefore, it is the opposite of a Source. However, this functionality is much less complex than that of a Source. There is not as much involved in removing items from a model, compared to creating them, which involves defining what to create, with what properties, when to create them, etc. The Sink is another modeling boundary; the stopping point for what is being considered in the system.

- The only change that is made to the Sink is its name, **NextProcess**.



If you haven't already done so, save the model. Recall, it is good practice to save often.

6 BASIC MODEL OUTPUT

Simulation models are developed to assess the dynamic behavior of an operations system. Therefore, there are multiple means to access the results or output from a simulation model, a few simple ones are described here. Output is usually obtained from *FlexSim*'s Experimenter. It, among other things, is used to:

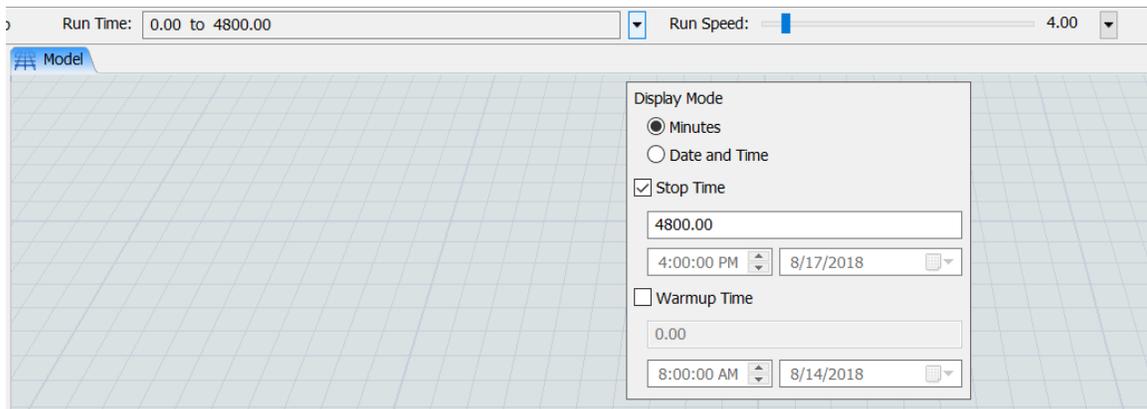
- Execute multiple experiments in a single run.
- Control experimental conditions, such as number of replications and warm-up period (advanced topics discussed later in the primer).
- Obtain statistical estimates of system performance measures.
- Directly link to OptTek's *OptQuest* optimization software.

However, for now, only a few very basic output measures are discussed here.



[section 4-10] **Single-run statistics**

Consider an 80-hour run of the simulation model (equivalent to ten 8-hour shifts). In model terms this is 4,800 minutes; recall, the model's time units were defined to be minutes when the model was initially created. The figure below shows setting the model's run time and is followed by the instructions.

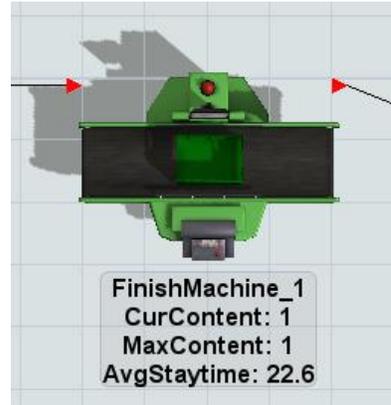


- Select the dropdown menu on the **Run Time** property that is located on the model's Execution Toolbar.
- Check the **Stop Time** box and enter 4800 is entered into the box below it.

So that model finishes more quickly, the **Run Speed** can be adjusted higher than the default value of 4 by using the slider interface.

6.1 Object statistics and Quick Properties

A few basic output measures are provided for each object on the modeling surface, as shown in the figure to the right. These values are continually updated as a model runs. At the moment when this screenshot was captured (at the end of a 4800-minute model run) the Processor, named FinishMachine_1, was processing an item (CurContent), the most it processed at one time was 1 (MaxContent), and the average setup and process time is 22.6 minutes (AvgStayTime).

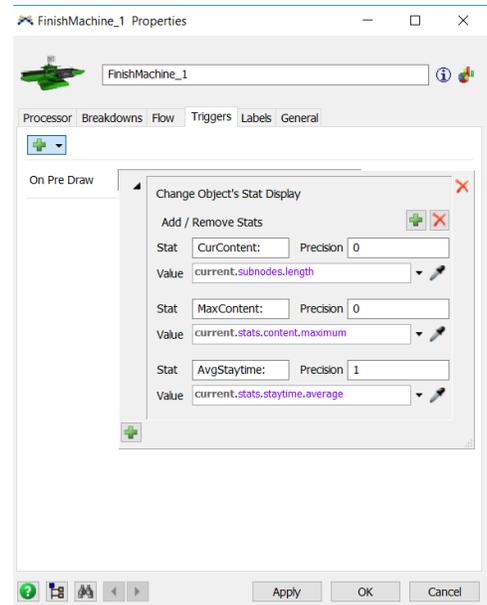


The default object statistics can be displayed for all objects through the **Show Names and Stats** setting in the Quick Properties panel. This is explained below.

As with just about everything in *FlexSim*, these measures can be changed. As shown in the figure to the right, this can be done through a trigger. In order to access the parameters:

- Select the **On Pre Draw** trigger on the object's Triggers tab
- Then, select the **Change Object's Stat Display** option on the object's **On Pre-Draw** trigger.

Discussion of how to add or modify the statistics is currently beyond the scope of the primer.

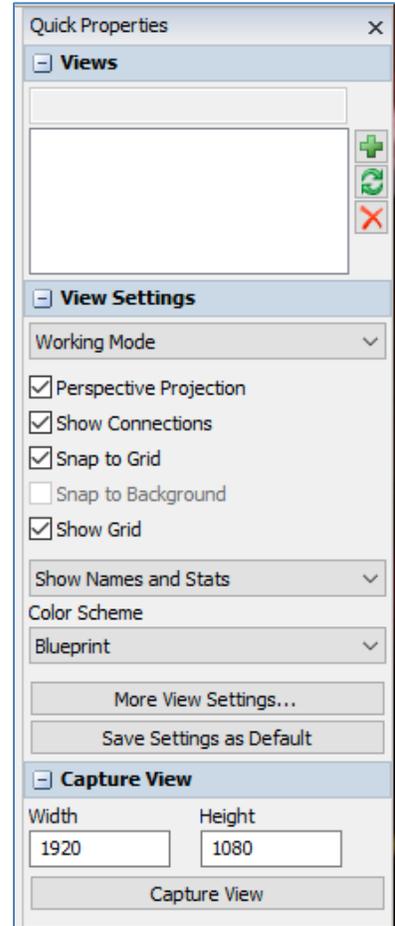


What is displayed on the modeling surface as a simulation runs is controlled by the **View Settings** section in the Quick Properties panel. An example of the panel is shown to the right.

This panel is accessed by clicking anywhere on the modeling surface in the 3D modeling view. Three display options are available: **Show Names and Stats**, **Show Names**, and **Show Nothing**.

The panel is also used to:

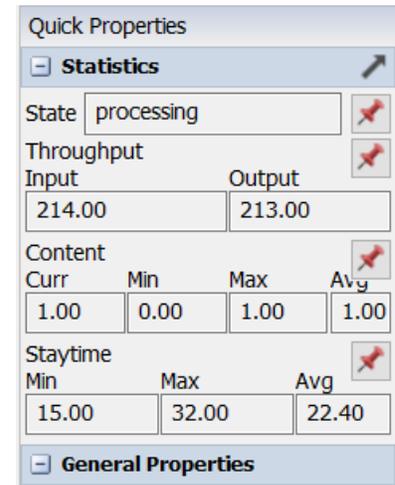
- Create views in order to easily switch the model’s display to predefined views, such as an overall view of the model, a close view of a particular object, etc.
- Change the basic viewing mode from the default, **Working Mode**, to **Presentation Mode**, which removes the grid and connection lines and uses perspective projection.
- Control the view settings individually by checking or unchecking the check boxes.
- Access additional, more advanced, view settings through **More View Settings ...**
- Take a snapshot of the model, **Capture View**, and save it as a graphics file with a png extension.



If an object is selected, a few basic output measures are displayed on the **Statistics** section of the Quick Properties panel, as shown in the figure to the right. These values are continually updated as a model runs. The panel provides information on the object’s:

- **Current State**; in this case, when the simulation ended, the object was busy processing.
- **Throughput**, how many items entered and left the object. In this case, 214 entered, 213 left; thus, one is currently being processed.
- **Content**, how many items are currently in the object and the minimum and maximum number to date that have been in the object, as well as the average number.
- **Staytime**, how long items have stayed in an object, in terms of the minimum, maximum, and average times. Since the process times are constant, the **Min** value, 15.0, is the time to process container Type 1; and, the **Max** value, 32.0, is the time to process container Type 3 plus a 2-minute setup time. The average value for all items is 22.40.

The pushpin icon is used to “pin” that statistic for that object onto a Dashboard, which is described in the next section.



6.2 Dashboard

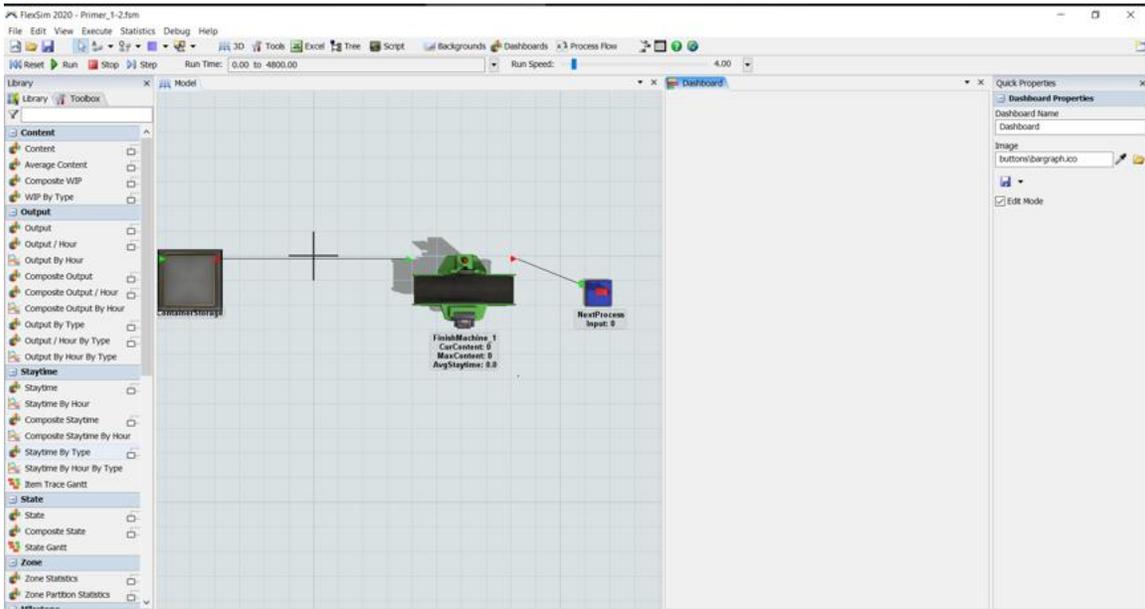
Compared to considering a single value at a snapshot in time, like the on-object and Quick Properties values described in the previous section, charts and graphs are very helpful for assessing the dynamics of a system. *FlexSim* provides a wide variety of charts and graphs through the Dashboard tool. The Dashboard is either accessed from the Main Toolbar or from the Toolbox (Tool Library). A model may have multiple Dashboards that capture various aspects of a system’s behavior.

Three example charts from the Dashboard are described here, two types of time-series plot (a value plotted over time), and a histogram. Prior to describing how to create a particular chart, some background is provided on the Dashboard itself.

Charts and graphs are created on a Dashboard.

- Click the **Dashboards** icon on the Main Menu and select **Add a Dashboard**. If you create multiple dashboards, then they can be selected either from this Main Menu option or can be selected from the Toolbox.

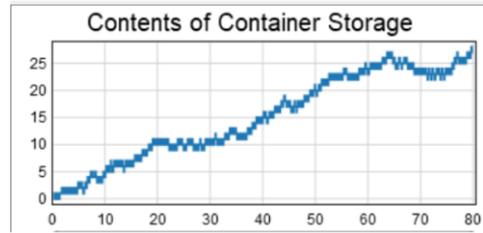
As shown in the figure below, a “blank” dashboard is automatically created in the right portion of the screen and a library of chart tools and templates is created on the left side. The blank space to the right is the workspace to create a set of charts and graphs.



The library of chart options on the left in the figure above offers many templates for commonly-used charts. Of course, as is always the case in *FlexSim*, custom charts can be created. However, for now, the primer uses the predefined ones. Notice that if you click on the grid in the 3D space, then the library changes back to the 3D objects. The library changes based on what part of the *FlexSim* environment you are in – 3D, Dashboard, Process Flow, etc.

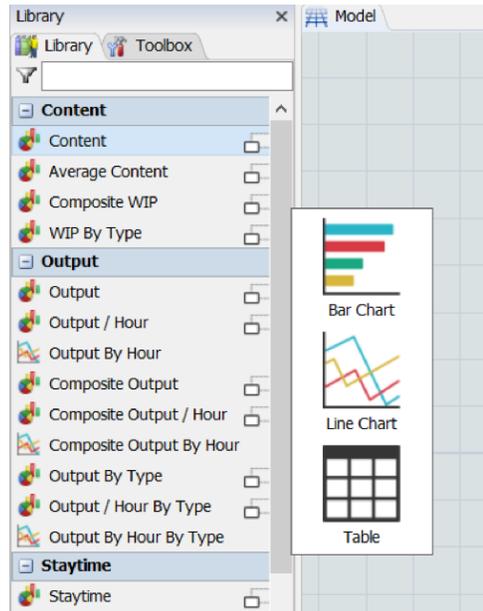
Time-series plot

The first type of chart that is considered in this example, as shown to the right, is a system-level plot of the total content of the ContainerStorage queue over the duration of the simulation (4,800 minutes). The chart clearly depicts the dynamics of the system. It also indicates that, as expected, the Queue is continually growing over the duration of the simulation. Recall, this is because the average process time is longer than the average time between arrivals. Conversely, the average service rate is less than the average arrival rate.



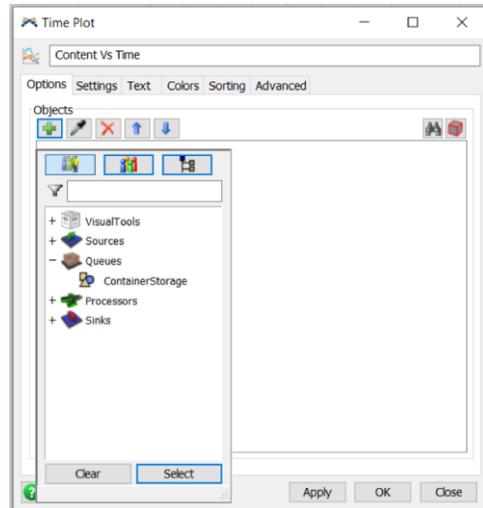
To create the time-series chart:

- As shown in the figure to the right, select the **Content** chart type in the **Content** section of the Dashboard Library; then, select **Line Chart** from the pop-up menu.
- Drag the Dashboard Library item onto the Dashboard panel where it can be sized and positioned.
- As shown in the figure to the right, use the Time Plot interface to rename the chart from Content Vs Time to something more meaningful for the example, **Contents of Container Storage**.



The only required input in the selection is to identify which object(s) to plot. Plots for multiple objects can be overlaid on the same chart. In this simple example, only one object is considered.

- As shown in the figure to the right, on the Options tab, in the **Objects** section, click the + button and choose **Select Objects**. This generates a list of all of the objects that are currently in the model by category (Sources, Queues, Processors, etc.).
- From the **Queues** category, select the object ContainerStorage. Currently the only queue in the model. Notice that the object is highlighted. Now press the **Select** button. The ContainerStorage object is now displayed in the **Objects** section.



- The only other parameter that should be changed for now is the **Time Axis Mode** property on the Settings tab. Change the value from the default **Show exact time and date** to **Show Duration**. Also, change the displayed units from **Seconds** to **Hours**.
- Since only one value is being plotted and the chart title is descriptive, uncheck the **Show Legend** box (also on the Settings tab).
- The default settings for all of the other chart parameters are fine for now. Therefore, press OK to close the interface.
- Relocate the chart on the Dashboard.
 - Notice when a chart is selected, it contains a double-lined frame around it with “handles.” Handles are small squares at the corners and midpoints of the sides of the charts. Also, notice that when the cursor is over the perimeter frame, it changes from the standard arrowed pointer to two crossed double arrows.
 - Left click with the mouse anywhere on the perimeter frame of the chart to move it around and place it anywhere on the Dashboard window.
 - Similarly, left click one of the handles and drag it to resize the chart. Notice the cursor changes to arrows depicting what direction is being resize – horizontal, vertical, or both (diagonal).



The “Content” chart that is introduced above indicates the importance of considering how long to run a simulation. Had the model only been run 480 minutes (8 hours), the maximum contents would have only been about five items, compared to nearly 30 when run 4800 minutes (80 hours). Different decisions could have resulted based on the different run lengths. Discussion of this topic (how long to run a simulation), while very important, is beyond the scope of this primer.



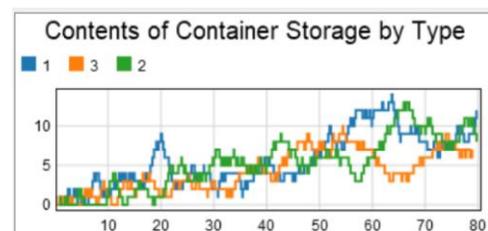
The “Content” chart above also illustrates the dynamics of the simulation. Due to variability and interactions the system could behave quite differently in another 4,800-minute time period, just as in the real system, behavior varies day to day or week to week. This illustrates the importance of running a simulation model multiple times, referred to as **replications**, and combining the results, e.g., through averaging. Discussion of deciding how many replication to run, while very important, is beyond the scope of this primer.



[Chapter 10] **Fundamentals of Output Analysis**

Time-series plot by type

The second charting example is also a system-level time-series chart that shows the contents of the ContainerStorage queue by product type, as opposed to the total of all products. The chart is shown in the figure to the right. It again clearly depicts the dynamic and stochastic nature of the system.



This chart is created in a manner similar to the Content chart above.

- Select the **WIP By Type** chart type in the **Content** templates section of the Dashboard Library; then, select **Line Chart** from the pop-up menu.
- Drag the chart Library item onto the Dashboard panel where it can be sized and positioned.
- Rename the chart to something meaningful, for the example, **Contents of Container Storage by Type**.

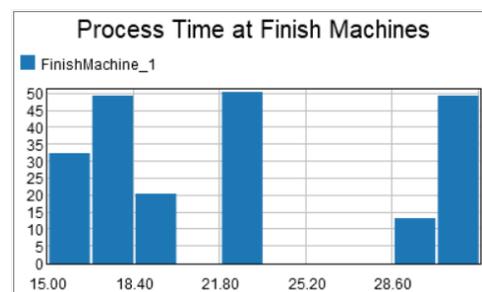
- On the Options tab, click the + button in the **Entrance Objects** section and select **Select Objects** from the drop-down menu. Again, this generates a list of all of the objects that are currently in the model by category.
- From the **Queues** category, select the object ContainerStorage so that it is highlighted and press the **Select** button. The ContainerStorage object is now displayed in the **Entrance Objects** section.
- Similarly, again on the Options tab, click the + button in the **Exit Objects** section and select **Select Objects** from the drop-down menu. From the **Queues** category, select the object ContainerStorage so that it is highlighted and press the **Select** button. The ContainerStorage object is now displayed in the **Exit Objects** section.
- As with the Content chart, the only other parameter that should be changed for now is the **Time Axis Mode** property on the Settings tab. Change the value from the default **Show exact time and date** to **Show Duration**. Also, change the displayed units from **Seconds** to **Hours**.
- Since only there are multiple values being plotted on the one chart, leave the **Show Legend** box checked (again, on the Settings tab).
- Relocate the chart on the Dashboard.
- The default settings for all of the other chart parameters are fine for now. Therefore, press OK to close the interface.

Note that the chart could track the contents of multiple objects over time; e.g., to track the number of items in the queue and the processor, then the Exit Objects would be changed to the Processor, FinishMachine_1.

Also note that the type can be any label value; this example used the Type label, which is quite common. Type is defined at the bottom of the Options tab.

Histogram plot

The third charting example is also a system-level chart and, as shown in the figure to the right, it considers the time an item spends in the Processor (at the finish machine). The stay time includes the three constant process times (15, 20, and 30 minutes) and the 2-minute setup times when they are needed in the operation, i.e. when the item type changes.



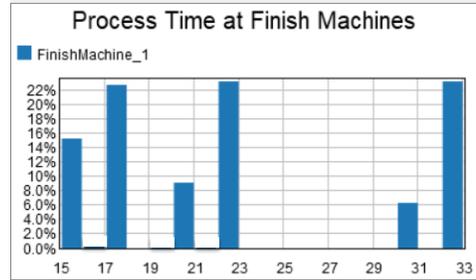
This chart is created in a manner similar to the Content charts described above.

- Select the **Staytime** chart type in the **Staytime** templates section of the Dashboard Library; then select **Histogram** from the pop-up menu.
- Drag the chart Library item onto the Dashboard panel where it can be sized and positioned.
- Rename the chart to something meaningful, for the example, **Process Times at Finish Machines**.
- The only required input is the selection of which object(s) is the subject of the plot. As before, click the + in the **Objects** section of the Options tab., then select the **Select Objects** option.
- Select FinishMachine_1 from the **Processor** section and then press the **Select** button at the bottom of the interface.

The resulting chart, when run for 4800 minutes, should look similar to the one in the figure above. While useful, the histogram can be reformatted to be more readable. Of course, the underlying data are not changed, just the way the data are summarized in the chart.

The following two additional steps make the chart correspond to the one to the right. The x and y axes are customized for readability and interpretation.

- Change the scale of the y-axis to display the frequency of the histogram as percentages (relative values) rather than the default, absolute number of occurrences. To do this, check the box **Normalize Values** on the Settings tab.
- Change the definition of the bars, the x axis, from a fixed number of bars (default is 10) to a fixed bucket width. To do this, again on the Settings tab, change the **Bar Mode** from **By Number of Buckets** to **By Bucket Width**. For this example, set the Bucket Width from the default of 50.0 to 1 (1 minute) and the Bucket Offset, the starting point of chart, from the default of 0.0 to 15 (the shortest processing time is 15 minutes with no setup).



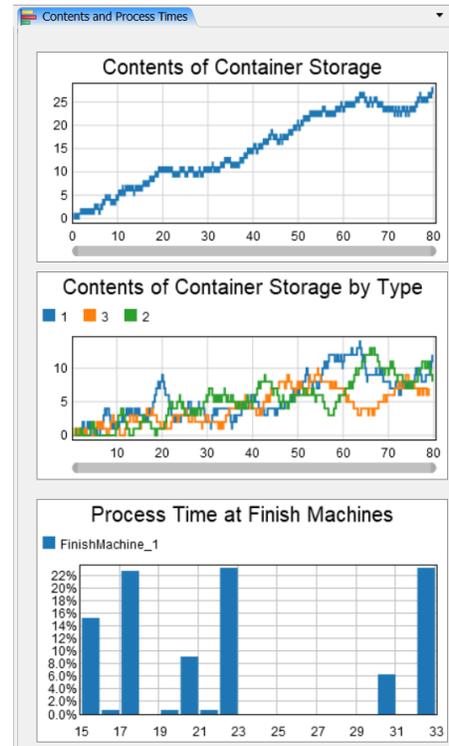
Setting the number of bars and bar width for histograms are oftentimes done by trial and error. However, there are some methodologies to help with this; three such methodologies are available in *FlexSim* – Sturges’, Scott’s, and Freedman-Diaconis Rules. They are available when selecting **By Bucket Width** property. Discussion of these methodologies is beyond the scope of the primer.

- On the Text tab, change the **Precision** property from the default of **2** (2 decimal places) to **0**.
- Press **OK** to close the chart window.

The final dashboard should resemble the figure to the right.

The dashboard has been renamed **Contents and Process Times** since other dashboards will be created later and the descriptive name helps for reference.

- Change the name in the Dashboard Name field in the Quick Properties window, as shown in the figure to the right.





Charts provide a very valuable means for verifying and validating simulation models.

Note the following in the charts on the Dashboard above.

- As shown in the first two plots, the contents of the container storage is growing over time. This is an obvious indicator that the system is unsustainable and not feasible, since in practice queues cannot continue to grow in size. It was indicated earlier that one solution would be to add another finish machine, which will be done later in the primer.
- The third plot, the histogram of staytimes, verifies the three deterministic process times by container type (15, 20, and 30 minutes), with approximate frequencies of occurrence of 15%, 10%, and 7%, respectively. The values of 17, 22, and 32 are the process times by type including setup times; their frequencies are approximately 22%, 23%, and 23%, respectively. Obviously, the setup operation is required more often than not.

Also note that the total percentage of Type 1 is 37% (15%+22%), Type 2 is 33% (10% + 23%), and Type 3 is 30%. (7% + 23%). This is close to the assumption that the products are equally likely to occur. Of course, they will not be exactly equal to 33.33% for any run due to sampling error. However, over the long run, the product mix should be equally likely.



If you haven't already done so, save the model. Recall, it is good practice to save often.

This page is intentionally blank.

PART 2 – NEXT STEPS IN 3D MODELING

The second part of the primer builds on the model started in the previous section. The model is used to introduce additional capabilities in *FlexSim* and illustrate how they are used to represent operations systems. In addition to introducing new modeling objects such as task executors and conveyors, this part introduces some of the modeling *tools* that are available in *FlexSim*.

The *FlexSim* capabilities included in this part are:

- importing layouts and modifying object graphics,
- routing flowitems between objects by various criteria,
- incorporating dynamic objects, referred to as task executors into a simulation model (e.g. operators and transporters).
- transferring items between objects via conveyor objects,
- managing model data through the use of global tables, and
- incorporating planned and unplanned downtime on objects through time tables and reliability, respectively.

1 CHANGING 3D GRAPHICS

Although excellent simulation models can be created with the default graphic for the objects in *FlexSim*, there are two main reasons to become acquainted with the basics of importing graphics.

1. Using images that correspond to the domain that is being modeled helps users better relate to the model. For example, a Processor object would typically be used to represent the operation of a MRI machine in a healthcare project. While the functional aspects of the Processor represents the MRI, the default shape is a machine found in manufacturing. Doctors, nurses, technicians, and administrators would understandably have a problem relating to the manufacturing image representing an MRI machine.
2. Building models on a layout makes setting sizes, locations, and distances much easier. This is especially important when considering the transport of items between objects whether on conveyors or by task executers (operators, fork trucks, etc.) since distance affects system performance.

This primer only considers the very basics of working with graphics. For more information, please see *FlexSim's* User Manual.



Begin with the basic model from the previous section, named Primer_1-2, and **Save As** Primer_2-1. Basically make a copy of the model so that it can be customized and the original model is retained.

1.1 Importing a layout

There are several ways to import a layout into a *FlexSim* model, including importing an *AutoCAD* file. For the primer, the layout is just a two-dimensional graphics file, named Layout.prg. It was created in *Microsoft Office*, but could have been created in any drawing program. It is imported into *FlexSim* as a texture on the **Plane** object, which is found in the **Visual** section of the object Library.

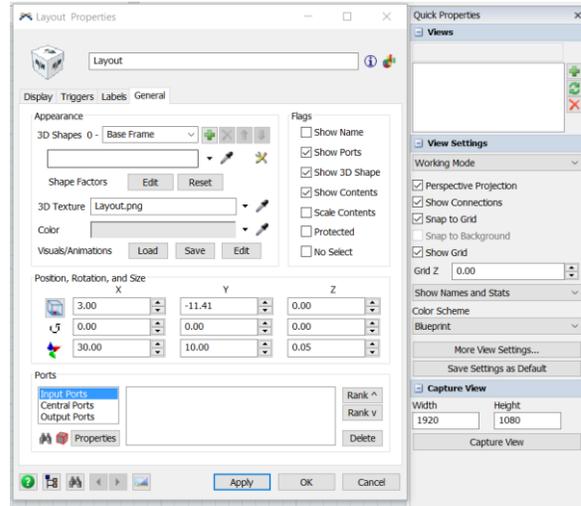
The object interface is shown below. To import the layout:

- Drag out the Plane object to the modeling surface in the 3D view.
- Change the name of the Visual object from Plane to Layout.
- By default, the plane is a solid-colored object. To show the layout, select the **Filename** drop-down menu in the **Texture** section of the Display tab and select **Browse**, at the top of the menu. The Browse option opens Windows Explorer. Use it to locate the layout file on your computer. Note that once the graphic is imported into *FlexSim*, it will appear as part of the **Browse** menu – therefore, if you use the graphic again in the model, you would not need to import it from your computer.

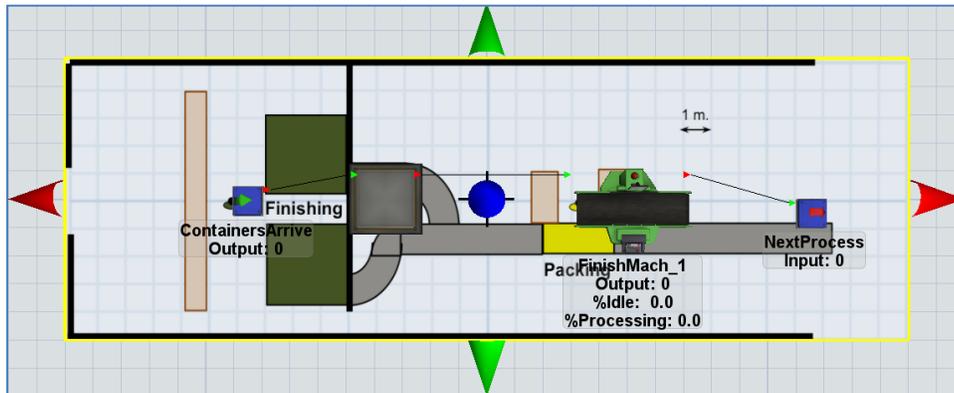


It is good practice to place all supporting model resource files, e.g. graphics and spreadsheet files, for folders containing the files in the same computer directory or folder as the *FlexSim* model. This makes transferring the model to other computers much easier.

- Size the layout object to $x=30$, $y=10$, $z=0.05$ since the workspace is 30 meters by 10 meters. As shown in the figure to the right, this can be done either through the General tab of the visual object or via the Quick Properties window.



- Drag the layout so that it is centered at the origin as shown below. The location values should be $x = 0.0$, $y = 0.0$, and $z = 0.0$.

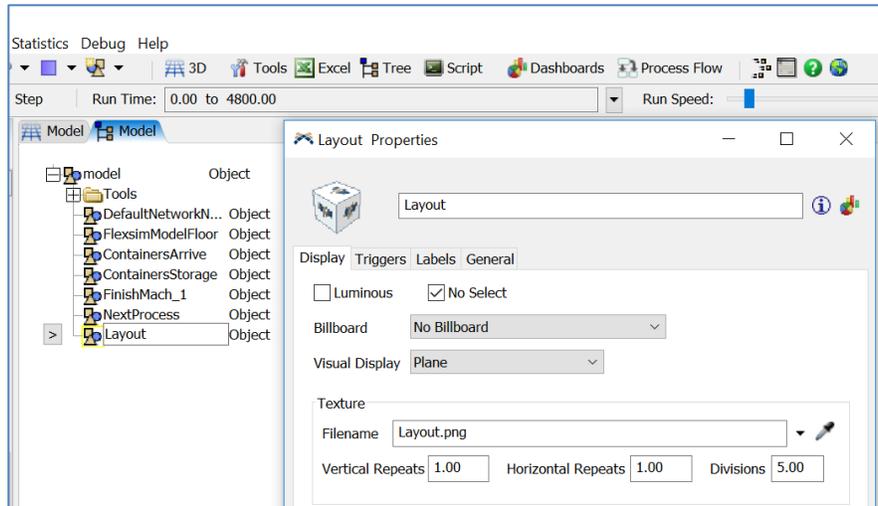


As other objects are placed onto the layout, the layout object might accidentally get selected and moved or other objects may become imbedded in the plane (a useful feature for building hierarchical models, but not in general). In order to avoid these issues,

- The property **No Select** should be checked. It is found both on the Display and General tabs of the Visual object named Layout.

A drawback of enabling the **No Select** property is that, as the name indicates, the object cannot be selected in the 3D view. It must be opened from the **Tree** – the underlying hierarchical data structure in *FlexSim*. Working with the Tree is an advanced topic, but is briefly introduced here.

The Tree is where all of the model data is stored in *FlexSim*. Each object interface is just a link to data in the Tree. The Tree for this model is shown below. It is accessed from the Main Menu icon labeled Tree. To open an object, just double click on the object in the Tree view. This is the same interface that is obtained by double clicking on the object in the 3D view.



Placing the objects in their proper location on the layout in the 3D view is explained in the next section.



When placing objects, it is usually better to work:

- in 2D rather than 3D (right click anywhere on the modeling surface, select **View**, and then **Reset View**).
- **not** in the default perspective view (right click anywhere on the modeling surface and uncheck the **Perspective Projection** box in the **View Settings** section of Quick Properties window).

1.2 Changing object graphics

Before placing the existing objects on the layout, a few of their properties are changed, i.e., their size and 3D shape.

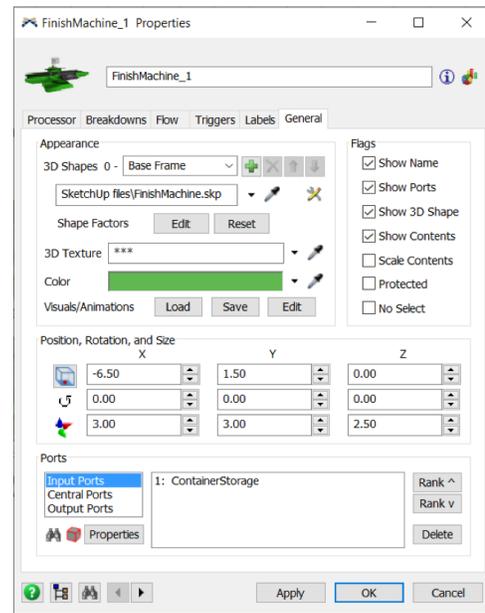
While a 3D shape can be created to represent any system object, many such shapes are readily available. For example, 3D Warehouse (<https://3dwarehouse.sketchup.com>) contains millions of 3D objects that have been created in the 3D modeling software *SketchUp*. Here are a few comments about *SketchUp* and using shapes developed in *SketchUp* in *FlexSim* simulation models.

- Shapes from 3D Warehouse can be downloaded for free and used in *FlexSim*.
- *SketchUp* 3D modeling software is free and can be used to modify any shape downloaded from 3D Warehouse or to build a custom 3D shape object from scratch.
- *SketchUp* is now a web-based application, but older downloadable versions for Mac OS and Windows are available at the SketchUp Help Center (<https://help.sketchup.com/en/article/60107>). *SketchUp Make 2017* is suggested. It is the free version; *SketchUp Pro* has more features, but requires a fee.
- Since file formats change often, it is best not to use the latest format since it may not be compatible with *FlexSim*. Therefore, downloads should be *SketchUp* file version 15 or earlier.
- The size of the *SketchUp* model, especially the number of polygons, affects the *FlexSim* model size and its run time. Therefore, very complex graphics should be avoided unless they are really needed and the model is run on a powerful computer.

Finish Machine (Processor)

The shape of the finish machine is changed from the default graphic to the *SketchUp* file named *FinishMachine.skp*. As shown in the figure to the right:

- Associate the *SketchUp* file with the *FlexSim* object *FinishMachine_1* by selecting **Browse** in the drop-down menu under **3d Shapes** in the **Appearance** section of the object's General tab. The path **SketchUp files\FinishMachine.skp** indicates that the graphics file is in a graphics file named "SketchUp files," which is the same location as the *FlexSim* model. By default, graphics media are imbedded within a model so that the model may be copied without the supporting external graphics files.
- Change the object size properties to $x = 3$, $y = 3$, $z = 2.5$, all values are in meters; i.e. each machine is three meters square and 2.5 meters high.
- Change the location properties to $x = -6.5$, $y = 1.5$, $z = 0$. The object could have been placed in the correct location by moving it manually on the layout



Another finishing machine will likely be needed since the average input rate is equal to the average processing rate; therefore, it is a good time to add it now.

- Copy *FinishMach_1* using Cntl-C shortcut key combination, then click on the layout near where the machine will be located, and paste using the Cntl-V keyboard shortcut.

- Rename the copied object as FinishMach_2 and place it in the proper location on the layout ($x = -6.50$, $y = 2.40$, and $z = 0.0$).
- Make the flow connections: an A-connection from the Queue named ContainerStorage to the finish machine and another A-connection from the finish machine to the Sink named NextProcess.



When copying and pasting objects, it is best to copy (Cntl-C), then click on the modeling surface approximately where the new object is to be located, and then paste the copied object (Cntl-V). If you don't click on the modeling surface, the pasted object may become a subset of the copied object. This is a good feature in some cases, especially where you want to build submodels, but that is a more advanced topic.

Container Storage (Queue)

Next, the Queue or ContainerStorage object is customized. Its shape is changed to a Plane, which is a basic 3D shape that comes with *FlexSim*. It is located on top of the storage area in the imported layout.

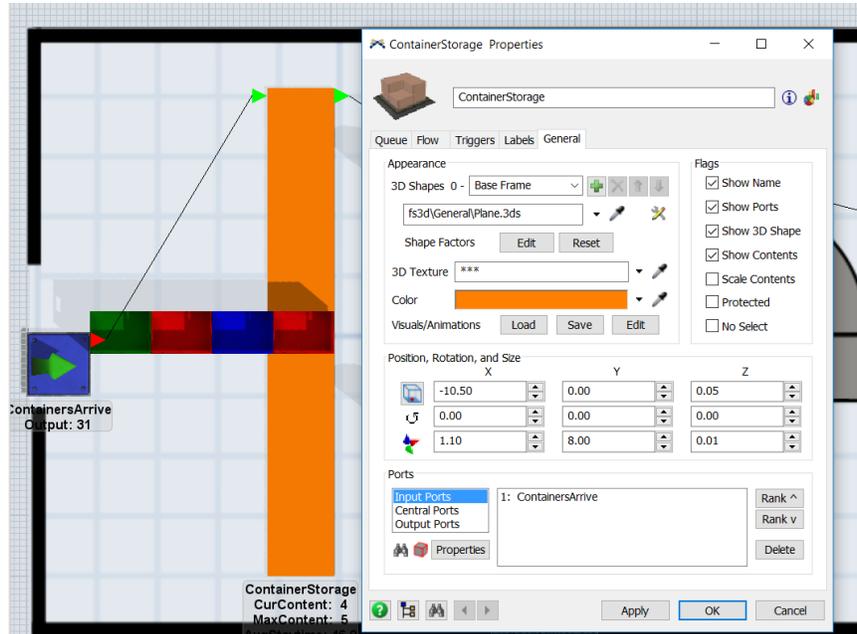
- As shown in the figure below, for the **3D Shapes** property in the **Appearance** section of the Queue's General tab, select **Browse ...** from the drop-down menu in order to find *FlexSim*'s fs3d folder. This folder contains a variety of graphics files and is located in the *FlexSim* folder that is within the Program Files folder, typically on the computer's C: drive.

Change the following properties on the General tab or in the Quick Properties panel when the object is selected.

- Using the drop-down menu, change the object's color to orange.
- Set the object's location to $x = -10.5$, $y = 0.0$, $z = 0.05$. Note the object is located a bit above the base of the modeling surface ($z = 0.05$, not $z = 0$) – this is so it can be seen above the layout object.
- Set the object's size to $x = 1.1$, $y = 8$, and $z = 0.01$. Note that the x value is set slightly larger than the container size.



If a Queue's size is less than or equal to the size of an item that is trying to fit onto the Queue, it will not fit; as a result, *FlexSim* will position it outside of the Queue. The logic will not be affected, but it will look strange.

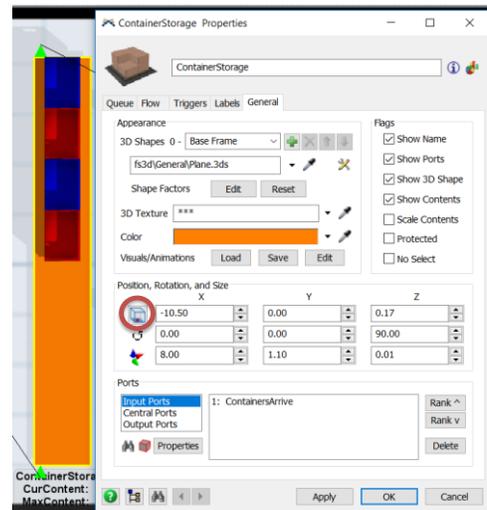


The above changes should place the object over the storage area on the imported layout.

When the model is run, as shown in the figure above, the containers do not conform to the shape of the storage area. This is because only the general shape of the object has been changed; logically items still queue in a horizontal line from the front of the object to the back of the object and beyond. Basically items line up from right to left.

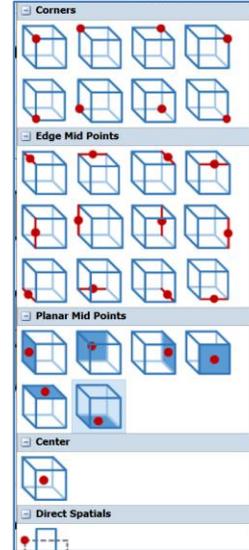
To correct this, make the following changes on the Queue's General tab. The revised properties are shown in the figure to the right. The figure also shows the effect on the model – containers now line up the same way they do in the real system.

- Rotate the Queue 90 degrees about the z axis so that it is vertically oriented and not horizontal. Set the z rotation to 90.0.
- Since the item is now rotated, reverse the sizes in the x and y directions; i.e., x = 8.00 and y = 1.10.
- Since the size and rotation have changed, the object needs to be repositioned; i.e., set the locations to x = -10.5 and y = 0.00.



This change may appear to be only for appearances, but in this case, the queue orientation will affect estimated system performance. Later, when an operator is added to move containers from the storage area to the machines, the location of the containers to be moved will affect the operator's travel time and thus performance.

You might notice a difference in the object's location property values between the General tab on the object and the Quick Properties window. This is due to the availability of different possible reference points in *FlexSim*. The possible reference points are shown in the figure to the right. The reference options are accessed by clicking on the small box-like icon to the left of the location values on the General tab, as indicated by the red circle on the interface in the previous figure.



ContainersArrive (Source) and NextProcess (Sink)

The Source and Sink shapes and sizes are not changed. They are just moved outside of the layout area – the Source is moved off to the left and the Sink is moved off to the right, both outside of the layout.



If you haven't already done so, save the model. Recall, it is good practice to save often.

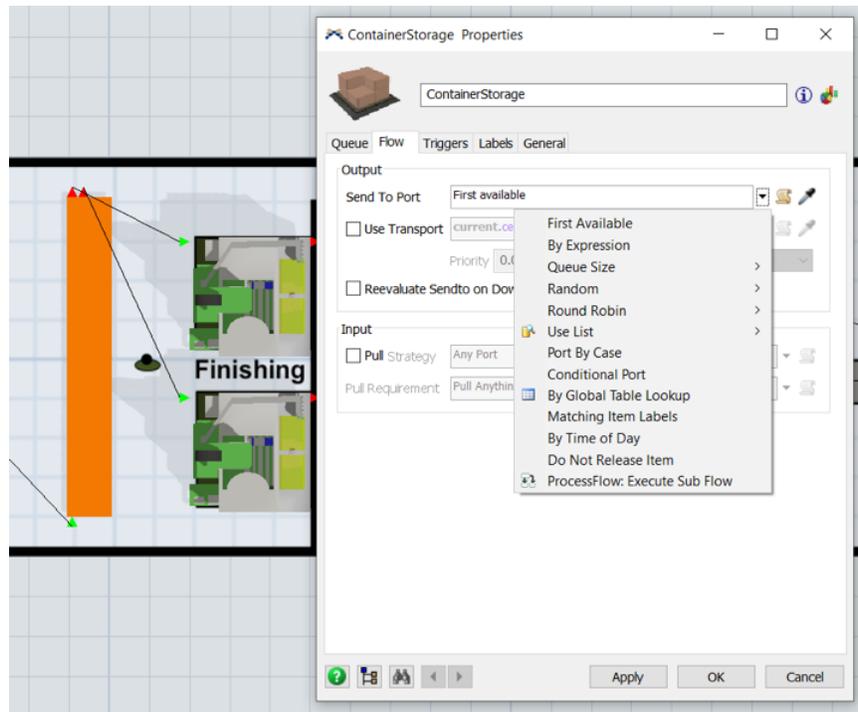
2 ALTERNATIVE ROUTINGS

Routings are a key part of modeling process flows. As items flow through a system they may follow different paths due to the current condition or state of the system. For example, different paths may be followed depending on the type of the object, the availability of a resource, the perceived waiting time to secure a resource, etc. Paths may also change randomly, such as a customer choosing between two available servers.

Until the previous step, all items (containers) followed the same routing path – they all flowed through the four basic objects in sequence, Source to Queue to Processor to Sink. But now there are two Processors in parallel; i.e., an item can go to either one. Thus, there is a routing decision at the Queue. Also, since the capacity of the Queue is limited, there is a routing decision at the Source – either go to the Queue or somewhere else if the Queue currently has no remaining capacity.

2.1 Choosing a route based on availability

The Queue for storing containers awaiting finishing has the option of sending a container to either finishing machine 1 or 2. For now, the default setting, FirstAvailable is used. The routing logic is controlled at the Send To Port trigger on the Flow tab, as shown in the figure below. Note in the figure that there are many available routing options to move items from the Queue to one of the machines.



The First Available option works as follows. When an item is ready to leave the Queue, it checks to see if the downstream object that is connected to Port 1 is available to receive the item. If it is, then the item is sent – in this case, the item leaves the Queue and arrives at the first machine in zero simulated time. Recall that by default, it takes no time to move from object to object in a model. If the first machine is not available, then the Queue checks to see if the downstream object that is connected to Port 2 is available to receive the item. If it is, then the item is sent – in this case, the item leaves the Queue and arrives at the second machine in zero simulated

time. If both machines cannot accept the item, then the item waits in the Queue until one of the downstream machines becomes available. Note that the port connections between objects drives the routing decision by setting the sequence for checking machines for availability.

Using the FirstAvailable routing logic usually results in the object connected to the first port receiving more items since it is always checked first. Similarly, the object connected to the last output port will receive the fewest items.

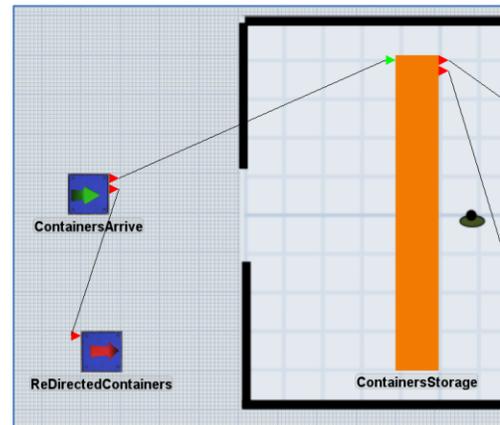
2.2 Choosing a route based on current system conditions

The container storage area has a limited capacity (Maximum Content's property value on the Queue is 50). Therefore, provisions must be made for the condition when the Queue is at capacity and cannot accept any more containers. In this case, containers will be diverted to a second Sink, named ReDirectedContainers, if the Queue is full.

Implement this in the model as follows and as shown in the figure to the right.

- Drag out a Sink object from the Object Library onto the modeling surface and name it **ReDirectedContainers**.
- A-connect the Source to the new Sink.

As shown in the figure to the right, there are now two output port connections from the Source, one going to the Queue and one going to the second Sink that was just created. As with the case above, the default **FirstAvailable** is used so that items go out Port 1 to the Queue if there is available capacity and go out Port 2 to the Sink if the Queue is full.



Note that if the connections are reversed the model would behave differently. That is, if the Source and Sink are connected through Port 1 and the Source and Queue through Port 2, then all items would go to the Sink and none to the Queue since a Sink is usually always available and has no capacity limits. Therefore, it is important to be careful on the order in which objects are connected.



If you haven't already done so, save the model. Recall, it is good practice to save often.

3 TASK EXECUTERS

Task executers (TEs) are mobile or dynamic resources – they move about in a model, typically moving flowitems between objects. There is a section in the Object Library, right below Fixed Resources, that contains different types of TEs, including Operators, Transporters, Cranes, Robots, etc. All of these have the same functionality and basically differ in the way they move; e.g. Operators can move in any direction (x, y, and z), whereas Elevators can only move vertically (z direction).

TEs can be used to represent complex behaviors, but only the most basic properties and a single type of TE, the Operator, are covered here. Prior to discussing how to implement TEs in a *FlexSim* model, a few general concepts are introduced.



[Section 6.1] **Mobile resource objects (task executers)**

3.1 Basic task executer concepts

TEs have speeds (and accelerations); therefore, the location of model objects and distances between them now become very important.

As the name indicates TEs execute tasks. Sets of tasks for performing a specific operation, called a **task sequence**, are passed to TEs, typically from fixed resources (sources, processors, etc.). *FlexSim* includes default task sequences, but as in most things in *FlexSim*, they can be customized. For now, only default task sequences are considered. For transporting items a TE needs to move items from one object (FromObj) to another object (ToObj). To do so, by default, a TE executes the following sequence that is sent to the TE from the FromObj.

Travel from current location to FromObj
Load item from FromObj
Travel from FromObj to ToObj
Unload item to ToObj

Tasks other than travel, load, and unload are available in *FlexSim*. However, developing custom task sequences is an advanced topic and is covered later in this primer.

Fixed resource objects communicate with TEs via special connections, referred to either as a Center-connection or S-connection. It is called a Center-connection because the connection is made between the center ports of the two objects. It is called an S-connection because the connection between the two objects is made by holding down the S key while dragging the mouse between the two objects.

Objects have center ports in addition to the input and output ports that were introduced earlier. Center ports are not for item flow, but are used for communications between objects. Communications can be bidirectional between the objects; this is in contrast to the unidirectional flow between input and output port connections (A-connections).

A TE will oftentimes receive more requests to perform task sequences than can be met at that time. For example, a TE may receive a request while performing a task sequence. Therefore, TEs can queue requests to

perform task sequences and can use different means to process items in the task queue, e.g. first-in, first-out or priorities.

TEs not only receive tasks, but can send them to other TEs. Thus, TEs are like working managers – they will carryout a task-sequence request unless they are busy when they receive the request. If they are busy when they receive the request, they can send it to an available TE. If all associated TEs are busy, the receiving TE puts the task-sequence request into its work queue. To implement this, decide which TE is the working manager and connect it to all associated TEs via A-connections (from the working manager TE to the associated TEs).

If all TEs are considered the same and no one is the manager, then a Dispatcher object is used. The only function of the Dispatcher is to allocate work and maintain the work queue for all of the associated TEs. In this case, the fixed objects communicate with the Dispatcher and not directly with the individual TEs. Therefore, the fixed objects are connected to the Dispatcher with S-connections through their center ports and the Dispatcher is connected to each TE with an A-connection (from Dispatcher to TE).

By default, a TE travels between objects in a straight-line path, traveling the shortest distance between two objects. A TE’s default travel path does not consider other objects in its path. However, there are several alternative means to control the travel path of a TE – one, using path networks, is discussed in a later sub-section and another approach, using the A* algorithm, is discussed in a later section once the conveyors and the packing station are included in the model.

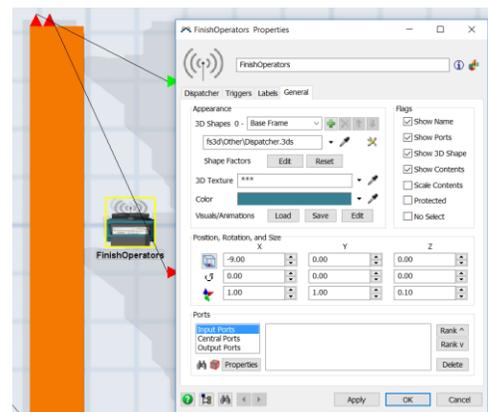
3.2 Adding a finish operator to the model

Since it is possible that more than one finish operator will be needed, use both the Dispatcher and Operator objects.

 Begin with the basic model from the previous section, named Primer_2-1, and **Save As** Primer_2-2. Basically make a copy of the model so that it can be customized and the original model is retained.

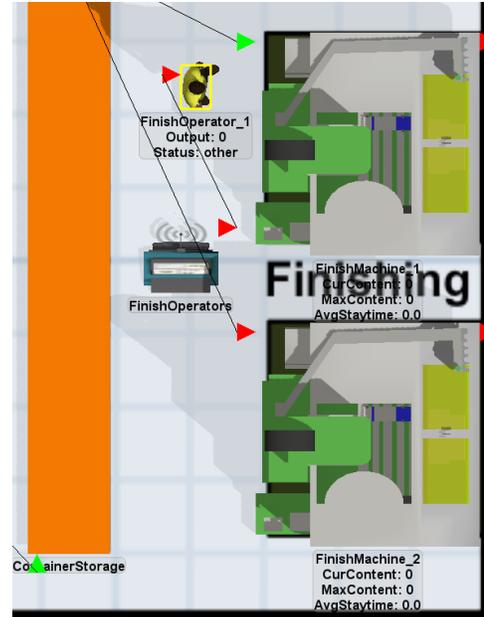
➤ Drag out a Dispatcher object from the Task Executors section of the Object Library to the modeling surface. As shown in the figure to the right:

- Name the object **FinishOperators**.
- Locate the object where the person symbol appears on the layout in the finish area ($x = -9.0, y = 0.0, z = 0.0$). It can be placed anywhere, but this is a convenient location in this example.
- Resize the object using the size settings $x = 1.0, y = 1.0, \text{ and } z = 0.1$. This minimize the object’s presence since the Dispatcher object is not a real entity in the system.



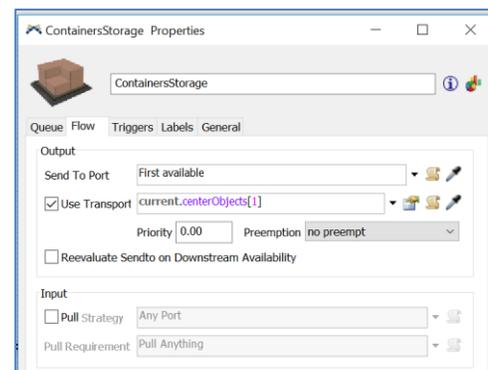
The Queue makes requests of the Dispatcher, i.e. sends task sequences, to move items from the Queue to one of the Processors. The Dispatcher then sends the request (task sequence) to one or more Operators that it manages. Implement this aspect in the model as described below and as shown in the figure below and to the right.

- Make an S-connection between the Dispatcher and the ContainerStorage Queue by holding down the S key, selecting the Dispatcher so that is highlighted by a yellow box, dragging the mouse to the Queue, and releasing once the Queue is highlighted with a yellow box. As shown in the figure to the right, there should be a connection between the two objects' center ports. When making center-port connections, the order of connections is not important since it involves bi-directional communications; i.e., the connection can be made from the TE to the fixed object or from the fixed object to the TE.
- Drag out an Operator object from the Task Executors section of the Object Library to the modeling surface. At the moment, it can be placed anywhere.
- Name the object **FinishOperator_1**
- Make an A-connection from Dispatcher to FinishOperator_1 by holding down the A key, selecting the Dispatcher so that is highlighted by a yellow box, dragging the mouse to the finish operator, and releasing it once the Operator is highlighted with a yellow box. As shown in the figure to the right, there should be a connection between the Dispatcher's output port and the Operator's input port. Make sure the connection is from the Dispatcher to the Operator since it is a one-way flow of information.



Now that all of the objects can communicate, the Queue needs to be able to send a task sequence to the Dispatcher telling it what to do; in this case, the Queue requests the TE to transport an item from the Queue to an available finish machine. This is accomplished as follows and as shown in the figure.

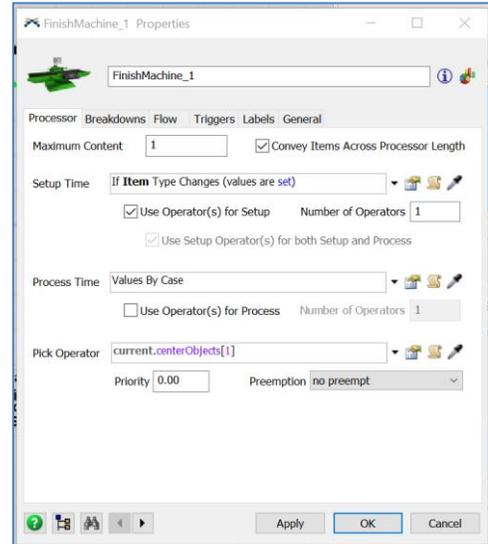
- On the Flow tab of the Queue object (ContainerStorage), check the **Use Transport** box. By default, the task sequence goes to whatever object is connected to the Queue's center port; in this case, that is the Dispatcher. This action is accomplished by the default **Use Transport** property value, which is the `current.centerObjects[1]` FlexSim command. This command is interpreted as the object connected to the current object's center port. In this case, the current object is the Queue and the object connected to the first center port is the Dispatcher.



Once the Dispatcher receives the task sequence it sends it to the Operator when the Operator is available.

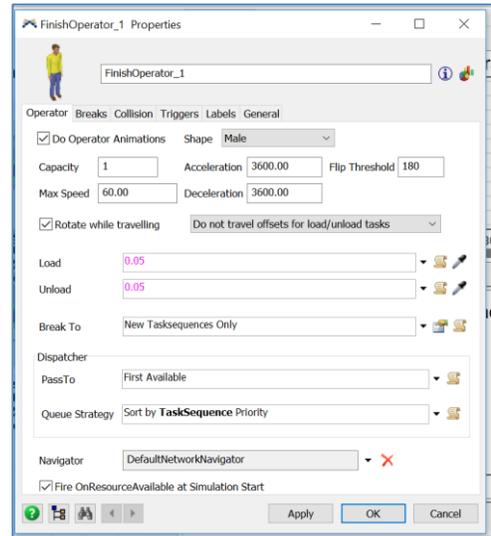
The Operator also needs to perform the setup operation at each finish machine. This is done similar to the process just described. As shown in the figure to the right and as described below, modify each Processor as follows.

- On the Process tab, check the **Use Operator for Setup** box that is below the Setup Time property..
- The **Pick Operator** property, near the bottom of the Process tab, can remain the default. Each Processor will use the object connected to its first Center port, which will be the Dispatcher.
- Make a S- or Center connection between each Processor and the Dispatcher.



Since the performance of the system depends on speeds and distances, Operator and other Task Executer properties are important. The default value for an Operator’s speed is 120 meters/minute. This seems a bit fast for this operation since the average walking speed is about 84 meters/minute. Therefore, use a value somewhat slower than average walking speed, 60 meter/minute.

- As shown in the figure to the right, change the **Max Speed** value, on the Operator tab on FinishOperator_1, from the default of 120 to 60.

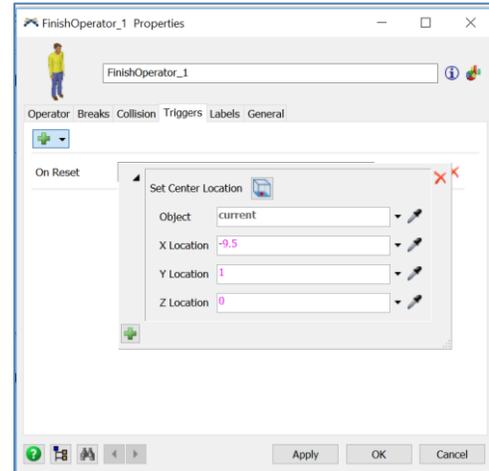


Also, consider for a transport task sequence the time it takes to load a flowitem at the originating object and unload a flowtime at the destination object. The default time is 0.0. For this example, change each to 3 seconds.

- Also on the tab as above, change the **Load** and **Unload** property values to 0.05 minutes (3 seconds).

Since the Operator object is mobile it can be in any location on the modeling surface when a running model is stopped. For consistency in comparisons and analyses, it is good practice to have the Operator start at the same location each time a model is **Reset** and **Run**. Therefore, in this case, set the location of the FinishOperator_1 to a location near finish machine 1 and the container queue.

- As shown in the figure to the right, on the Operator object's Trigger tab, add an **On Reset** trigger and then select the **Set Location** option. Set the location values to $x = -9.5$, $y = 1$, $z = 0$. The Operator moves to this location whenever the **Reset** button is pushed.



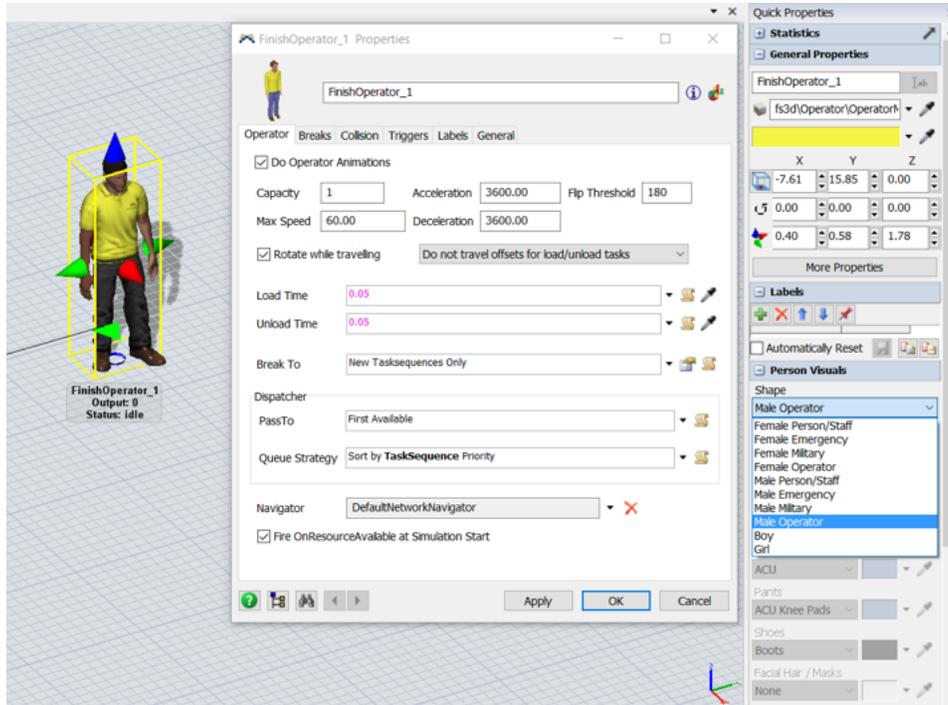
By default, the last task in the Operator's transport task sequence is Unload; therefore, it will become available for another task once the Unload operation is complete. If there are no other tasks waiting, the Operator will remain at the Unload location until it receives another task. For now, this is okay, other options can be considered later.



If you haven't already done so, save the model. Recall, it is good practice to save often.

- **Reset** and **Run** the model. Note that the Operator carries containers between the Queue and Processors and remains at the Processor to perform the setup operation. You may need to slow down the model's speed to see this (e.g. Run Speed = 0.1) or stop the model as a transport is taking place. Also, note that the Operator returns to the same location when the model is reset.

Another feature of the Operator object is that its form is very customizable. The form or shape of the default Operator object is that of a **Male Operator**. The value of the **Shape** property is shown in the Quick Properties window when an object is selected. Note that in the figure below the only customizable feature of the **Male Operator** (in the Quick Properties window) is the **Skin Texture** property; all other properties, e.g. **Head, Hair/Hat, Shirt, Pants** are all grayed out. This is also true of the **Female Operator**. However, all other **Shape** values, as shown in the drop-down menu in the figure below, are customizable, such as **Female Person/Staff, Female Emergency, ... Boy, Girl**. While changing the shape/form of the Operator object does not affect the system's behavior, it does help those not familiar with simulation better identify with, and oftentimes have more confidence in, the model. At this point in the primer, no changes are made to the Operator's shape; the default Operator properties are sufficient for the problem domain.



Another feature of Operator object, and most other objects in *FlexSim* as well, is that custom **animations** can be defined. Animations are a sequence of object movements that are triggered as a simulation runs. For the Operator, the default animations are walking and carrying a flowitem. Animations are created, edited, imported, and exported via the Animation Editor, which is accessed via the **Visuals/Animations** property on the object's General tab. At this point in the primer, no changes are made to the Operator's animations; the default Operator properties are sufficient for the problem domain. However, animations are introduced via an example later in the primer.

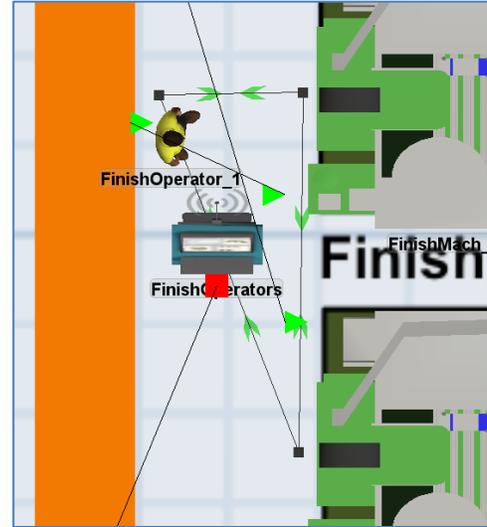
One visual that is quite useful, and is easy to control, is an object's color. The color of an object can be changed through drop-down menu options on a number of triggers. For the Operator object, the shirt color is what is changed. This can either be done statically via the **Color** property on the General tab or via triggers, e.g. set the color of a flowitem when it is created based on its type or change an object's color when it is down or not available. Using colors is a very effective way of testing and validating a model and is used extensively in this primer.

3.3 Controlling task executor travel with path networks

By default, when TEs travel between objects in a model, they use straight-line paths and are not inhibited by other objects. To better match reality, and to provide more realistic performance measures, TEs are constrained to defined paths that are formed by what is referred to as a **path network**, which is similar to a roadway. A path is defined by connecting a series of nodes, referred to as **network nodes**. The nodes are fixed objects and are located in the Travel Networks section of the Object Library, just below the Task Executor section.

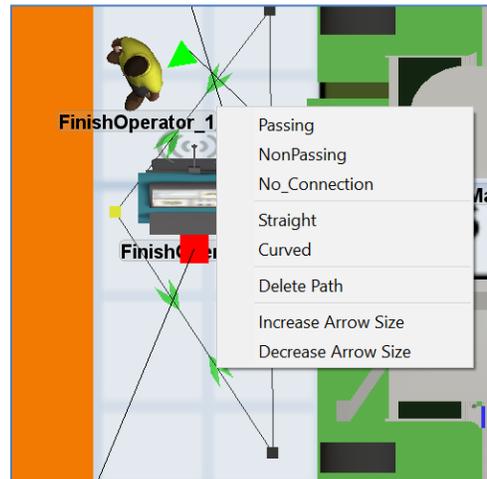
Only a simple network is constructed here since later, once more objects are included in the model, a different means of controlling TE travel will be used. For now, travel will be controlled with three nodes, one at each finish machine and one at the queue (ContainerStorage).

- Drag three Network Nodes from the Object Library to the modeling surface and place them in the locations shown in the figure to the right. Network Nodes are small square objects.
- In order to get exact placement of the Nodes, you may want to uncheck the **Snap To Grid** box under **View Settings** in the Quick Properties window. Otherwise, nodes will be placed on the closest grid point. Recall this interface is accessed when clicking anywhere on the modeling surface where there is not an object.
- Since it is good practice to name objects, name the Nodes: **nn_ContainerStorage**, **nn_FinMach_1**, and **nn_FinMac_2**. No other properties need to be modified.
- The path segments, oftentimes referred to as *edges*, are formed by connecting the Nodes with A-connections, as shown in the figure to the right,



The green arrows on the edges denote the permitted direction of travel. By default, the edges are bidirectional. As shown in the figure to the right, right clicking on an arrow provides the following options:

- **Passing** permits multiple TEs on the same path to pass or move past each other. This is the default.
- **NonPassing** restricts faster TEs from passing a slower TE on the same path.
- **No_Connection** restricts the direction of travel.
- **Straight** declares edges, or travel paths between nodes, to be straight lines. This is the default.
- **Curved** declares edges, or travel paths between nodes, to be curves that can be adjusted.
- **Delete Path** eliminates an edge between nodes.
- **Increase Arrow Size** makes the directional arrows larger.
- **Decrease Arrow Size** makes the directional arrows smaller.



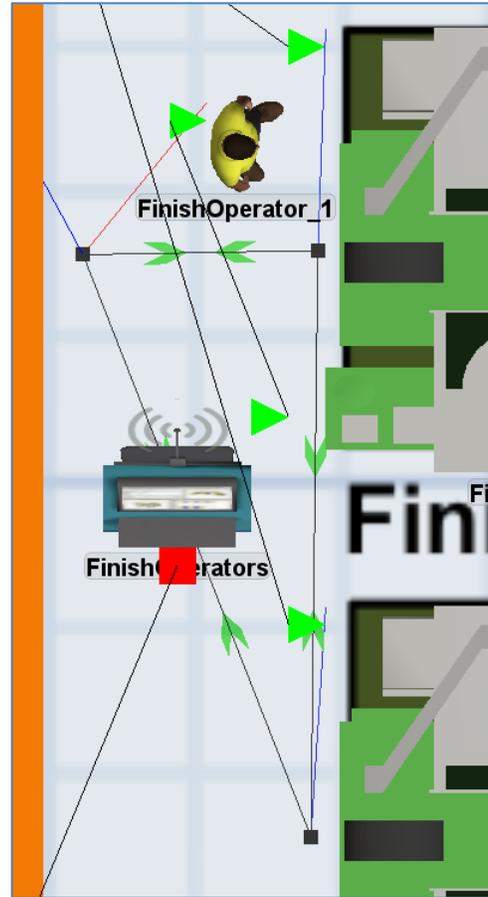
In order for the network to function, fixed objects must be associated with network Nodes. This is so that a TE knows where on the network an object is located. For example, if an Operator is to travel to a Queue in order to pick up an item, and the TE must stay on the network path, it must know which node to travel to in order to reach the Queue.

Objects are associated with Nodes by making an A-connection between them. When connected, a blue line denotes the connection, as shown in the figure to the right.

- A single object can be associated with multiple Nodes; e.g., if an object can be accessed from more than one side or if an object is associated with multiple networks.
 - Similarly, a single network Node can be associated with multiple objects, e.g. if a TE interacts with a group of objects from a common point.
- Connect each network node to its associated object; i.e., there are three connections. For example, one connection is from Network Node nn_FinMach_1 to Processor FinishMachine_1.

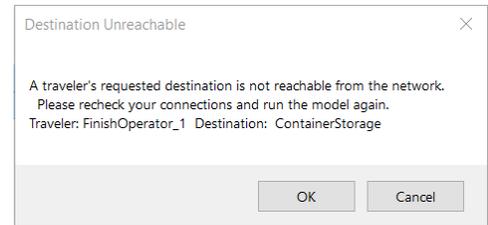
To use a network, each TE must be associated with the network. This is accomplished by an A-connection between the TE and only one of the Nodes on the network. This is considered the TE’s “home” Node – where it will be located when the model is Reset. The resulting connection between the TE and the Node is indicated by a red line, as shown in the figure to the right.

- Connect the Operator to the Network Node associated with the Queue, ContainerStorage.



When the model is run, an error will result if any of the objects where the TE needs to perform a task is not associated with a network node.

An example error message is shown to the right. In this case, the Queue ContainerStorage has not been connected to its associated Network Node because the finish operator needs to travel there in order to pick up a container, but the Operator doesn’t know where to go on the network to find the Queue.

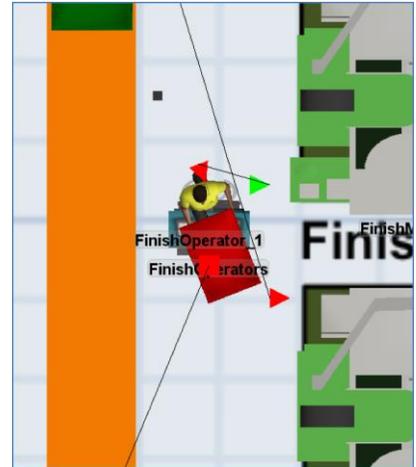


If a Task Executer is not connected to the network, then the TE will not travel on the network, but there will be no error messages. It should be obvious by watching the model that the TE is not constrained by the network paths.

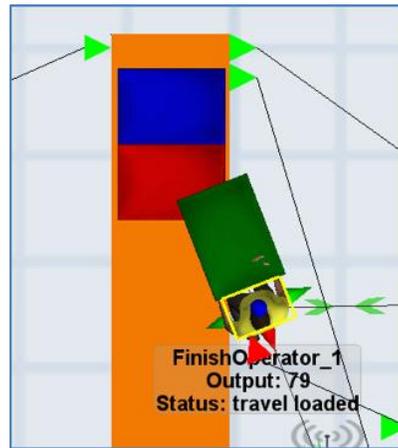
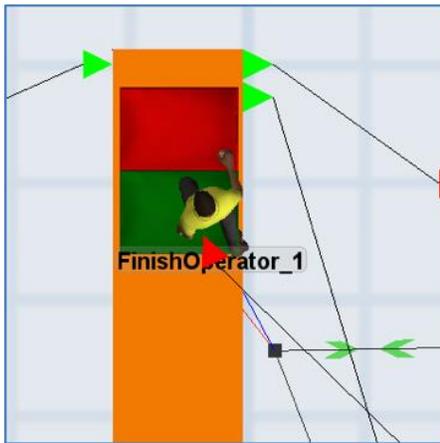
The network can be hidden by right clicking on one of the Nodes and selecting **Network View Mode** from the drop-down menu and then selecting one of the following options:

- **Show All** is the default and what is seen when the networks are created.
- **Edges** show only the connecting lines and not the Nodes.
- **None** shows only one Node, the one selected, and no edges.

The last case is shown in the figure to the right.



A TE will remain on a network path until it gets to a Node that is associated with the task it is executing. At the Node it can either **travel offset** or remain at the Node to execute the task. By default, it travels offset. This is illustrated in the figure to the left below. In this case, the Operator leaves the Node and travels to the object to load the item. If the TE does not travel offset, it performs its task at the Node, as shown in the figure to the right below. In this case, the Operator acquires the container from the Node no matter how far it is from the Node. Whether a TE travels offset or not is a property on the Operator tab.



For the primer example:

- Change the property on the Operator to **Do not travel offsets for load/unload tasks**. This invokes the case shown in the figure to the right above.



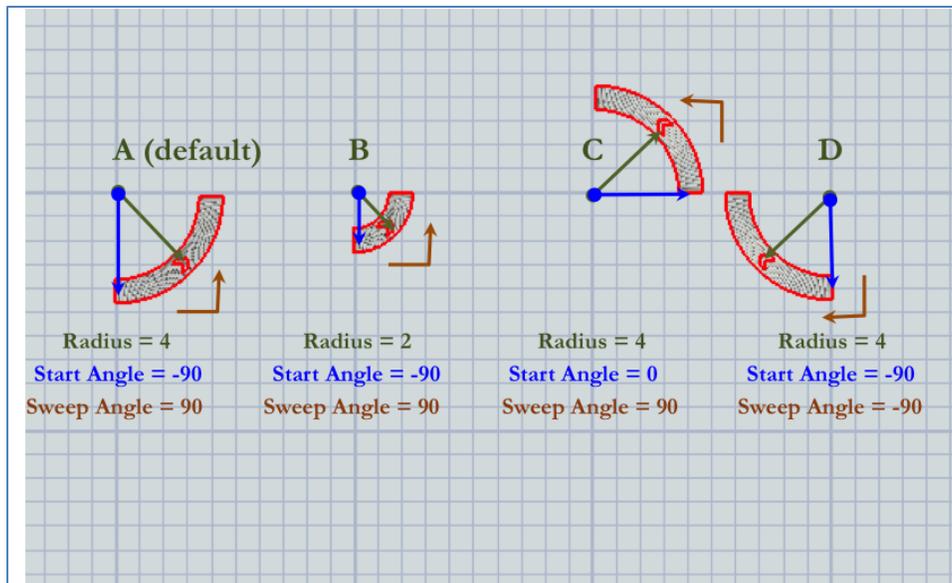
If you haven't already done so, save the model. Recall, it is good practice to save often.

4 CONVEYORS

A common way to move items through a system is via conveyors. Conveyor systems are often complex, but *FlexSim* has extensive capabilities to model such systems. Of course, for this introductory primer, only the basics are considered. Objects associated with conveyors are in a section in the Object Library just below Travel Networks. At this time, the primer considers only the first three conveyor objects: Straight Conveyors, Curved Conveyors, and Join Conveyors. These object can be used to represent either roller or belt conveyors.

Conveyors are much like the Fixed Resources, e.g., Source, Queue, and Processor, with a few key differences.

- Conveyors have two ends, referred to as Start and End, that can be modeled independently. Flow items enter the conveyor object at the Start and leave at the End.
- Conveyors cannot be rotated in the x, y, or z directions as fixed resources can. However, the height (z location) can be changed at either end, resulting in a ramp or elevated conveyor.
- Each conveyor has a direction of travel, indicated by an arrow on the conveyor surface that is visible when the object is selected.
- Curved conveyor shapes are defined by three properties, which are described below and illustrated through a few examples in the figure below. These properties are edited through the Quick Properties interface, rather than double clicking on the object. The properties can also be changed by selecting the conveyor segment and dragging the resizing arrows.
 - **Radius** is the size of the curvature of the conveyor, as measured from the center of a hypothetical circle to the midpoint of the conveyor width.
 - **Start Angle** is the rotation, in degrees, to the location of the Start of the conveyor
 - **Sweep Angle** is the rotation, in degrees, from the Start to the End of the conveyor.



In the figure above,

- Case A shows the default setting; i.e., those of the curved conveyor section when it is dragged out from the Library. The conveyor segment starts 90 degrees below horizontal (Start Angle = -90) and is formed

by sweeping up to the horizontal (Sweep Angle = 90) with a radius of 4 grid units (meters in this model).

- Case B is the same as Case A, but with a smaller radius.
- Case C is the same as Case A but starts at a different location (Start Angle = 0).
- Case D is the same as Case A except it sweeps in the opposite direction (Sweep Angle = -90).

Obtaining the proper shape sometimes involves experimenting with the settings.



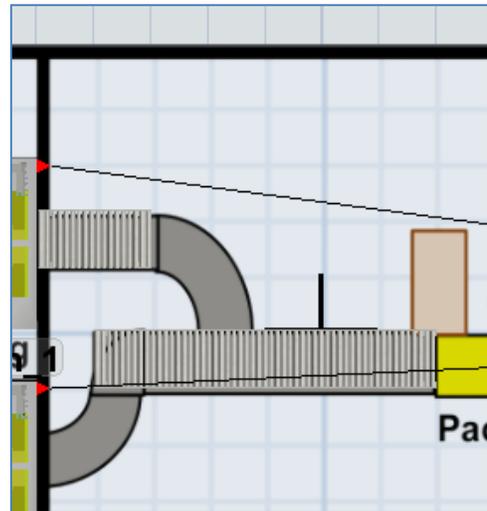
Begin with the basic model from the previous section, named **Primer_2-2**, and **Save As** **Primer_2-3**. Basically make a copy of the model so that it can be customized and the original model is retained.

4.1 Straight and curved conveyor sections

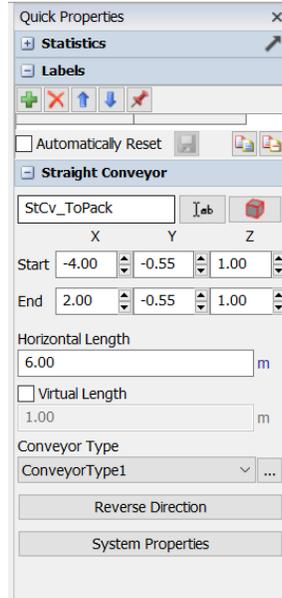
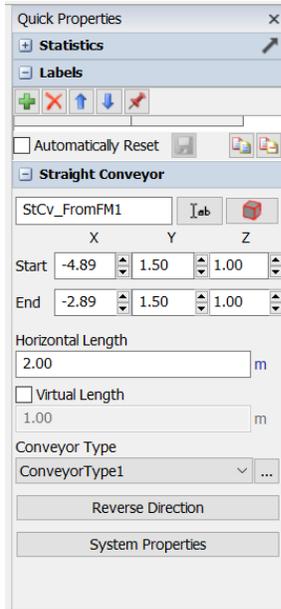
For now, only the conveyors up to the Packing area are modeled. Again, this is an example of the best practice of incremental model building – build small segments, test and validate, and then consider the next aspect of the system.

As described earlier, when precisely placing objects it is best to work in an environment with the following view characteristics:

- 2D rather than 3D view. Right click on the modeling surface, select **View**, and then **Reset View**.
- Flat rather than perspective view. Uncheck the **Perspective Projection** box in Quick Properties.
- Drag out two Straight Conveyor sections.
 - In the Quick Properties window, rename the two sections to **StCv_FromFM1** (straight conveyor from Finish Machine 1) and **StCv_ToPacking** (straight conveyor to the Packing area).
 - Set the first segment’s length to **2** and the second to **6** by selecting the conveyor segment and in Quick Properties window change the **Horizontal Length** from the default of 10 (meters in this example).
 - Position both segments on the layout where the conveyors are drawn, as shown in the figure to the right.



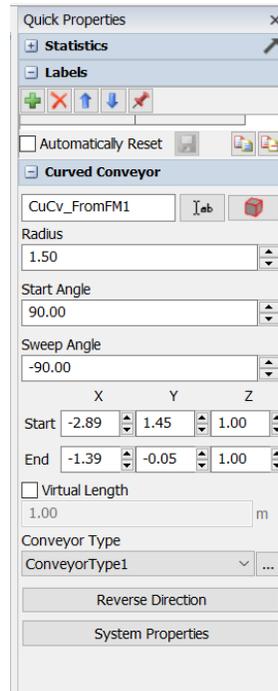
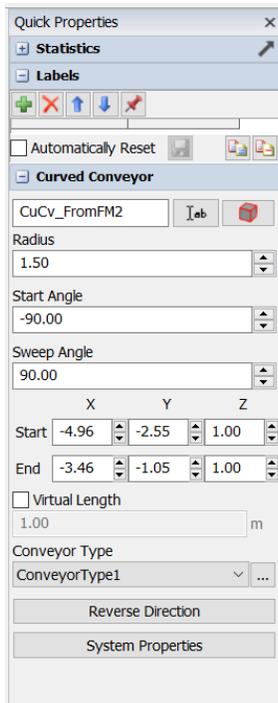
The interfaces for each segment is shown below.



➤ Drag out two Curved Conveyor sections.

- Name one **CuCv_FromFM2** (curved conveyor from Finish Machine 2), set its **Radius** to 1.5, and locate the object in the proper place on the layout. Note and retain the default settings for **Start Angle** and **Sweep Angle**.
- Name the other **CuCv_FromFM1**, set its **Radius** also to 1.5, change the settings for **Start Angle** to 90 and **Sweep Angle** to -90, and locate the object in the proper place on the layout.

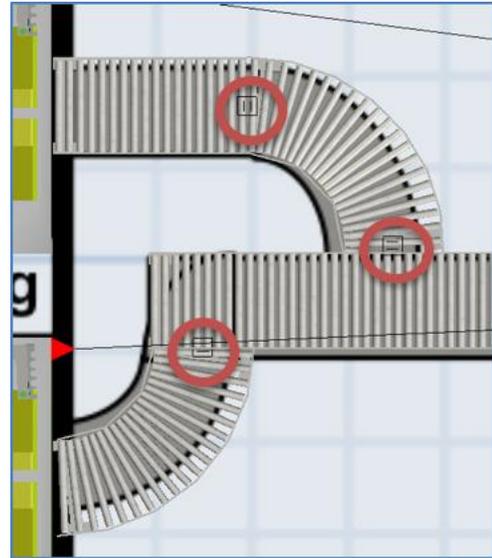
The interfaces for each segment is shown below.



4.2 Connecting conveyor sections

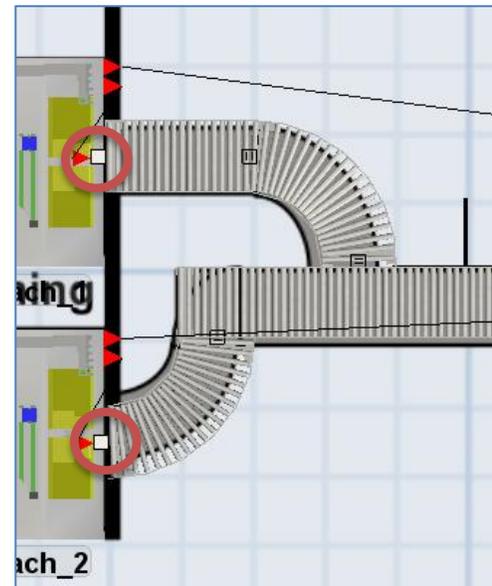
Connecting conveyor sections to each other is different than connecting other *FlexSim* objects. When one conveyor section is selected and dragged close to another section they automatically snap together. As shown in the figure to the right, and highlighted by the circles, a **Transfer** connects the two sections together. The Transfer is the small square that overlaps the two conveyor sections. You may need to zoom in to see the Transfers.

- Check to be sure all segments are connected as shown in the figure.



Conveyor sections are connected to basic *FlexSim* objects in the same way as described throughout the primer, with an A-connection. An Entry Transfer is automatically added when a Fixed Resource object is connected to a conveyor section and an Exit Transfer is automatically added when a conveyor section is connected to a Fixed Resource. In the figure to the right, the Transfers are small white squares at the start of the conveyor sections at the finish machines. They are highlighted by the circles.

- A-connect FinishMachine_1 to StCv_FromFM1
- A-connect FinishMachine_2 to CuCv_FromFM2



Since the conveyors now move the flowitems to the Sink, remove the connections from both finish machines to the Sink.

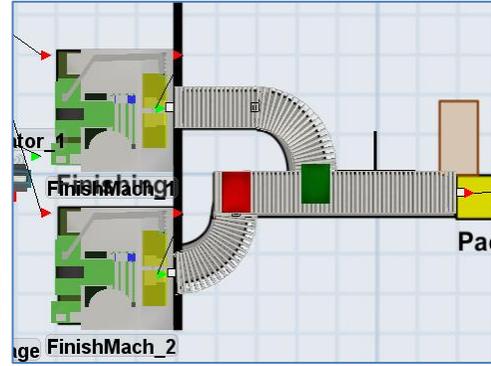
- Q-disconnect the FinishMachine_1 and NextProcess objects. The Q-disconnect is the opposite of an A-connect; therefore, hold down the Q key, select the finish machine and then select the Sink.
- Using the same process as above, Q-disconnect the FinishMachine_2 and NextProcess objects.

Connect the Straight Conveyor StCv_ToPacking to the Sink. Note that connecting the Straight Conveyor to the Sink creates an Exit Transfer at the end of the conveyor.

- A-connect StCv_ToPack to NextProcess, the Sink.

➤ **Reset and Run** the model.

The operation should resemble the figure to the right. Note how the containers move on the conveyor system. As shown in the figure, the containers change orientation because objects are transported based on their orientation when entering a conveyor section. When entering the conveyor system from the finish machines the leading edge of the containers is the short dimension. However, as the containers move onto the straight conveyor that goes to the packing area, the leading edge of the container is now the long dimension - that is the orientation when the items joined the straight section of conveyor from the curved sections. If the orientation should remain the same on all sections of the conveyor, then the Join Conveyors object can be used. This is defined in the next section.



If you haven't already done so, save the model. Recall, it is good practice to save often.



The *Transfers* (between two conveyor sections and between a Conveyor section and a Fixed Resource object) are actually objects as well. They contain advanced properties that can be accessed by double clicking on a transfer.

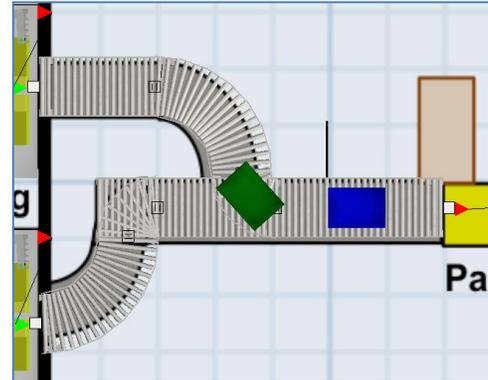
4.3 Join Conveyors object

The Join Conveyor object is more of a tool than an object. It connects two conveyor sections with a curve. In this case, we'll use the Join Conveyor tool for where the curved conveyor from each finish machine joins the main straight conveyor that goes to the packing area.

- Click on the Join Conveyors object in the Library to enter into *conveyor joining mode*. When in conveyor joining mode, the mouse pointer changes to a plus sign with a Join  Conveyors icon next to it. To exit this mode, press the ESC key.
- Click near the center of the first conveyor section to be joined, CuCv_FromFM1, and a yellow line will follow the mouse as it moves, similar to the line that appears when connecting two *FlexSim* objects. Click on the second conveyor section that is to be joined, StCv_ToPack. As a result, a new curved conveyor will appear between the two conveyor sections and two transfer points will also appear on both ends of the new conveyor section.
- Use the ESC key to exit conveyor joining mode.
- Repeat the above steps to join CuCvFromFM2 and StCv_ToPack, where the second finish-machine conveyor joins the straight conveyor.

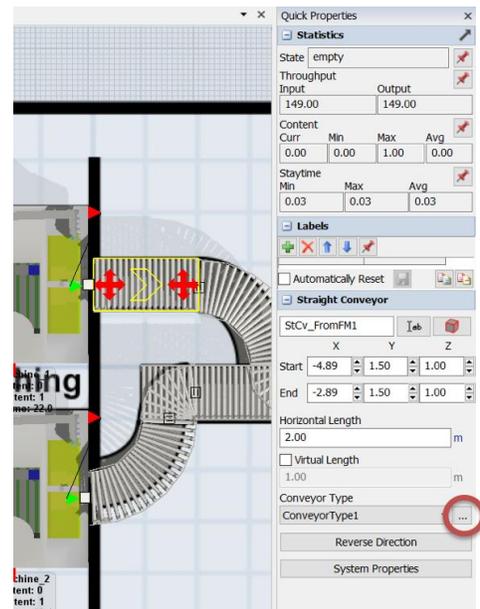
- **Reset** and **Run** the model.

The operation should resemble the figure to the right. Note how the containers now maintain the same orientation with the small edge of the container as the leading edge.

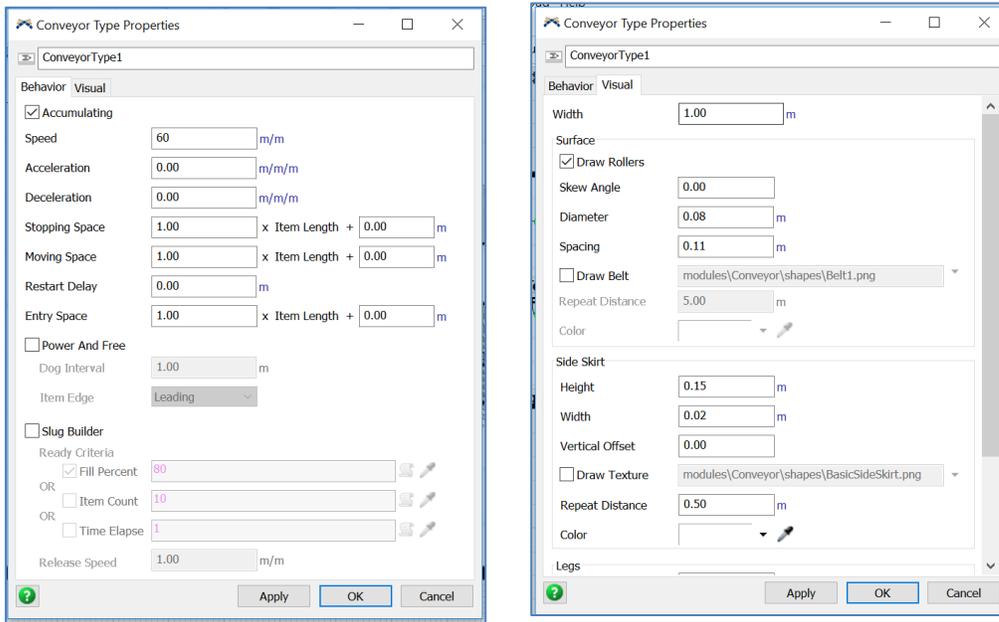


4.4 Conveyor types

Each conveyor segment belongs to a Conveyor Type. By default, all conveyors are of type Conveyor Type 1. The type is accessed through the Quick Properties window. Accessing the type is illustrated in the figure to the right for the straight-conveyor from Finish Machine 1, named StCv_FromFM1. Conveyor Type is accessed through the ... button, just to the right of **Conveyor Type** on the object's Quick Properties window; this is highlighted by the red circle in the figure to the right..



As shown in the figures below, the Conveyor Type Properties controls many aspects of the conveyor's operation. The tab in the left figure, named Behavior, is used to set speed, spacing, etc. The tab shown in the figure to the right, named Visual, is used to set appearance properties.



- On the Behavior tab, change the **Speed** from the default of 60.0 to 90 meters per minute. Since all conveyor segments in the model are of Type 1, they all have a speed of 90 meters per minute. To model additional conveyors in the system that have different speeds, create a new type and assign all applicable segments this type value.
- Note, also on the Behavior tab, that by default the conveyor type is accumulating since the **Accumulating** check box is selected. This means that the conveyor acts like a *roller conveyor*. In a roller conveyor, an item travels along the conveyor until the end and then stops if it cannot be removed (downstream object or transport is not available). Subsequent items on the conveyor continue to flow on the conveyor and stop behind the one in front.

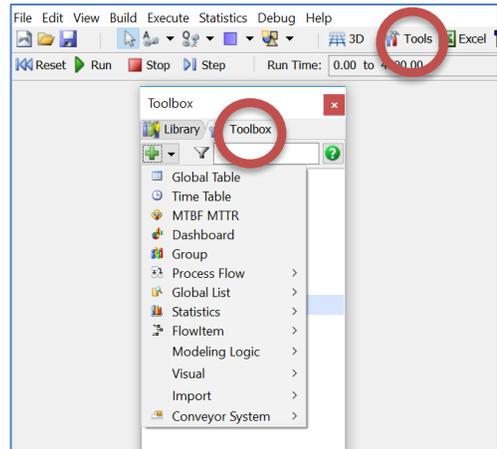
If **Accumulating** is not checked it is a *non-accumulating conveyor* which means it operates like a *belt conveyor*. In a belt conveyor, as in a roller conveyor, an item travels down the conveyor until the end and then stops if it cannot be removed (downstream object or transport is not available). However, on a belt conveyor when one item stops, the belt stops and then all other items on the conveyor stop in their current location.
- Also, note a few of the properties on the Visual tab, albeit none of their values are changed at this time.
 - **Width** is the conveyor’s width with a default of 1 grid unit; in this example, this represents 1 meter.
 - The **Surface** section is where the conveyor’s rendering properties are set for drawing rollers, belts, or a texture, setting color, etc.



If you haven’t already done so, save the model. Recall, it is good practice to save often.

5 GLOBAL TABLES

The primer now switches from *FlexSim* modeling *objects* to *FlexSim* modeling *tools*. As the name suggests, tools are means to support various aspects of simulation modeling and analysis. Tools are accessed through the Toolbox Library. As shown by the circles in the figure to the right, the Toolbox Library is accessed either through: (1) the tab next to the Objects Library or (2) the Tools icon on the Main Menu bar.



Also, shown in the figure to the right, tools are added to a model through the + button in the upper left portion of the interface. Tools include Global Table, Time Table, MTBF/MTTR, etc. These are discussed in subsequent sections of the primer; however, since this is an introduction, not all of the tools are discussed. One tool, Dashboards, was discussed in Section 6.2 of Part 1.



It is good modeling practice to separate data (property values) from objects, especially if the values are to be changed, e.g. when conducting what-if analyses and experimentation.

Tables provide a very convenient means for storing data because during model execution the model can reference a table cell when it needs a parameter value. Tables can also be used to store output from a model. *FlexSim* provides a link to *MSE_{excel}* so that data can be easily exchanged (imported or exported) between a model and *MSE_{excel}* via Global Tables.

Two examples are provided in this section in order to introduce the use of Global Tables. The first example uses a table to store process times for the different types of containers; the second uses tables to store the container product mix, the expected percentage of each type of container.



[section 5.3] **Managing data through tables**



Begin with the basic model from the previous section, named Primer_2-3, and **Save As** Primer_2-4. Basically make a copy of the model so that it can be customized and the original model is retained.

5.1 Using tables to store process times

Process times at the finish machines are stored in a table rather than within each Processor object. For one thing, this facilitates changing parameter values. A value only needs to be changed in the table and is propagated to all objects that reference that table cell.

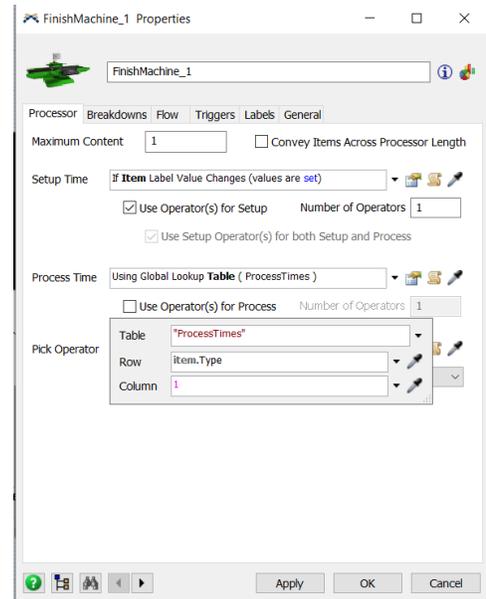
- Using the green + on the Toolbox, create a Global Table and customize it to reflect the figure to the right.
- In the Quick Properties window, change the name of the table from GlobalTable1 to a more meaningful name, say **ProcessTimes**.
- Also in the Quick Properties window set the number of **Rows** to **3** and leave the default number of **Columns** at **1**.
- Set the cell values as shown - the process times for container types 1 through 3, respectively: 15.0, 20.0, and 30.0 minutes.
- Change shaded “headings” for the rows and columns from the default Row 1, Row 2, ... and Col 1 to the more descriptive names shown in the figure. Note that the headings are just text for information purposes and are not table values. However, they are very helpful in understanding what data tables contain.
- The row heights and column widths can be changed by dragging the lines dividing the cells.

	Finish Times
Type 1	15
Type 2	20
Type 3	30

Note that the table opens in a tabbed window, like the model. The view may be closed by pressing the **X** in the upper-right corner of the interface. The window can be reopened by double-clicking the table in the Toolbox.

Now the finish machines’ process logic has to be changed in order to read the values from the Global Table.

- Change the Processor’s **Process Time** trigger option from the previous **Values By Case** logic to **By Global Table Lookup** by using the drop-down menu.
- Complete the property interface as shown in the figure to the right. The **Table** value is **ProcessTimes** - the name of the Global Table created above. It is selected from the drop-down menu options.
 - If Global Tables are created before accessing them in the model, then they are available via a drop-down menu. Otherwise, the table name must be typed in with quotation marks around the name. The name must be typed exactly as it is named in the Global Table – names are case sensitive in *FlexSim*. Therefore, it is better to select the name from the drop-down menu.
 - Use the default values for the **Row** and **Column** properties. The **Row** value is the value of the item’s (container) label Type. As defined earlier, this is specified in *FlexSim* using dot notation in the format **item.Type**. The finish times are stored in Column **1**.
 - Repeat the above steps for the other finish machine.



In summary, the process time for an item (container in this example) entering the Processor (finishing machine) is determined by a trigger that obtains the value stored in Column 1 and in the row in the Global Table ProcessTimes that corresponds to the value of the item’s label Type.

- **Reset** and **Run** the model. Verify that the containers have different finishing times by observing the different speeds that the containers have as they are conveyed across each Processor.



Items can remain static on a Processor while being processed by unchecking **Convey Items Across Processor Length** on the Processor tab. Checking or unchecking the box is for visual purposes only and does not affect the model’s behavior; i.e., the processing time is the same whether the item is conveyed across the object or not.



If you haven’t already done so, save the model. Recall, it is good practice to save often.

- Uncheck the **Convey Items Across Processor Length** box on both finish machine’s Processor tab.

5.2 Using tables to store product mix percentages

A second example of using a Global Table to store data involves the product mix of containers. Recall that in the current model, as defined in Section 5.1 of Part 1, the container type is randomly generated when it is created at the Source ContainersArrive. Recall also that it is implemented by using the *FlexSim’s* discrete-uniform distribution command `duniform(1, 3, getstream(current))`; i.e., each container’s Type is equally likely between 1 and 3.

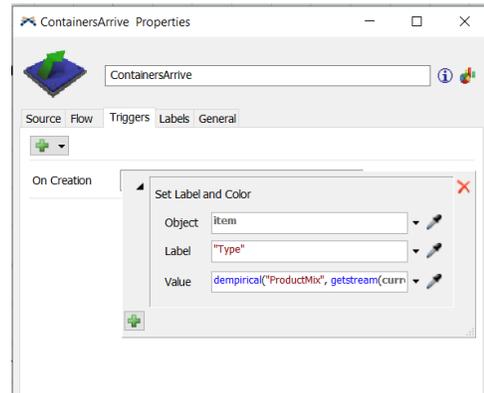
However, the product mix of container types is most likely not equal. Even if this were the case now, it would likely change in the future. A more general product mix is implemented by storing the information in a Global Table and using a different *FlexSim* command at the Source.

- In the same manner as described above, create a Global Table, name it **ProductMix**, and set the number of **Rows** to **3** and the number of **Columns** to **2**.
- As shown in the figure to the right, label the header rows and columns. Again, the headers are used to clarify what information is stored in the table. The rows correspond to the product types.
- Set the cell values as shown. Column 1 is the percentage of that type and Column 2 is the value of the type. In this case, 20% are Type = 1, 30% Type = 2, and 50% Type = 3; i.e., in the long run, half of the containers will be Type 3 (blue). *The percentages in Column 1 of the table must add to 100%.*

	Percent	Type
Type 1	20	1
Type 2	30	2
Type 3	50	3

Now the Source’s logic has to be changed in order to randomly generate a Type based on the percentages stored in a Global Table. The change is shown in the figure to the right and described below.

- Select the Source’s existing **OnCreation** trigger option **Set Label and Color**.
- In the resulting interface, no changes need to be made to the **Object** and **Label** properties.
- Replace the **Value** property, `duniform(1, 3, getstream(current))`. To do so, select the **Statistical Distributions** option from the drop-down menu option and then select **dempirical** from the list of distributions.
- Modify the **dempirical** command in the **Value** box to the following. (Change the parameter “MyTable” to “ProductMix”)
`dempirical("ProductMix", getstream(current))`



The discrete-empirical command randomly generates a number between 0 and 100 percent and uses that number to look up a Type value in the Global Table, in this example ProcessTimes.



If you haven’t already done so, save the model. Recall, it is good practice to save often.

6 DOWNTIME

In *FlexSim*, there are two means for making resources not available, i.e. incurring downtime – the Time Table tool and the MTBF/MTTR tool. Both are accessed through the Toolbox Library. A powerful feature of *FlexSim* is the management of downtimes, especially multiple types of downtimes. As such:

- Any object can be subjected to downtime.
- Any object can be subjected to multiple types of downtimes, oftentimes referred to as *competing downtimes*.
- Multiple objects can follow the same downtime process.



Begin with the basic model from the previous section, named *Primer_2-4*, and **Save As** *Primer_2-5*. Basically make a copy of the model so that it can be customized and the original model is retained.

6.1 Time Tables

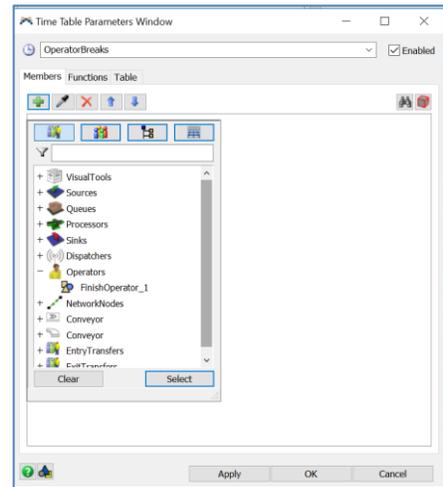
Time tables are used to model planned downtimes; i.e., ones that occur on a known and recurring basis, such as breaks, shift schedules.



[section 7-1] Establishing Time Tables

In this example, the finish operator takes a 15-minute break two hours into an 8-hour shift, a 30-minute lunch break four hours into the shift, and another 15-minute break six hours into the shift. It is implemented via the Time Table tool and its three tabs – Table, Functions, and Members. The property settings for this tool are explained below.

- In the Toolbox, press the **+** button and select 'Time Table' from the drop-down menu.
- Change the name from TimeTable1 to **OperatorBreaks**.
- The Members tab is used to assign this downtime pattern to one or more objects. Using the **+** button, select **FinishOperator_1** from the **Operators** object category and press the **Select** button. The selection is shown in the figure to the right.

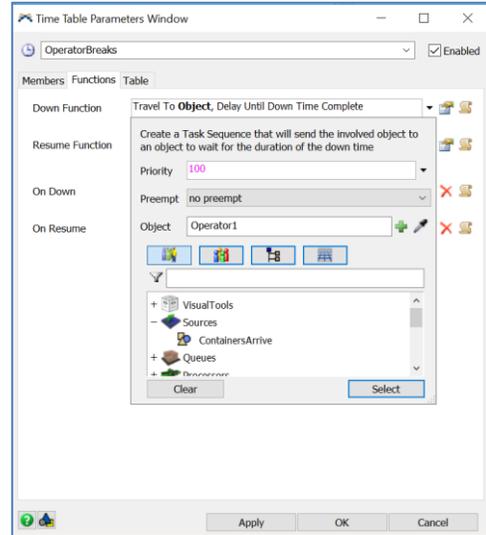


The Functions tab is used to control what happens when a resource goes down (becomes unavailable) and then resumes. By default, *FlexSim* stops the object at the appropriate time in the simulation and the object remains down for the prescribed time. In this example, the following behavior is added to the default.

The default **Down Function** stops the resource wherever it is and whatever it is doing and it remains at that location for the prescribed duration. However, in this example, the Operator takes a break at a specified location, at the Source (ContainersArrive). The prescribed delay starts when the Operator reaches the specified location.

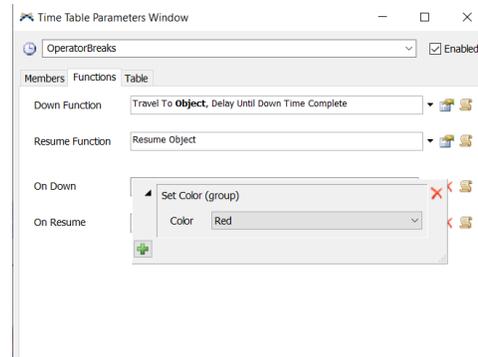
To implement this, complete the following steps and refer to the figure to the right.

- Change the **Down Function** property from **Stop Object** to **Travel To Object, Delay Until Down Time Complete** by selecting the option from the drop-down menu.
- In the resulting dialog set the **Object** property to **ContainersArrive** using the + button and selecting the object from the **Source** category. Retain the default values for the **Priority** and **Preempt** properties.

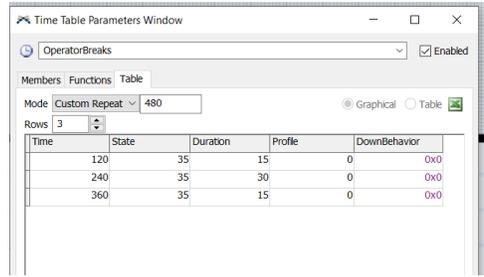


In order to provide a visual cue that the operator is not available, change the object's color during the break period. To implement this, complete the following steps and refer to the figure to the right

- Change the **On Down** option to **Set Color (Group)** and the **Color** property value to **Red** (the default). This changes the operator's color to red when on break.
- Reset the operator's color after the downtime is concluded in a similar manner. Change the **On Resume** property to **Set Color (Group)** and then the **Color** property to **Yellow**. from the drop-down list of colors. This changes the operator's color back to its default color, yellow, when the operator is available for work.

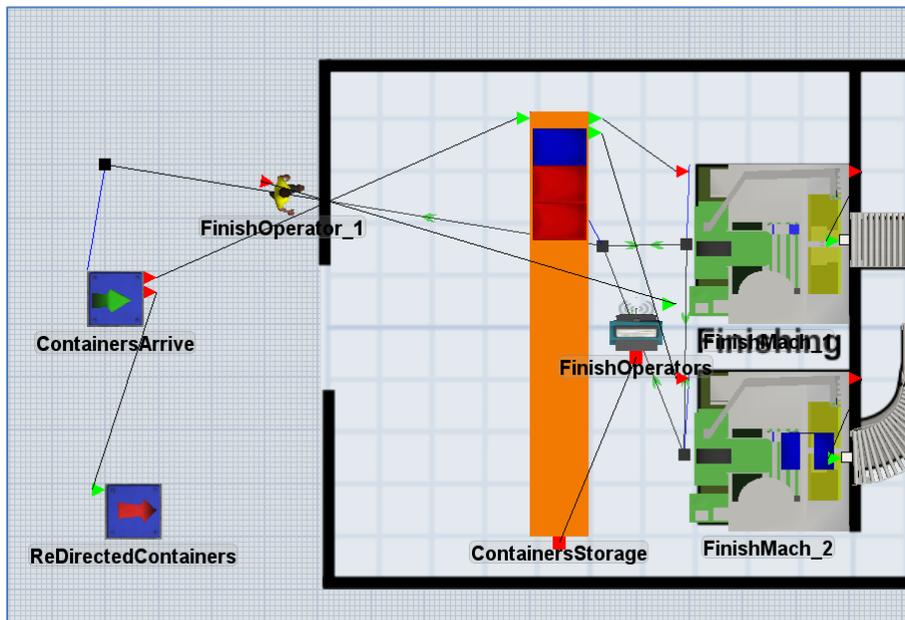


As shown in the figure to the right, the Table tab is used to set the times the resource becomes unavailable (**Time**) and for how long (**Duration**). In this case, there are three downtimes per shift and all values are in minutes, the time unit specified for the model.



- Change the **Mode** from **Date Based** to **Custom Repeat**.
- In the field next to the **Custom Repeat** value, set the repeat time to 480. The pattern entered into the table will now repeat every 480 minutes of simulated time. This corresponds to an 8-hour shift. This is handy when simulating multiple shifts so that the downtimes for each shift do not have to be explicitly included in the time table; the provided pattern just repeats for as long as the simulation runs.
- Since there are three downtimes per shift, set **Rows** to **3**.
- The first downtime is a 15-minutes break that occurs two hours into the shift; therefore, set **Time** to **120** minutes and **Duration** to **15** for the first row.
- Change the **State** parameter for each of the three rows from the default 12 to **35**. Clicking on the cell results in a list of states; select **35**. The state value 12 means “scheduled down” and state value 35 means “on break.” These are two of the 50 possible states defined in *FlexSim*. The **State** property is used for reporting the resource’s utilization. Keep the default values for the **Profile** and **DownBehavior** columns.
- The second downtime is a 30-minute lunch break that occurs four hours into the shift; therefore, set **Time** to **240** minutes and **Duration** to **30** for the second row.
- The third downtime is another 15-minutes break that occurs six hours into the shift; therefore, set **Time** to **360** minutes and **Duration** to **15** for the third row.

Since the Operator travels to the Source for breaks, and since the Operator is constrained to the network, the Source needs to be placed on the network; i.e., associate it with a new network node that is connected to the other nodes. This is shown in the figure below. While the implementation steps are the same as described in Section 3.3, they are repeated again below.



- Select a Network Node from the Object Library and drag and drop it near the Sink, ContainersArrive.
 - Name the node **nn_OperatorBreaks**.
 - A-connect the node to the previously-constructed network for the Operator, e.g. to the node near the Queue, nn_ContainerStorage.
 - A-connect the Source object to the new node.
-
- **Reset** and **Run** the model. Verify that the operator travels to the Source at the prescribed times (120, 240, 360, etc.).



If you haven't already done so, save the model. Recall, it is good practice to save often.

6.2 Reliability

Reliability refers to an object being able to perform its required functions for a specified time. It is an important contributor to overall system performance. Two key factors are used to specify reliability, *Mean Time Between Failures* (MTBF) and *Mean Time To Repair* (MTTR). The former is the up time, or time an object operates, expressed as an average or mean value; the later is the time it takes to get a down object back to its operational state, again expressed as an average or mean value.



[section 7-3 through 7-6] Reliability, Estimating MTBF and MTTR, Simulating machine failures, and Setting MTBF and MTTR in a simulation.

In this example, each finish machine is subject to two types of downtimes. Thus, the model has competing downtimes that must be managed, which *FlexSim* does very well. For example, in most cases if a resource is already down or not available, it cannot incur another type of downtime.

6.3 Constant MTTF/MTTR; MTBF based on clock time; no resource for repair

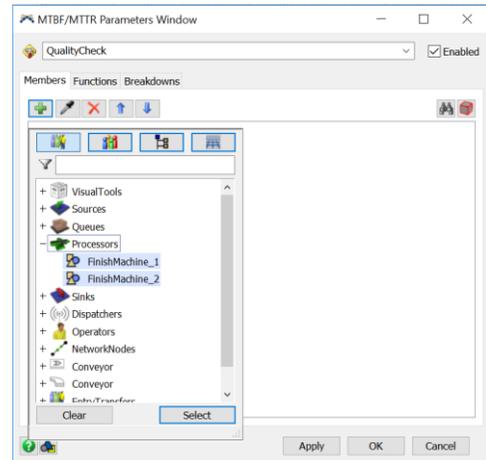
The first type of downtime on the finish machines is a quality check that occurs every 10 minutes for 15 seconds; i.e., MTBF is a constant 10 minutes and MTTR is a constant 15 seconds. The downtime occurs regardless of the machine’s current state and how much the machine has been used; i.e., MTBF depends on clock time only. No other resource is needed; i.e., the machine is just down for the duration of the self check.

- In the Toolbox, press the **+** button and select the **MTBF MTTR** tool from the drop-down menu.
- In the tool’s Parameter Window change its name from MTBFMTTR1 to **QualityCheck**.

The MTBF MTTR tool contains three tabs – Members, Functions, and Breakdowns. The settings for the quality check downtime are explained below.

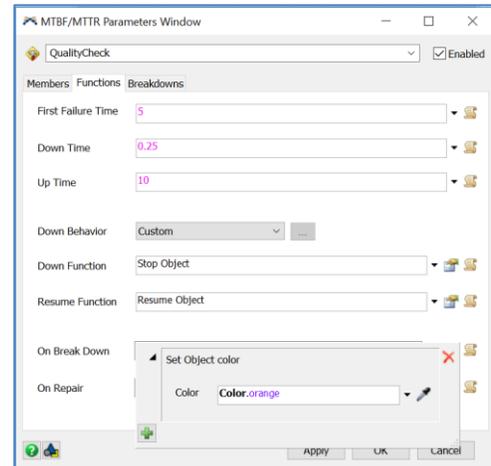
The Members tab is used to assign this downtime pattern to one or more objects; i.e., the downtime behavior defined on the Functions and Breakdowns tabs will be applied to all objects in the Members tab.

- In this case, using the **+** button, select both finish machines, **FinishMach_1** and **FinishMach_2** from the **Processors** category. This is shown in the figure to the right. Alternatively, if the category **Processors** is selected, all Processors are selected.



The Functions tab defines how often downtimes occur and their durations. In this case, all values are deterministic or constant; thus, the default probability distributions will be replaced by constants, as shown in the figure to the right.

- Change the **First Failure Time** property from the exponential probability distribution to a constant **5** minutes. This assumes the simulation starts halfway between downtimes.
- Change the **Down Time** property from the exponential probability distribution to a constant 15 seconds (**0.25** minutes).
- Change the **Up Time** property from the exponential probability distribution to a constant **10** minutes. This is the time between failures or between down states,.



The default values are used for the **Down Behavior**, **Down Function**, and **Resume Function**; i.e., when the object experiences a downtime it is stopped for the duration of the downtime and then restarted. No resource is needed during the downtime; the machine processes the downtime itself.

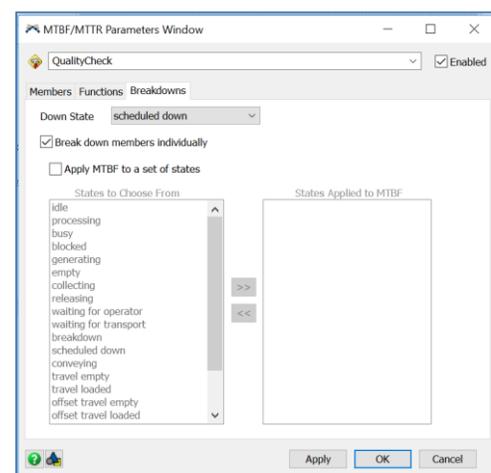
The **On Break Down** and **On Repair** triggers are used to change the color of the object when it is down. In this case, the object is orange when it is down for a quality check and back to green when not being checked.

- The **On Break Down** trigger sets the object’s color to orange indicating that it is down. As shown in the figure above, the **Set Color (individual)** option is selected from the drop-down menu and the color is set to **Color.orange**.
- Similarly, the **On Repair** trigger sets the object’s color back to green indicating it is not down. The **Set Color (individual)** option is selected from the drop-down menu and then **Color.green** is selected.

Now consider the third tab, Breakdowns.

For clarity, it is best to differentiate the downtime state for failures, which is a “breakdown,” as described in the next section, and the planned downtime for the quality checks. For the check quality case, the time between failures (downtimes) is based on the simulation clock and not on any of the object’s states; i.e., the machine will be unavailable for 15 seconds every 10 minutes, regardless if it is processing or not, in order to upload data. Therefore, the default is used in this case. This will change in the failure downtime described in the next section.

- On the Breakdowns tab, as shown in the figure to the right, change the **Down State** from the default, **breakdown**, to **scheduled down**, using the list of states on the menu. This state is more descriptive of the behavior since the quality check is a scheduled downtime and not regarded as a failure.



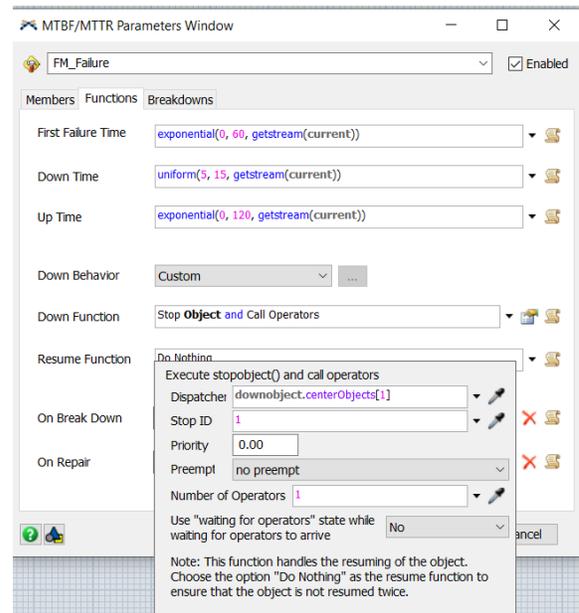
6.4 Random MTTF/MTTR; MTBF based on system states; resource for repair

The second type of downtime on the finish machines is a random machine failure. The times between failures and the times to repair are both considered random variables or probabilistic and are based on probability distributions. For this type of downtime, a resource – the finish operator – is needed to perform the repair.

- In the Toolbox, press the **+** button and again select the **MTBF MTTR** tool from the drop-down menu.
- In its parameter window, change its name to **FM_Failure**.
- On the Members tab, just as was done with the QualityCheck downtime described above, use the **+** button and select both finish machines, **FinishMach_1** and **FinishMach_2**, from the **Processors** category.

The Functions tab defines how often downtimes occur and their duration. It also defines what actions occur in the simulation whenever a downtime occurs. The Functions tab for this example is shown in the figure to the right and explained below.

In this case, all of the times are probabilistic and are specified in terms of the default probability distribution. For the times between failures (**First Failure Time** and **Up Time**), the exponential distribution is the default. The only parameter required for this distribution is its mean value. For the repair times, the default distribution is the uniform. Two parameters are required for the uniform distribution, the lowest and highest values, i.e., the shortest and longest possible repair times. All repair times are assumed equally likely between these two limits.



- Change the **First Failure Time**'s second parameter in the exponential distribution from the default value of 1000 to **60** minutes. This is the distribution's mean value and assumes the simulation started halfway between downtimes.
- Change the **Down Time**'s parameters from the default values of 50 and 100 to **5** and **15**. This assumes downtime is uniformly distributed between 5 and 15 minutes.
- The **Up Time** property is the time between failures or time between down states, It is assumed to be exponentially distributed with a mean of two hours (120 minutes). Therefore, change its second parameter, the mean, from the default value of 1000 to **120**.
- Change the **Down Function**, from the default **Stop Object** to **Stop Object and Call Operators** via the property's drop-down menu. The resulting interface is shown in the figure above. All default values are used. This selection, in addition to stopping the object during a downtime, calls a finish operator to perform the repair. The default calls the object that is connected to the object's center port; in this case, it is the Dispatcher.
- According to the note at the bottom of the interface for the **Down Function** property, the **Resume Function** value is to be set to **Do Nothing** via the drop-down menu. This is because the logic for resuming is managed in the down function.

As with the QualityCheck downtime, the **On Break Down** and **On Repair** triggers are used to change the color of the object when it is down.

- For the **On Break Down** trigger, as with the QualityCheck downtime, select **Set Color (individual)** option. However, for failures the object is colored red when it is down; therefore, choose the **Color.red** option.
- For the **On Repair** trigger, as with the QualityCheck downtime, select **Set Color (individual)** option and then set the color back to green using the **Color.green** option.

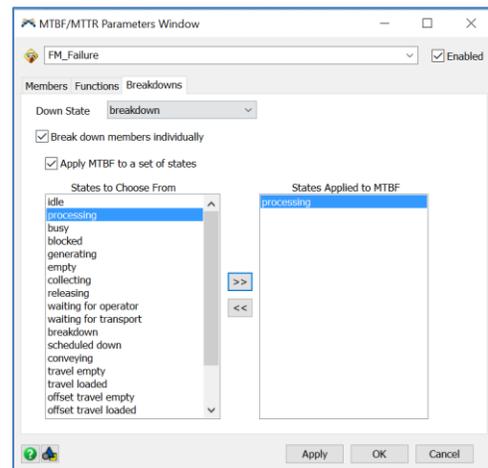


It is important to use the correct distribution since the choice can significantly impact system performance and simulation results. *FlexSim* includes many types of probability distributions. Discussion of how to select the distribution is beyond the scope of this introductory primer.

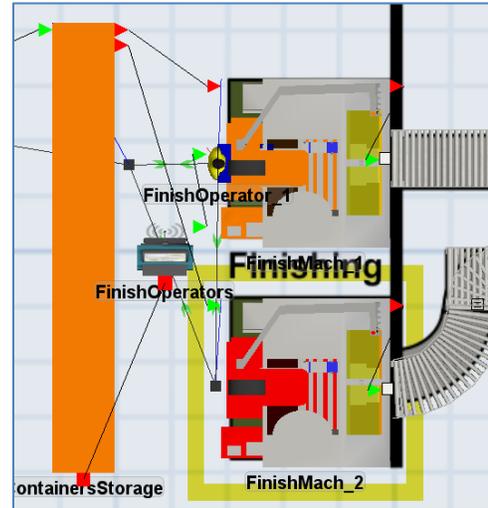
In this example the time between failures (downtimes) is based on the state of the object and not the simulation clock. The finishing machines only accrue downtime when they are running or processing. For example, if the time to the next failure on a machine is two hours and the machine operates only 50% of the time, then the next downtime will occur in four hours of simulation time. Also, a failure will occur only when the object is in one of the selected states. For example, a failure will occur only when a machine is processing and not when it is idle.

The above situation is defined on the Breakdowns tab and is shown in the figure to the right.

- Check the box **Apply MTBF to a set of states**.
- Move states from the **States to Choose From** list to **States Applied to MTBF** using the >> move button. In this case, select only the **processing** state.
- The type of state can be changed in the **Down State** drop-down menu. However, the default **breakdown** is used in this example.



The figure to the right shows the model running when both finish machines are experiencing downtime, albeit different types. FinishMach_1 is undergoing a quality check and thus is colored orange; it is in the scheduled down state. FinishMach_2 has failed and is in the breakdown state, as indicated by its red color.



If you haven't already done so, save the model. Recall, it is good practice to save often.

6.5 Charting the effect of downtime

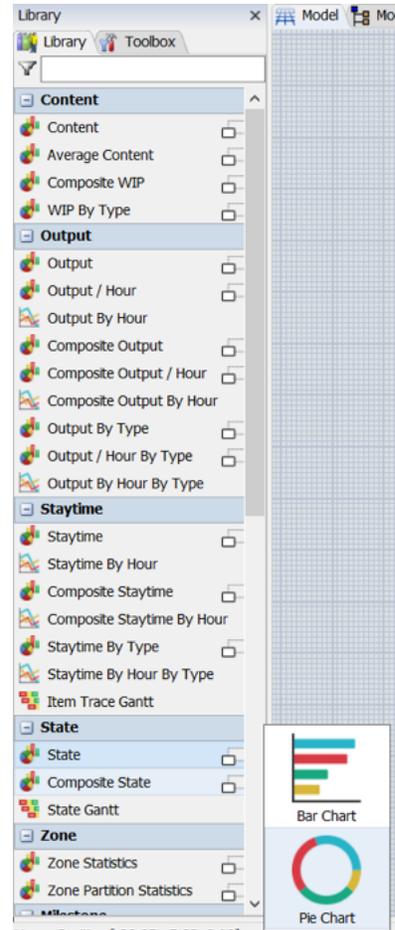
One way to visualize the effect of downtimes on system performance, is to create a dashboard of charts that show the utilization of the finish machines and the finish operator.

- From the Main Toolbar select **Dashboard** and then **Add a Dashboard**. Alternatively, click the green + on the Toolbox Library and select **Dashboard**.
- Name the new dashboard, **Utilizations**.

Pie chart for finish machine utilization

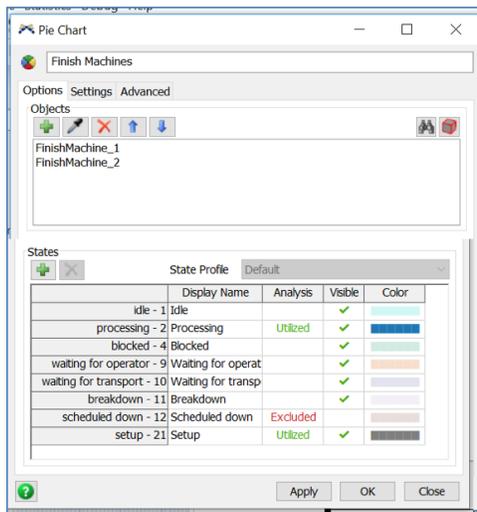
Develop a pie chart of the various states of the finish machines to assess their utilization.

- As shown in the figure to the right, select the **State** chart type in the **State** section of the Dashboard Library; then, select **Pie Chart** from the pop-up menu.
- Drag the selection to the dashboard workspace.



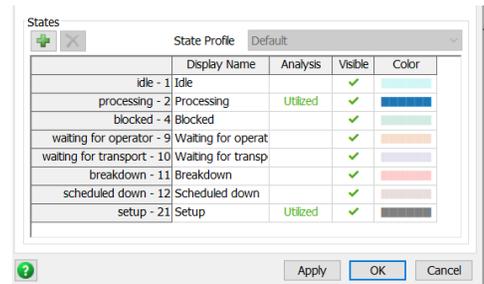
Change the property values as follows and to those shown in the figure below.

- Change the name of the chart from State Pie to **Finish Machines**.
- In the **Objects** section of the Options tab, click the + button and choose the **Select Objects** option. Then, select the category **Processors** and press **Select**. As a result, both FinishMach_1 and FinishMach_2 should appear in the **Objects** section of the interface.



The States section for the Pie Chart shown above contains the default options. However, for this example, the values will be changed to those shown in the figure to the right, both for the state **scheduled down – 12** and **breakdown – 11**. The steps for making the changes are described below.

- Click the cell in the **Analysis** column and **scheduled down – 12** row, currently displaying **Excluded**. This will change the cell to a blank, which means that the state percentage will now be considered “not utilized” on the chart.
- Since the default color of the pie slices for breakdown and scheduled down states are quite similar, they may be hard to differentiate on the chart. Therefore, click the cell in the **Color** column for the **breakdown** row, then select red from the color palette.
- Be sure the **Visible** column has a green check for all of the states. If not, click the blank space in the **Visible** column for that state. The green check means that the state percentage will be displayed on the chart.

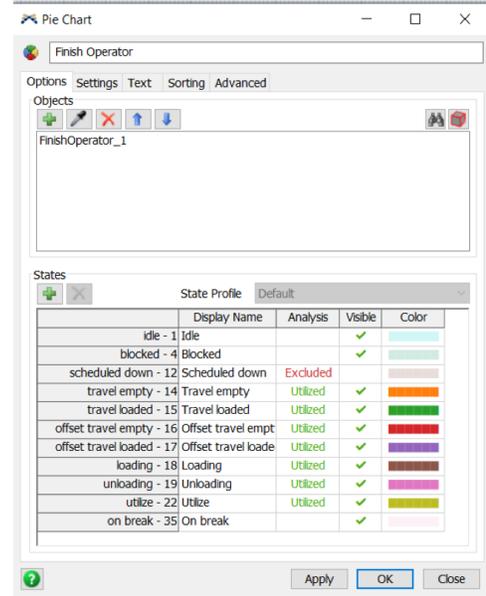


Pie chart for finish operator utilization

Develop another pie chart of the various states of the finish operator to assess its utilization. The process is very similar to the one used for the finish machines.

- Select the **State** chart type in the **State** section of the Dashboard Library; then, select **Pie Chart** from the pop-up menu. Drag the selection from the library to the dashboard workspace. Change the property values as shown in the figure to the right and described below.
- Change the name of the chart to **Finish Operator**.
- In the **Objects** section of the Options tab, click the + button and choose the **Select Objects** option. Then, select the **FinishOperator_1** object in the Operators section of the list and press **Select**.
- For this chart, all of the defaults will be used in the **States** section.

Note the list of states included in this chart is much longer than the previous utilization chart because the Operator object will be in different states than the Processor during the simulation. The percentage utilization for the Operator includes all of the travel-time tasks (empty and loaded) as well as the time spent loading and unloading items, performing setup, etc. Note the on break task is included in the states list, but is not one where the Operator is considered utilized.

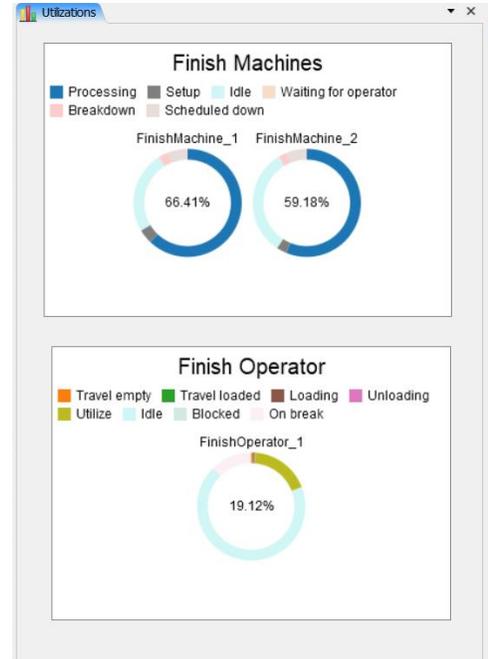


- **Reset** and **Run** the model run for 80 hours (4800 minutes). The results should be similar to that shown in the figure to the right.

Based on this simulation run, the two finish machines are each busy about 66% and 59% of the time, respectively; or, about 62% overall. These percentages include the times the Processors are in the processing and setup states (overall about 59% and 3%, respectively), as defined in the States section on the Options tab of the Finish Machines' pie chart.

The pie chart also shows the small percentage of the time that they were in the breakdown (about 2%) and scheduled down (about 6%) states. The remainder of the time, the idle state, is when the machines were ready and available, but had nothing to process; in this case, that is about 30% overall.

The utilization of the finish operator, the time the operator is busy, is about 19%. This includes traveling both loaded with containers and unloaded, loading and unloading containers, performing setup operations (part of the Utilize state on the chart), and repairing the finish machines when they breakdown (the other part of the Utilize state). In addition, the operator is in the On-break state about 13% of the time. The remaining 68% of the time the operator is in the idle state, ready and available, but has no tasks to perform.



Note that the exact percentages for each piece of the pie is displayed when the cursor hovers over that area of the chart.



If you haven't already done so, save the model. Recall, it is good practice to save often.

PART 3 – MORE 3D MODELING + ANALYSIS

The third part of the primer is the final section that deals exclusively with the basic modeling in *FlexSim*'s 3D environment. Of course, as has been stressed throughout the primer, this is only an introduction to the capabilities of *FlexSim* simulation software. Therefore, there are many more features and capabilities available in the 3D environment for effectively building and analyzing simulation models.

This part also introduces the basics of using simulation models for analysis.

In particular, the sections in this part of the primer describe:

- Combining and separating flowitems through an example that is an extension of the finishing area model from the previous sections. The extension is to a packing area where the finished containers are packed with a variety of components.
- An alternative means to control the travel path of Task Executors, in this case the finish operators, by using the A* algorithm.
- Using Lists as a means for representing complex flow rules. In this case, how the finish operator decides which container to process next.
- Setting up *FlexSim*'s Experimenter to compare multiple options based on key performance measures

Subsequent parts of the primer describe experimentation and analysis with *FlexSim* and building complex operations logic, both using ProcessFlow and coding in FlexScript.

1 COMBINING AND SEPARATING FLOWITEMS

The model from the previous section is extended to include a packing area. In this area, containers are packed with components that are produced elsewhere. At this time, it is assumed that the components arrive at the packing area on a schedule and in batches. In a subsequent section, the model is modified so that the components arrive based on current inventory levels and specified reorder points.

Also, in this example, there are only two types of components – referred to as A and B, which are purple boxed-shaped items and white cylinder-shaped items, respectively. Of course, the model could consider any number of components, but two is sufficient to introduce some key concepts. The model is developed in such a way that scaling up to a model including more components is quite easy.



As has been mentioned before, but it is a very important concept, it is good modeling practice to build smaller models first in order to test, validate, verify the approach and methods, then scale up to represent the real system.

Arriving batches wait in a buffer until the finish operator travels to the batch. The operator then unloads the individual components in the batch so that they can be used in the packing operation. Each container type is packed with its own mix of components. However, initially, each container will contain the same component mix.

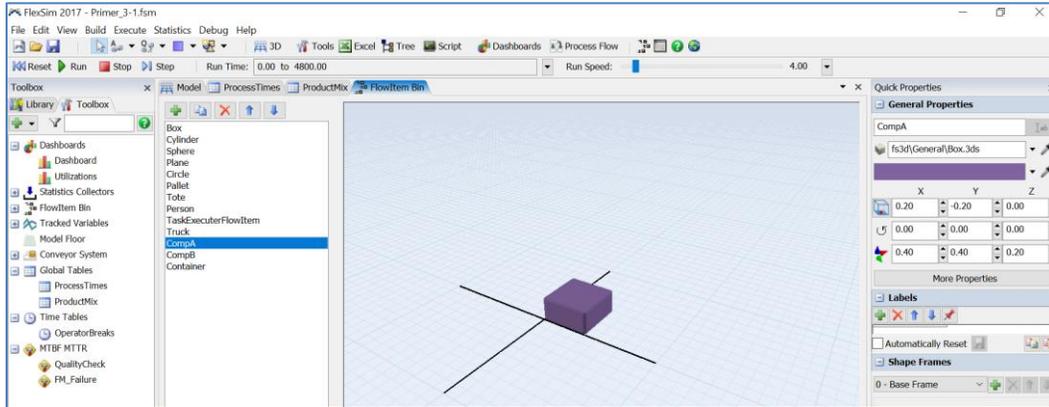


Begin with the basic model from the previous section, named Primer_2-5 and **Save As** Primer_3-1. Basically make a copy of the model so that it can be customized and the original model is retained.

1.1 Scheduled arrivals of components

The first step is to create two new flowitems to represent components A and B. This is done via the FlowItem Bin, which is accessed by the button on the Main Menu bar (to the right of the Process Flow button) or through the Toolbox.

- Create the first component as follows and as showed in the figure below.
 - Duplicate the **Box** item by highlighting the box in the list of flowitems (Box, Cylinder, Sphere, ...) and pressing the **Duplicate the selected flowitem** button (to the right of the + button). The result is a new flowitem at the bottom of the list called “Box copy.”
 - In the Quick Properties interface:
 - Change the name of the flowitem from Box copy to **CompA**.
 - Change its color from brown to purple using the color palette.
 - Change its size from the default values (x=0.61, y=0.61, z=0.3) to x=0.4, y=0.4, z=0.2.



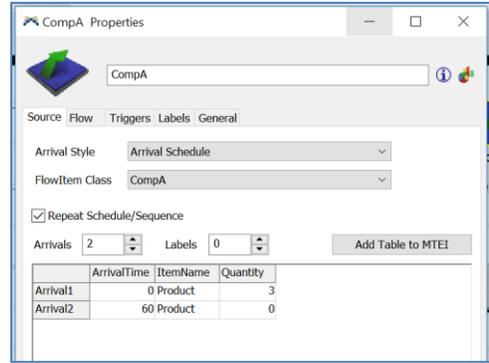
- Repeat the above process for the second component.
 - Duplicate the **Cylinder** item: highlight the cylinder in the list of flowitems (Box, Cylinder, Sphere, ...) and press the **Duplicate the selected flowitem** button (to the right of the + button). The result is a new flowitem at the bottom of the list called “Cylinder copy.”
 - In the Quick Properties interface:
 - Change the name of the flowitem from Cylinder copy to **CompB**.
 - Change its color from brown to white using the color palette.
 - Change its size from the default values ($x=0.70, y=0.70, z=0.8$) to $x=0.2, y=0.2, z=0.2$.
- Repeat the above process for the container.
 - Duplicate the **Tote** item: highlight the tote in the list of flowitems (Box, Cylinder, Sphere, ...) and press the **Duplicate the selected flowitem** button (to the right of the + button). The result is a new flowitem at the bottom of the list called “Tote copy.”
 - In the Quick Properties interface:
 - Change the name of the flowitem from Tote copy to **Container**.
 - Change its size from the default values ($x=1.0, y=0.70, z=0.55$) to $x=0.50, y=0.50, z=0.50$.
 - Leave its color as the default.

Each of the components need to be created by a Source.

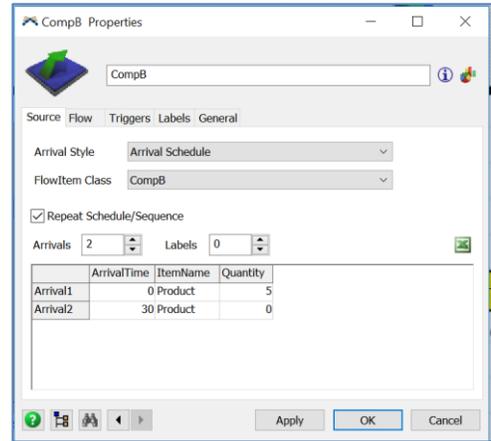
- Create a Source for each component by dragging out two Sources, placing them near the packing station, and renaming them **CompA** and **CompB**.

The components arrive in batches, three per hour for CompA and five every half hour for CompB.

- Change the properties of the CompA Source to those shown in the figure to the right and described below.
 - Change **Arrival Style** from the default **Inter-Arrival Time** to **Arrival Schedule** by using the drop-down menu.
 - Change the **Flowitem Class** property from the default **Box** to **CompA** by using the drop-down menu.
 - Check the **Repeat Schedule/Sequence** box.
 - Configure the two-row **Arrival Table** as shown in the figure (**Arrivals** is **2** and **ArrivalTime** and **Quantity** as shown). This creates three flowitems at time 0 and three more flowitems every 60 minutes. This pattern will repeat since the box in the previous step is checked.

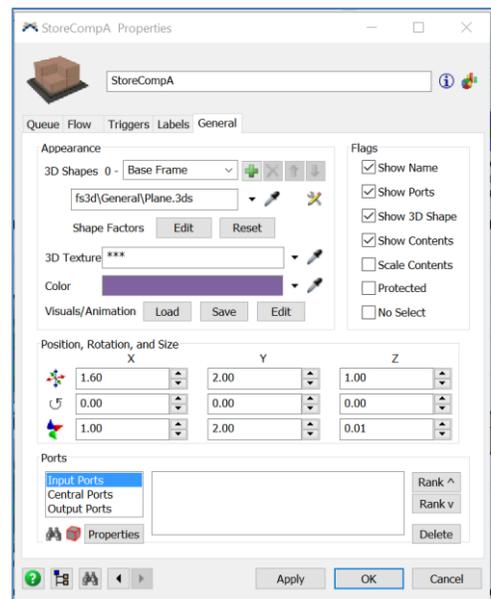


- Repeat the above steps for source CompB, as shown in the figure to the right and described below.
 - Change **Arrival Style** from the default **Inter-Arrival Time** to **Arrival Schedule** by using the drop-down menu.
 - Change the **Flowitem Class** property from the default **Box** to **CompB** by using the drop-down menu.
 - Check the **Repeat Schedule/Sequence** box.
 - Configure the two-row **Arrival Table** as shown in the figure (**Arrivals** is **2** and **ArrivalTime** and **Quantity** as shown). This creates five flowitems at time 0 and five more flowitems every 30 minutes. This pattern will repeat since the box in the previous step is checked.

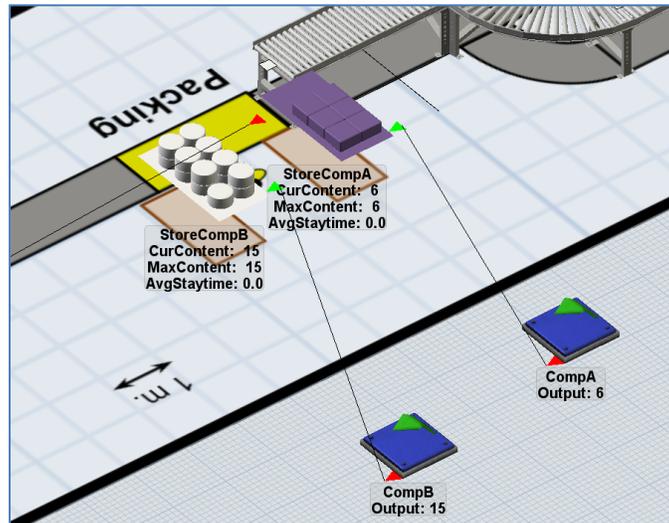


Add a Queue for each component to store the components. Since their properties are very similar, it is easier to set up one and then copy and paste it to create the other.

- Drag out a Queue object from the Library and place it near the Source.
- Modify the Queue's General tab as shown in the figure to the right and described below.
 - Name the Queue **StoreCompA**.
 - Set the **3D Shape** to a **Plane**.
 - For the Color property, select purple from the drop-down menu so that the Queue is the same color as the component it stores.
 - Set each **Size** to **x=1.0**, **y=2.0**, and **z=0.01**.
 - Set the height, **z Location** not Size, to 1.0 so that is similar to the height of the conveyor.



- Locate the Queue on the layout so that it is in the space indicated on the side of the packing station.
- Copy and Paste the StoreCompA Queue by either of the following methods:
 - Select (click on) the Queue, press the **Ctrl-C** keys, click somewhere on the modeling surface, press the **Ctrl-V** keys.
 - Right click on the Queue, select **Edit** from the menu, then select **Copy**. Right click somewhere on the modeling surface, select **Edit** from the menu, then select **Paste**.
- Modify the General tab for second Queue, which will store Component Bs, as follows.
 - Change the name of the Queue to **StoreCompB**.
 - For the **Color** property, select white from the drop-down menu so that the Queue is the same color as the component it stores.
- Locate the Queue on the layout so that it is in the space indicated on the side of the packing station.
- A-connect each of the Sources to its respective Queue.
- **Reset** and **Run** the model. Validate that the arrivals of the components occur as expected, three CompA arrive every 60 minutes and five CompB arrive every 30 minutes. Below is a screenshot of the model just after one hour showing the expected queue contents – 6 in StorCompA and 15 in StorCompA.



- While changing Sources, change the **Flowitem Class** property on the ContainersArrive Source from Tote to **Container** so that it will generate the new flowitem that was created earlier.



If you haven't already done so, save the model. Recall, it is good practice to save often.

1.2 Modeling the packing operation using the Combiner object

Now that the components have been created, they can be packed into containers. This is done through *FlexSim's* Combiner object. As the name indicates it combines flowitems.

The Combiner object can operate in three modes, which is specified by the **Combine** property on the Combiner tab.

- *Pack*, the default mode option, allows items to be combined so they can be separated later. In this case, all flowitems received through input ports 2 and above are placed *in* the item that enters port 1, referred to in general as the container item.
- *Join* combines the items permanently. The flowitem entering port 1 is the only item that exits the object once all of the components have been gathered.
- *Batch* releases all items that are collected based on the quantities specified in the Components List.

The Separator object is the “opposite” of a Combiner – it unpacks items that are in a container that were packed by a Combiner. A Separator can also “copy” items; e.g., to simulate a cutting operation where single sheet of material enters a separator, but multiple pieces of that material exit by using the “Split” option on the Separator tab. The Separator object will be discussed further in a later section.



[section 5-4] Grouping and ungrouping flowitems.

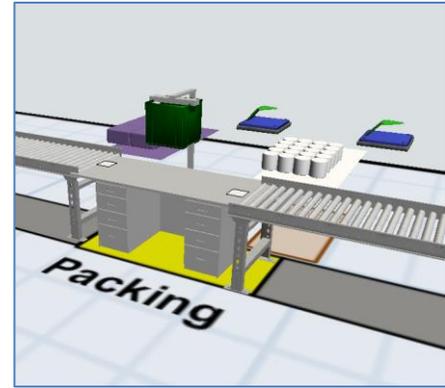


If you have already done so, **Save** the model and then **Save As** Primer_3-2. Basically make a copy of the model so that it can be customized and the original model is retained.

In order to model the packing operation in this example, complete the following steps.

- Drag out a Combiner object from the Fixed Resources section of the Object Library onto the modeling surface.
- Name the Combiner, **Packing_1**.
- Disconnect the Straight Conveyor section just prior to packing, named StCv_ToPacking, from the Sink NextProcess. (Recall, using Q-disconnect)
- A-connect the Straight Conveyor section just prior to packing, named StCv_ToPacking, to the Combiner.
- Drag out a new Straight Conveyor section and then
 - Name the conveyor section **StCv_PackToNext**
 - Set its **Horizontal Length** to **8.0**.
 - Place the conveyor on the layout after the packing station.
- A-connect the Combiner to the new conveyor section, StCv_PackToNext.
- Connect the conveyor section, StCv_PackToNext, to the Sink NextProcess.
- One the General tab:
 - Change the Combiner's **3D Shape** property to the provided *SketchUp* file named **Workstation**.
 - Change the Combiner's size to x=**2.5**, y=**1.0**, z=**2.25**.
 - Change the **Color** property from yellow to **gray**.
- Locate the Combiner in the packing location on the layout, between the two Straight Conveyor sections.

- **Reset** and **Run** the model to verify that the basic flow is correct, as shown in the figure to the right. Of course the components are not yet packed in the containers. That is described below. However, before moving to the next step, it is important to make sure all of the objects are connected properly.



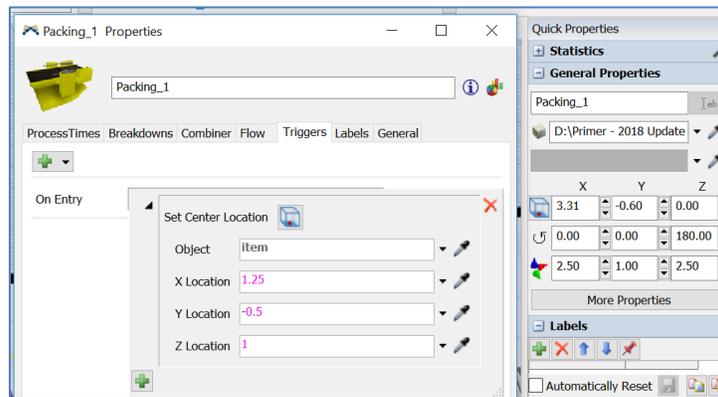
Upon closer examination, you may notice two visual problems with the packing station, as shown in the figure to the right:

1. the container appears above the packing table and
2. the packing table is facing the wrong direction.

These types of effects can occur when importing shapes. While they don't affect model performance, they look odd.

The following changes correct these visual problems. The resulting property modifications are shown in the figure below.

- To fix the flowitem's position at the packing station, use the Combiner's **OnEntry** trigger and **Set Location** menu option, which is in the **Visual** category of the drop-down menu. Set the properties as shown in the figure below. Normally the property settings require experimenting with different values until the position looks right.
- To fix the rotation of the packing station, change its **rotation** to **180** degrees about the z axis as shown in the General Properties section of the Quick Properties interface in the figure below.

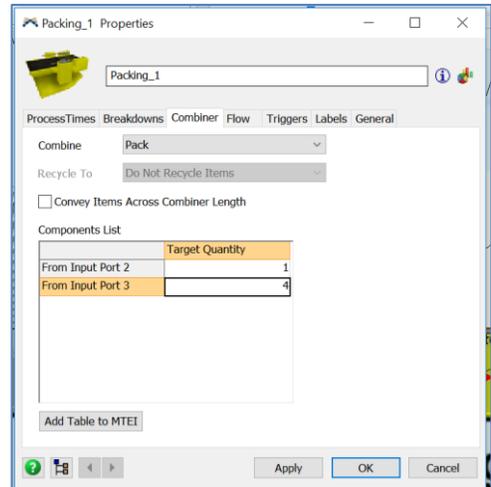


The two components that are to be packed are supplied to the Combiner via connections that are made from each component storage area (Queue) to the Combiner.

- A-connect the Queue StoreCompA to the Combiner.
- A-connect the Queue StoreCompB to the Combiner.

Each time a connection is made to a Combiner, a row is added in the **Components List** on the Combiner tab. The list is like a “recipe” for what to combine with the container. The **Target Quantity** is the number of items to collect from each port.

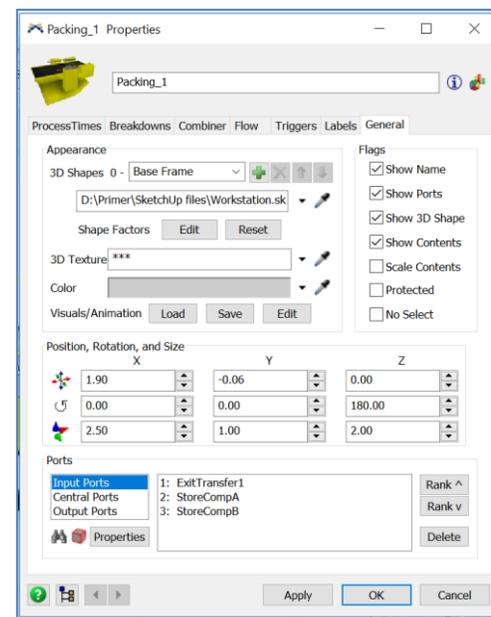
- Update the Components List as shown in the figure to the right. In this case, the Combiner is to collect **1** item from Port 2 (CompA) and **4** items from Port 3 (CompB). A single container always enters through Port 1.



One way to verify the port connections on any object is to examine the Ports section of the General tab. The figure to the right shows the input to the Combiner from ports 1, 2, and 3 are the conveyor (ExitTransfer1), StoreCompA, and StoreCompB, respectively.

Also, the ports (Input, Central, and Output) can be reordered by using the **Rank** controls and deleted with the **Delete** control, both are located to the right of the Ports list. If there are many connections to remove, it is much easier to do it from this interface than numerous Q-disconnects with the mouse.

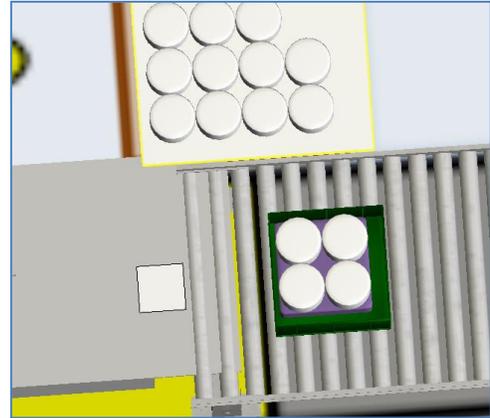
- Verify that the objects are connected properly and correct as necessary.



If you haven’t already done so, save the model. Recall, it is good practice to save often.

Once the Combiner collects all of the items it needs from all of the ports, it invokes a process time that is specified in the **Process Time** property in the ProcessTimes tab. This tab is very similar to that tab in the Processor object. The process time represents the combining time, such as packing, assembly, etc. The default time is a constant 10 time units. The process time does not start until the **Target Quantity** for all of the items specified in the **Components List** have been collected.

- **Reset** and **Run** the model to verify that each container contains one CompA (purple box) and four CompBs (white cylinders). This is shown in the figure to the right.



The current model uses what is referred to as “fixed-recipe” combining; i.e., each type of container receives the same number and type of components. However, in this example the **Components List** on the Combiner tab needs to depend on the container type. Also, the process time, i.e., the packing time, should depend on the type of container.

The recipe for each container type is specified in a Global Table that is named Packing. The table, as shown in the figure to the right, has two rows and three columns. The rows correspond to the rows in the combiner’s **Component List**. In this case, they refer to Port 2 (CompA) and Port 3 (CompB). The columns represent the Type of the container. In this case, the container has Type values 1, 2, or 3. Therefore, the cells in the table are the **Target Quantities** for the Combiner. For this example, container Type 1 receives only two CompA, Type 2 receives only four CompB, and Type 3 receives one CompA and four CompB. Note the use of row and column headers (Row 1, Row 2, Col 1, etc.). Again, they are text fields that are useful for annotating the table - they are not used by the model, but clarify what information is stored in the table.

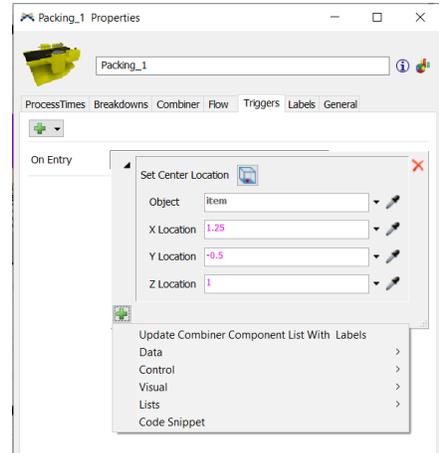
	Type 1	Type 2	Type 3
Port 2 - CompA	2	0	1
Port 3 - CompB	0	4	4

- Create a Global Table named Packing and edit the cells to correspond to the figure above.

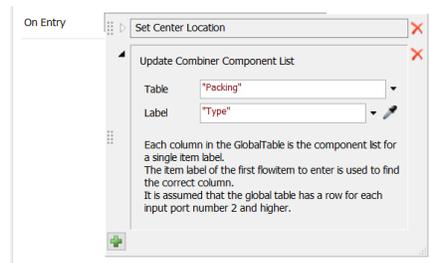
Notice that it would be very easy to expand this table to include many more product types (containers) than the three included in this example model. Also, as will be discussed later, the table could be imported from *MSExcel*.

An **OnEntry** trigger on the Combiner is used to instruct the Combiner to use the recipes specified in the Global Table. Since the Combiner’s **On Entry** trigger is used to set the flowitem’s position, as was defined earlier, the + button on the **On Entry** trigger is used to add another **On Entry** trigger.

- Add the additional On Entry trigger to the Packing Combiner as shown in the figures to the right and described below.
 - Select the previously-defined **On Entry** trigger and press the green + in the lower left part of the interface.
 - Select the menu option **Update Combiner Component List With Labels**.



- As shown in the figure to the right, for the **Table** property, select **Packing** from the drop-down list of Global Tables.
- Retain the default value for the Label property, "Type."



Based on how the components are wrapped and prepared at the packing station, it is estimated that it takes 30 seconds to pack a component A and 20 seconds to pack a component B. In addition, about two minutes per container is needed to check and prepare the container, close it, and complete the necessary forms. Therefore, it takes 2.5 minutes to pack container Type 1, 3.33 minutes to pack Type 2, and 3.83 minutes to pack Type 3. In this case, based on the current product mix, the average packing time is 3.4 minutes. This is an average rate of 17.6 containers per hour (60/3.4). Since containers arrive at an average rate of 3/hour, the packing area is not expected to be a constraint.

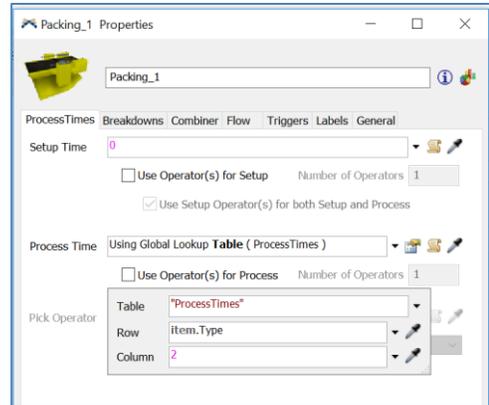
The packing times by container type that are defined above are used by the Combiner in the same manner as the finish times are used in the Processors - they are read from a Global Table. Since the format is the same, the packing times can be added as the second column in the **ProcessTimes** table.

- For the Global Table Process Times, add a column through the table's Quick Properties and update the cell values as shown in the figure to the right.

	Finish Times	Packing Time
Type 1	15	2.50
Type 2	20	3.33
Type 3	30	3.83

Now the **Process Time** property on the Combiner is changed to use the values from the table created above.

- Change the Process Time property on the Combiner's Process Times tab from the default of 10 to the **By Global Table Lookup** option on its drop-down menu, as shown in the figure to the right.
 - For the **Table** property, select **ProcessTimes** from the drop-down list of Global Tables.
 - The default **Row** property is not changed since it uses the container's label Type for the lookup.
 - Change the **Column** property from 1 to **2** so that the values in the second column of the table are used.



- **Reset** and **Run** the model to verify that the containers are packed with the correct components and that the packing times vary by container type.



If you have already done so, **Save** the model and then **Save As** Primer_3-3. Basically make a copy of the model so that it can be customized and the original model is retained.

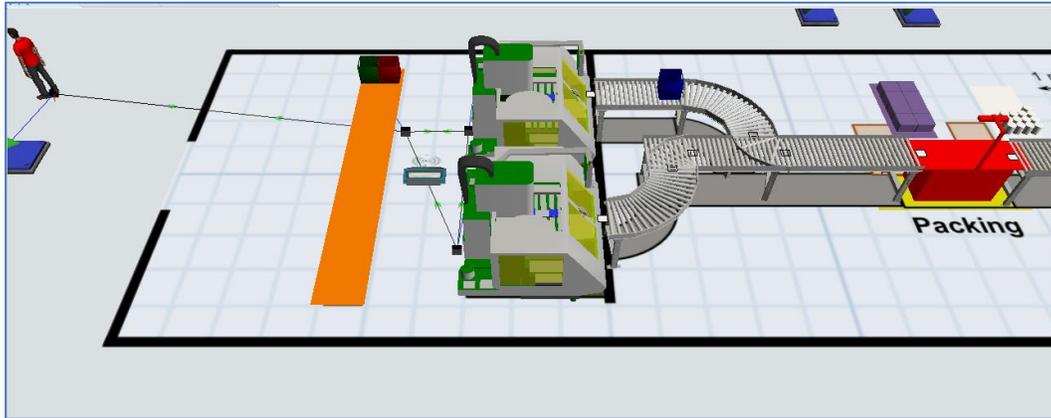
1.3 Downtime at the packing operation

In this model it is assumed that the packing operator is dedicated to a single packing station; i.e., the operator is not dynamic or a shared resource. Therefore, even though in the physical system there is a packing operator, an Operator object is not needed in the simulation. Of course, one could be added for visual effects, but it would not influence system performance.

Even though a packing operator is not explicitly used in the model, the downtime due to operator breaks should be considered. Therefore, the packing station, the Combiner object, is subjected to planned down time by using a Time Table. Since the packing area will follow the same downtime pattern as the finishing area, the Combiner can be added to the existing Time Table OperatorBreaks. However, in that table the operator travels to a location during the break, which is not possible in packing since it is considered a fixed resource for modeling purposes. Therefore, another Time Table is created, named OtherBreaks, that has all of the same properties as OperatorBreaks except the for the travel. Rather than recreate all aspects of the new table, it can be just be a copy of the previous table.

- Right click the OperatorBreaks Time Table in the Toolbox and select **Duplicate** from the drop-down menu. Modify the resulting Time Table OperatorBreaks_2 as follows:
 - Rename the Time Table, **OtherBreaks**.
 - On the Members tab, delete the **FinishOperator_1** object.
 - On the Members tab, add the **Packing_1** object (Combiner).
 - Change the **Down Function** back to the default by selecting **Stop Object** on the drop-down menu.
 - For the **On Resume** property change the **Color** property for the **Set Color (group)** option from **Yellow** to **Gray**.

As shown in the figure below, both the finishing operator and the packing area are on break since the objects are red. However, the finish machines continue to process containers and containers continue to arrive to the finish area. Of course the containers cannot be loaded onto the machines because the operator is on break. If upstream departments take a break at the same time, then the Source ContainersArrive could be added to the OtherBreaks Time Table.



Any finished containers that arrive at the packing area while the area is on break will wait on the conveyor until the packing area returns to work.



If you haven't already done so, save the model. Recall, it is good practice to save often.

1.4 Batches of components unloaded by the finish operator using Separator objects

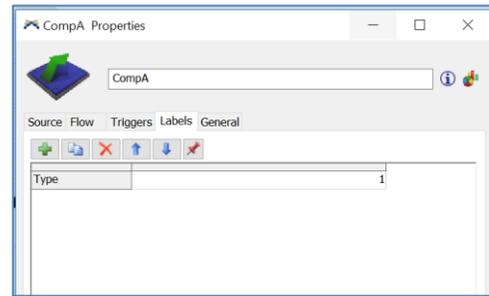
Currently, the Sources for the components each generate a specified number of components to arrive at scheduled times. An alternative approach is to generate a single arrival, considered to be a batch or order, and then “split” or duplicate that item the number of times that is specified by the batch size. The model is now changed to take this approach.

- First, set up some parameters in a Global Table named **Components**, as shown in the figure to the right. Each column corresponds to a component and each row is a parameter. For example, for CompA, every 60 minutes a batch of 3 items is delivered.

	CompA	CompB
Time between batches	60	30
Batch size	3	5

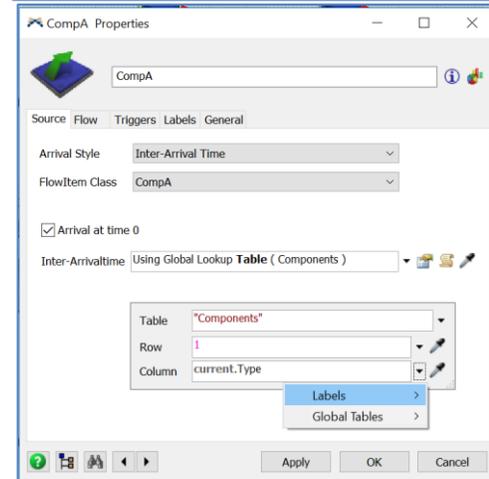
Prior to changing the arrival process, a **label** will be added to each component’s Source in order to identify what type of item the Source is generating. Recall that a label is a characteristic of an object or item that may be accessed as a model runs. Also, recall that for flowitems the system automatically creates one label named Type.

- As shown in the figure to the right, the Source label is created on the Labels tab of each Source for the components.
 - Using the + button, select **Add Number Label**.
 - Change **labelName** to **Type**.
 - Set the value of Type to be **1** for Source CompA and **2** for CompB.



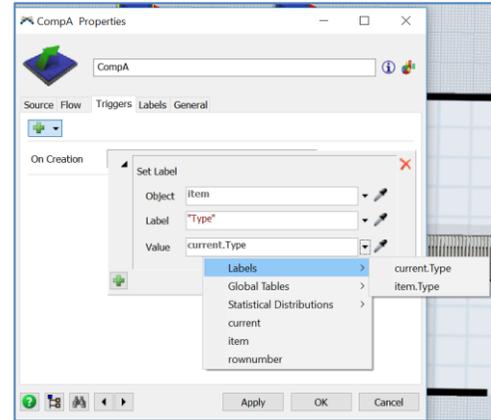
Each component’s Source is modified so that it reads the time between batches from the Global Table Components.

- Modify CompA’s Source as shown in the figure to the right and defined below. Also modify CompB’s Source in the same manner.
 - The **Arrival Style** is changed back to **Inter-Arrival Time**.
 - The box **Arrival at time 0** is checked. This means that a batch is delivered when the simulation begins and all subsequent batches are delivered based the time between batches.
 - In the drop-down menu, the **Inter-ArrivalTime** property is changed to **By Global Table Lookup**.
 - For the **Table** property, the **Components** table is selected from the list of Global Tables that are in the model.
 - **Row** is set to **1** since the first row in the Components table is for the time between batches.
 - **Column** is changed to **current.Type** by selecting the **Labels** option in the drop-down menu and then **current.Type** on the subsequent menu. This means the column number that is used to obtain a value from the table is based on the label value named Type on the current object. For Source CompA, the lookup value is 1; whereas, the lookup value is 2 for CompB.



The items created by the Source will have the same Type value as the Source. This is accomplished as follows and as shown in the figure. Use this process for both Sources.

- Select the **On Creation** trigger on the Source’s Trigger tab, then the **Data** category, and then **Set Label**.
- The **Object** property is the default value, **item**. The label is being set on the created flowitem.
- The value of the **Label** property is the default **Type**.
- The **Value** property is set by selecting **Labels** from the drop-down men, then selecting **current.Type**.



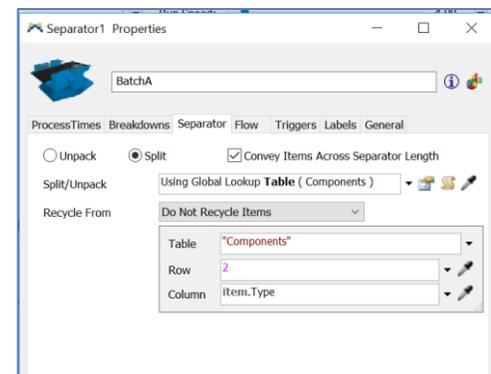
It is important to note that when referencing the label named Type on the *object*, **current.Type** is used; when referencing the label named Type on the *flowitem*, **item.Type** is used.

Notice that by using Global Tables it makes extending a model much easier and faster. Additional component Sources could be added by copying and pasting CompA or CompB and just changing their name and the value of the object’s label Type.

The component Sources now generate one flowitem at a frequency determined by the time between batches. The Separator object is used to convert the single flowitem that represents a batch into the number of items in the batch.

The model needs two Separators, one for each component. One Separator will be created for CompA, as described below, and then it will be copied and pasted and the copied Separator will be modified for CompB.

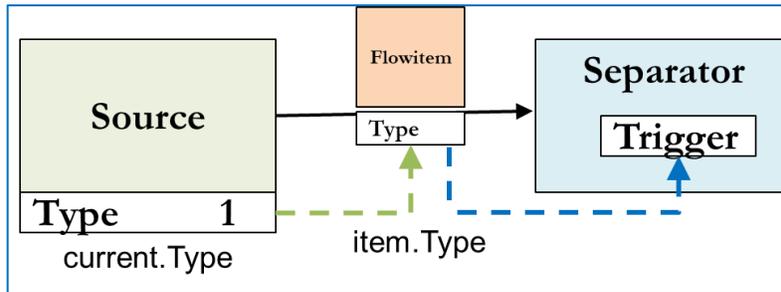
- Drag out a Separator object from the Fixed Resources section of the object and place it on the modeling surface near the packing area.
- Name the Separator **BatchA**.
- On the General tab,
 - Set its size to **x=1, y=1, z=0.05**.
 - Set its **z location** to **1.0** so that it is at the same height as the Queue. This makes it like a worktable.
 - Set the **Color** property to purple - the color of the flowitem it is processing.
- On the Separator tab,
 - Uncheck the property **Convey Items Across Separator Length**.
 - Click the **Split** button; note the default is **Unpack**.
 - As shown in the figure to the right, change the property **Split/UnPack** from the default **Entire Contents** to **By Global Table Lookup**.
 - For the **Table** property, select **Components** from the drop-down list.
 - Set the **Row** property to **2**. This is the row in the table that contains the values for batch size.



- Set the **Column** property to **item.Type** by selecting **Labels**, then **item.Type**. The column in the table is the value of the label stored in the entering item's label named Type.
- Copy and Paste Separator BatchA by: selecting the Separator, pressing Ctrl-C, clicking on the modeling surface, and pressing Ctrl-V.
- Change the following on the pasted Separator. All other properties created above pertain to both objects.
 - Rename the Separator **BatchB**.
 - Change the color to white.
- Integrate the Separators into the model.
 - Delete the connection between each component Source (CompA and CompB) and the Queue (StoreCompA and StoreCompB).
 - Connect each Source to their associated Separator, e.g., connect CompA to BatchA.
 - Connect each Separator to their associated Queue, e.g., connect BatchA to StoreCompA.
 - Position the Separators on the layout adjacent to their associated Queues, e.g., BatchA near StoreCompA.

Note that the time to split batch is 10 minutes since the default Process Time property is used in the Separator. That is okay for now since it will be changed shortly.

The use of labels in this example is summarized in the figure below. Each Source has a label denoting the type of component it generates; the name of the label is Type. Whenever a flowitem is created by the Source, the value of its label named Type is assigned to a label on the flowitem, in this case it is also named Type. Basically the operation `item.Type = current.Type` is performed. The Separator then uses the flowitem's label value to determine the number of items that are in a batch.



The time for a finish operator to split a batch is estimated to be one minute plus 3 seconds per item. The number of items in a batch is stored by component type in the second row of the Components table. Since the fixed time per batch and the time per item might change, or vary by product type, it is a good idea to put these in a table as well. Therefore:

- Add two rows to the Components table for these parameters, as shown in the figure to the right. While the fixed and variable times to unpack a batch may be the same for all component types, adding these parameters to the Components table provides the flexibility to vary the values by type. For example, some items might be more difficult to handle based on their size or may require special handling.

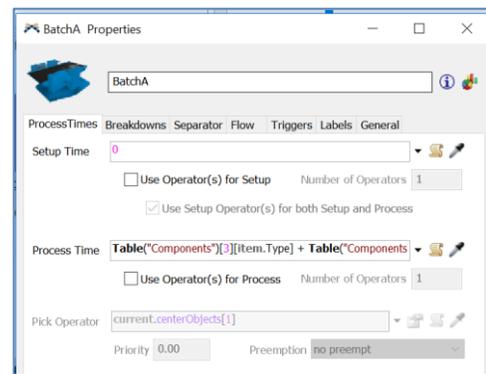
	CompA	CompB
Time between batches	60	30
Batch size	3	5
Time to unload a batch - fixed (minutes)	1	1
Time to unload a batch, per item (minutes)	0.05	0.05

The **Process Time** at each Separator, to unpack a batch, is expressed as the following.

```
Table("Components")[3][ item.Type]
+ Table("Components")[4][ item.Type] *Table("Components")[2][item.Type]
```

- Enter this expression carefully and exactly as it appears because this actually a programming statement. It all must be entered on one line; the statement above is broken up across multiple lines for readability. Implementation of the expression is shown in the figure to the right.

As the expression is typed in the text box, *FlexSim*'s context-sensitive help makes it a bit easier by suggesting command names and table names and when a leading bracket or parenthesis is entered, the ending symbol is automatically provided by *FlexSim*.



Be careful on entering (or [- one is a parenthesis and the other is a bracket and they have different meanings in computer programming.

Also, copying and pasting can make entering the statement a bit easier; e.g. the first term can be copied and pasted and then edited, as opposed to typing in each character.

Once typed into one Separator, the expression can then be copied and pasted into the other Separator.

The Process Time expression that is defined above is interpreted as follows. In general, the fixed time per batch is added to the product of the time per item and the number of items in the batch. In this case, the fixed process time per batch is the value obtained from the cell in the Global Table Components that is located in row 3 and in the column number that corresponds to the type of the entering flowitem. Similarly, the process time per item and the number of items in a batch are the values obtained from the cells in the Global Table Components that are located in rows 4 and 5 respectively and in the column number that corresponds to the type of the entering flowitem.

While this operation may seem quite complicated, it demonstrates the power and flexibility of *FlexSim*. This is actually a coding or programming statement, albeit a simple one. A constant time could have been used, and that might be a good strategy early in the model-building process. However, one goal of this primer is to introduce, or at least illustrate, some of the power of *FlexSim*.

It was mentioned earlier in the primer that one way to customize objects and logic in *FlexSim* is via scripting in a C++ type of computer programming language called Flexscript. The expression above is a Flexscript statement. Other Flexscript statements and commands have been used in the primer, but they have not been

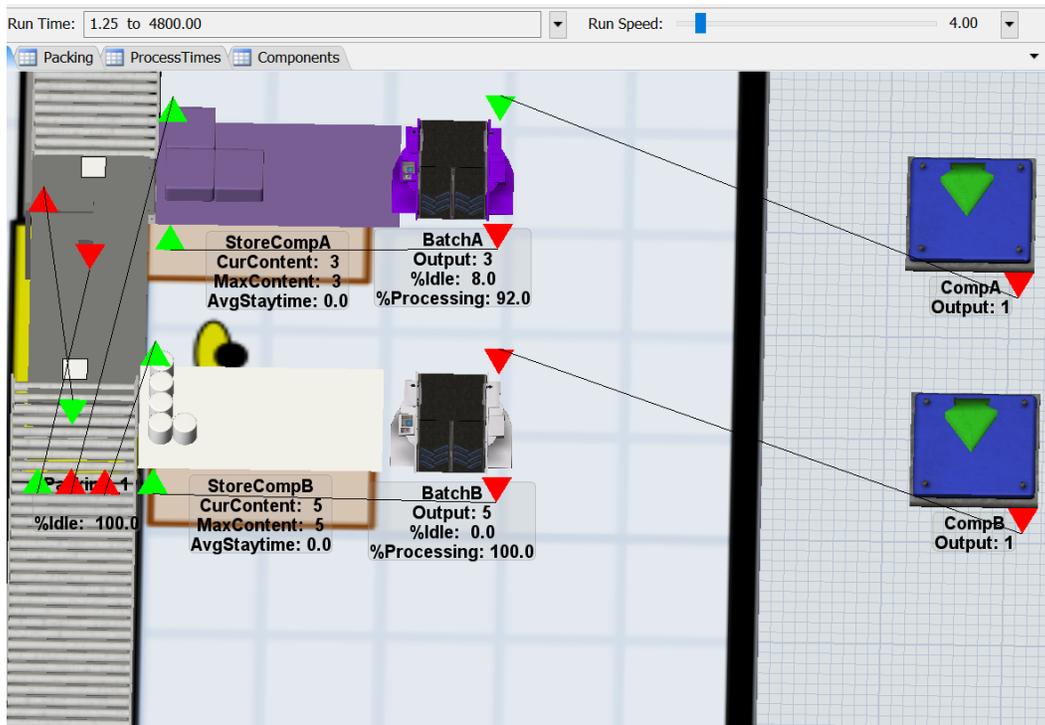
identified as such. For example, the statistical distributions - such as **triangular(min, max, ml, getstream(current))**, **duniform(min, max, getstream(current))**, **dempirical("tableName")** - are all examples of Flexscript commands that are programmed to obtain random samples from the designated distribution. Also, **item.Type** is Flexscript's reference to the value of the label named Type on the flowitem that is being processed. More information on coding in Flexscript is available in the *FlexSim User Manual* and the following:

 Another primer by this author, entitled *A Primer on Coding in FlexSim 2017*.



[Section 12-3] Scripting in *FlexSim*

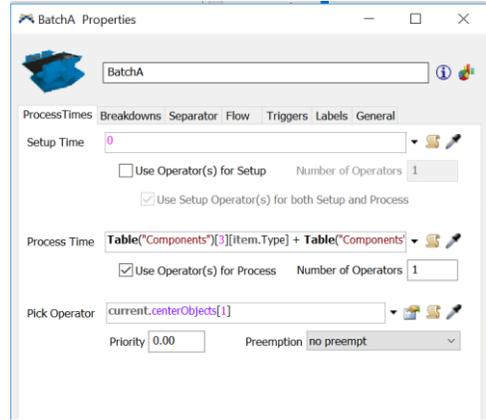
- **Reset** and **Run** the model and verify that batches arrive when scheduled and the Separators split each batch into the specified batch size. In the figure below, the batch of three CompA entered the queue at simulation time 1.15 minutes ($1+3*0.05$); similarly, the batch of five CompB entered its queue at time 1.25 ($1+5*0.05$). Thus, the Process Time expressions in the Separators are working correctly.



If you haven't already done so, save the model. Recall, it is good practice to save often.

The finish operator must perform the operation at the Separator, i.e., unload batches.

- As shown in the figure to the right, check the box **Use Operator(s) for Process** on the ProcessTimes tab of each Separator, BatchA and BatchB. This sends a request for an Operator to travel to the object and be delayed by the value of the **Process Time** property - in this case the time to unload a batch of components.

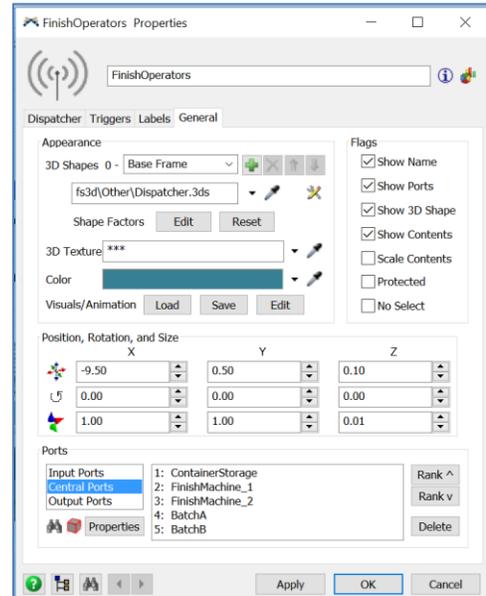


Note that the **Pick Operator** property value is grayed out until the **Use Operator(s) for Process** box is checked. In this case, the default **Pick Operator** property is used and the result is that the request to perform the task is sent to the Dispatcher, then the Dispatcher sends the request to an attached Operator. Of course, in this case there is just one finish operator.

Once the process time is complete the finish operator is free to perform another task; it becomes idle and waits at the batch area until it receives another task. This is referred to as the basic **utilize** task sequence and its tasks differ from the basic **transport** tasks described earlier. This task sequence is composed of just two tasks: (1) travel to the object and (2) be utilized for the specified time. As mentioned in a previous section, the task sequences can be customized, but that is a more advanced feature of *FlexSim*.

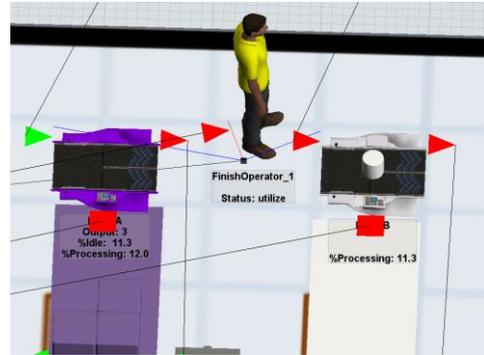
Of course each Separator must be connected to the Dispatcher.

- Use a center- (or S-) connection to connect each Separator and the Dispatcher. Notice, as shown in the figure to the right, the Dispatcher is now connected to five objects; i.e., it receives tasks from each of these objects.



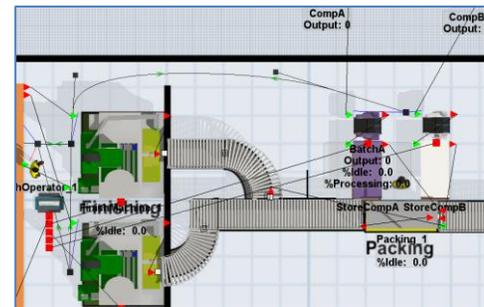
In a previous part of the primer the Operator was placed on a network in order to restrict its travel paths. Therefore, for the operator to do the unpacking of the component batches, the network needs to be extended to include the packing area. Otherwise, an error will occur since the Operator object will not know how to reach the area to unpack the batches by staying on the network.

As shown in the figure to the right, only one Network Node is added to the Operator's path network. This is because in the next section an alternative means for controlling the travel paths of Task Executors is discussed. Note in the figure that both Separators use the same Network node and the Operator performs the unpacking operation at the Node between the two Separators since the Operator has been instructed not to travel offsets.



To enable the Operator to travel to the packing area:

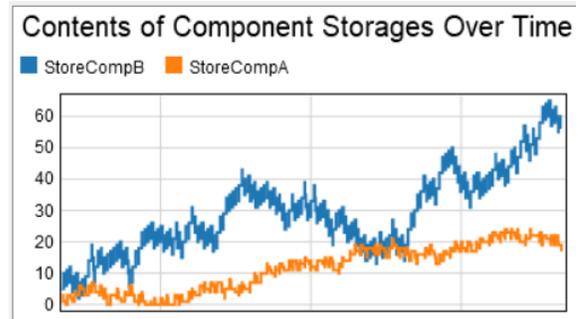
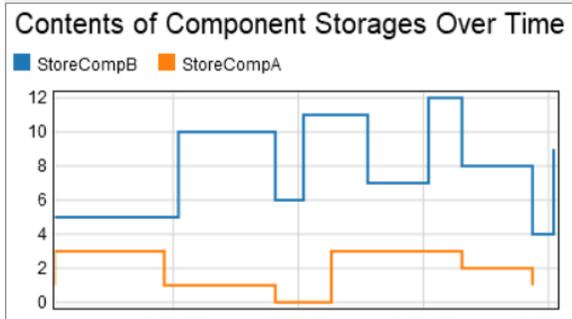
- Drag and drop a Network Node from the Object Library and place it between the two Separators.
- Name the node, **nn_UnBatching**.
- A-connect the node to nn_ContainersArrive.
- A-connect the node to both Separators, BatchA and Batch B.
- Optionally, as shown in the figure to the right, reshape the path so that the Operator does not travel the straight-line route between the finishing and packing areas and travel through FinishMachine_1 and other objects. To do so:
 - Right-click on one of the green arrows on the just-completed path and select **Curved** from the menu.
 - As a result of the above selection, two black squares, referred to as “handles,” appear on the path between the green arrows. Use the handles to define a smooth path that avoids any obstacles, as shown in the figure to the right. You might need to initially place the Node away from the area so that you can access the two handles since they may be hidden by another object.



It would be informative for analysis to add a Dashboard that displays the contents of the component queues over time.

- Create a new Dashboard by using the **Add Dashboard** drop-down menu and name it **Component Storage**.
- Select the **Content** chart template from the Content section of the Dashboard Library and then select Line Chart. Drag the template to the new Dashboard.
- Name the chart, **Contents of Component Storage over Time**.
- Using the + button in the **Objects** section of the Options tab, add two objects, both component Queues, StoreCompA and StoreCompB.
- On the Settings tab, change the **Time Access Mode** to **Show Duration** and set the time unit to **Hours**.

- **Reset** and **Run** the model. The contents of the component storages should be similar to the plots shown in the figure below. The plot on the left is after about two hours (120 minutes); the plot on the right is at the end of the simulation run of 4,800 minutes (80 hours).



Based on the single 80-hour simulation run, shown to the right above, it appears that the trend in the number of components waiting to be packed continues to grow over time. Therefore, the time between batches and/or the batch size should be adjusted. The simulation model can be used to decide the best settings. Also, in the model there is basically no limit on the number of components that can be stored prior to packing. The physical space limitations need to be considered.



If you haven't already done so, save the model. Recall, it is good practice to save often.

2 CONTROLLING TASK EXECUTER TRAVEL PATHS USING A*

By default, Task Executors, such as Operators and Transporters, travel in straight lines between points in a model. While this is efficient, it can be unrealistic – in reality operators cannot go through machines, conveyors, etc. The straight-line path not only doesn't look right, it can affect estimated system performance – it takes longer to follow an object-avoidance route compared to the straight-line distance between objects.

Up to this point in the primer models, the travel paths of the Task Executors, i.e. the finish operator, is controlled through a network of connected Network Nodes. Each Node is considered an object and is placed on the modeling surface at key locations to establish the paths, which can have either straight or curved segments. In essence, this approach tells the Task Executor where to go. The alternative approach that is considered here, referred to as A* or AStar, tells the Task Executor where *not* to go and let's the TE decide the path that avoids the specified obstacles.

A* (pronounced A-star) is a popular search algorithm that is commonly used in computer science and mathematics. It uses heuristics to find the shortest path between points considering barriers or no-travel zones. As with all other topics in the primer, only the basics are introduced here and mostly the default settings are used.

Utilizing the A* algorithm in *FlexSim* is quite easy. The objects associated with the A* algorithm are in the section of the Object Library named A* Navigation, just below the Visual section.

2.1 The basics of the A* algorithm using a simple study model

Before implementing A* in the current primer model, use a simple study model to understand the basics.

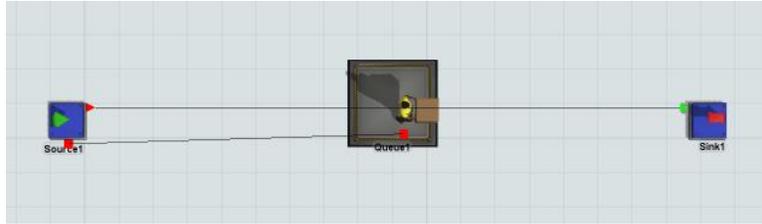
- Create a new model using all default settings.



It is good modeling practice to try new concepts outside of the main model being constructed. These simple models, often referred to as *study* models, are useful in learning new concepts or understanding how *FlexSim* features behave. They focus on a particular problem or a portion of complex logic.

As shown in the figure below, create a very simple study model of an operator transporting items between a Source and a Sink. Use all default values. Add a Queue that is not connected to any other objects – it is used only to represent a storage area that the operator should avoid.

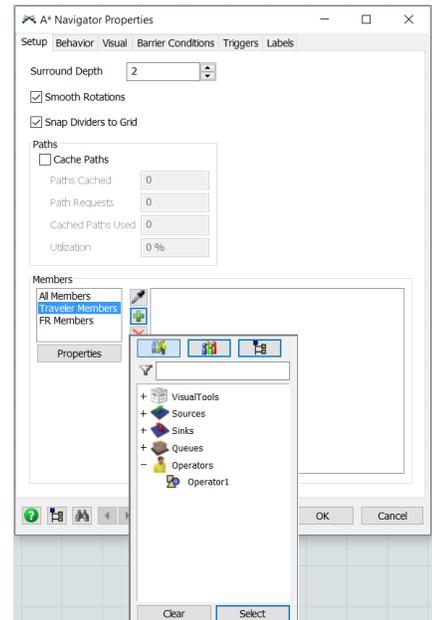
- Drag out a Source, Sink, Operator and Queue from the Object Library.
- A-connect the Source to the Sink.
- Reset the view so that it is in planar view.
- Place the Queue on the straight-line path between the Source and Sink.
- Center-connect the Operator to the Source.
- On the Flow tab of the Source, check the property **Use Transport**.



- Save the model as A-StarStudy.
- **Reset** and **Run** the model. As expected the Operator’s path should be through the Queue.

Now, use the A* tools to avoid the queue and other barriers. The tools are located in the A* Navigation section of the Object Library, right after the Visuals section.

- Drag out the A*Navigator object and place it anywhere on the modeling surface.
- Double click the A*Navigator object and note it has Setup, Behavior, Visual, and Barrier Conditions tabs in addition to the common FlexSim object tabs Triggers and Labels.
- In the **Members** section of the Setup tab, use the + button to add objects to the **Traveler Members** list. In this case, select the **Operator1** in the Operators section, as shown in the figure to the right. Objects selected here will have the A* algorithm applied to them.



- In a similar manner, add the **FR Members**, where FR means Fixed Resource. The interface is just below the **Traveler Members**, again on the Setup tab.
 - Select the **Queue** since this is the object the Operator needs to avoid.
 - By selecting an object category, e.g. Queues, all objects in that category are selected. Of course, in this case, there is only one object in an object category.

- **Reset** and **Run** the model. Note, as shown in the figure to the right:
 - A blue box provides an outline of the objects that are considered by A*. The box can be hidden by unchecking **Show Bounds** on the Visual tab of the A*Navigator object.
 - The Operator now avoids the Queue. By using A*, the Operator takes the shortest path between the Source and Sink that avoids the Queue.
 - The size and location of the blue box may need to be adjusted since the operator will not travel beyond its boundaries. It can be repositioned by selecting anywhere on the blue border and dragging the box to the desired location. It can be resized by selecting and dragging one of the red arrows that appear on the border when the object is selected.

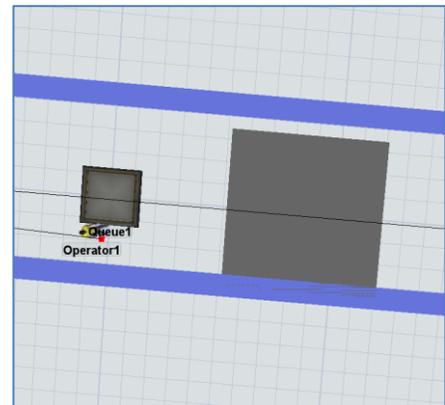


If you haven't already done so, save the model. Recall, it is good practice to save often.

TEs can avoid objects that are not basic Fixed Resource objects. Two examples are illustrated here. A *Barrier* object is a rectangular shape that will cause a TE to move around the object; it can be located anywhere in the 3-dimensional space. A *Divider* object is a set of line-segments in 3-dimensional space that creates either a one-way or two-barrier. The line segments are analogous to walls.

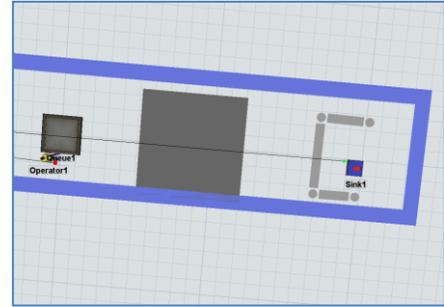
- As shown in the figure to the right, drag out a Barrier object from the Object Library and place it on the modeling surface. Its size and location can be changed by double clicking the object or via Quick Properties.

The barrier could be used to represent a storage area instead of using the queue object.



As shown in the figure to the right, a Divider object is added around the sink:

- Select the Divider object from the A* section of the Object Library and then click on the layout where a divider should start. A gray circle marks this spot.
- Use the mouse to draw the Divider, setting its direction and length. Click at an end point of the Divider and another circle marks this point. Continue this process until the desired shape is drawn on the layout.
- Click the Esc key to stop drawing the Divider.
- The Divider can be repositioned and reshaped by:
 - Selecting one of the line segments and moving the entire divider object, retaining its shape and size.
 - Selecting and moving a circle segment to change the dividers shape and size. Moving a circle changes the length of the line segment and/or the angle between the segments.

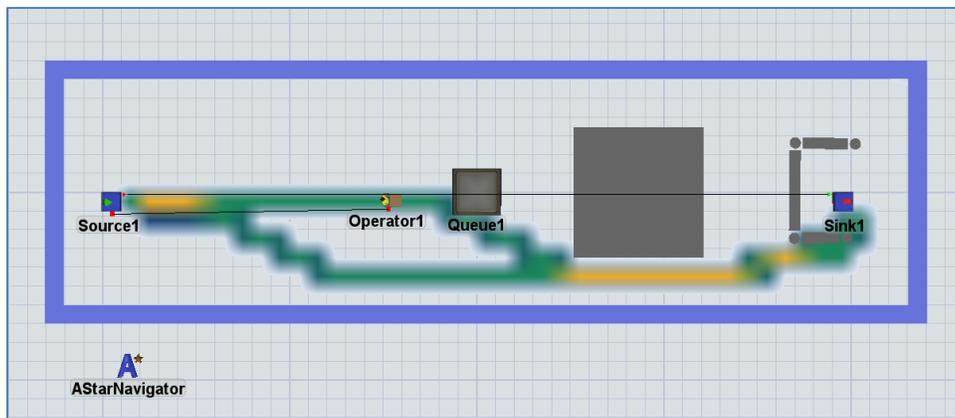


- **Reset** and **Run** the model and observe the behavior. The Operator should identify and follow a path between the Source and Sink that avoids the Barrier, Divider, and Queue.

A heat map can be used to see the path the operator travels and the frequency the path is used.

- Check the **Show Heat Map** box on the Visual tab. Also, just below this property, check the **Transparent Base Color** box.

The results are shown in the figure below. Initially, notice that the operator chooses a slightly different path when going to the Sink from the Source and to the Source from the Sink. The “warmer” zone, which indicates a higher travel frequency on that path, is in yellow, as opposed to green.



If you haven't already done so, save the model. Recall, it is good practice to save often.

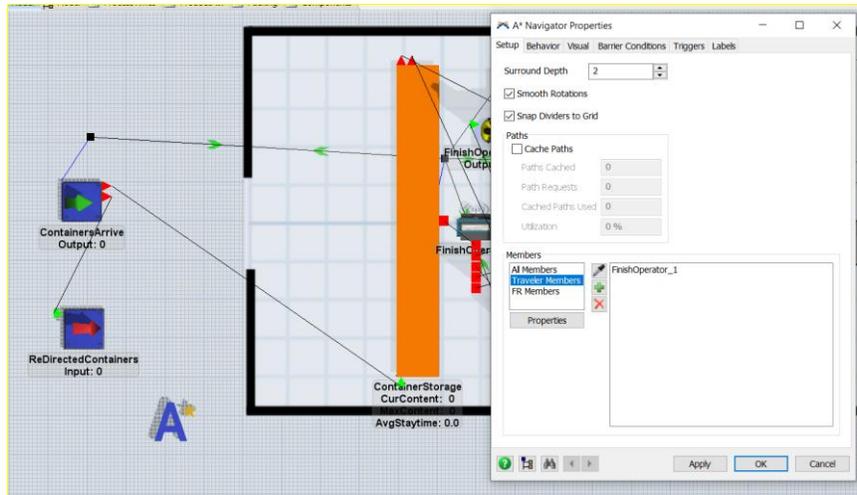
2.2 Implementing the A* algorithm in the primer model

Now that the basics of using the A* algorithm is introduced through a simple study model, it is added to the primer model.



Begin with the basic model from the previous section, named Primer_3-3, and **Save As** Primer_3-4. Basically make a copy of the model so that it can be customized and the original model is retained.

- Drag out the A*Navigator object anywhere on the modeling surface outside of the layout, as shown in the figure below.
- On the Setup tab and in its **Members** section, use the + button to add objects to the **Traveler Members** list. In this case, select the **FinishOperator_1** in the Operators section, as shown in the figure below. Objects selected here will have the A* algorithm applied to them.



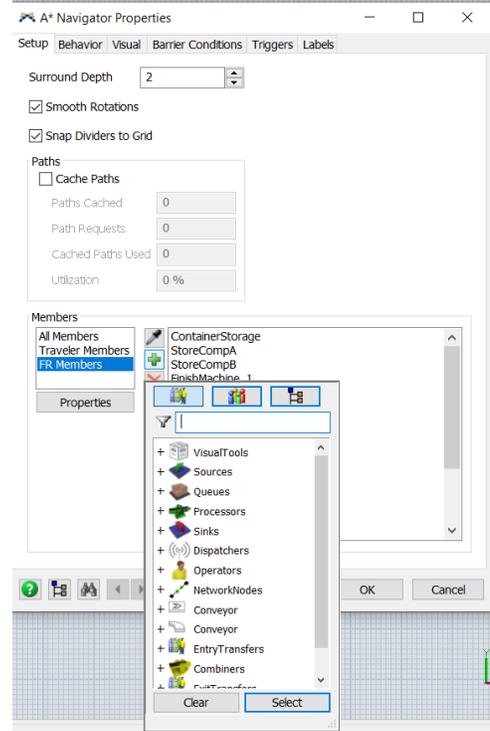
In a similar manner, add the **FR Members**.

As shown in the figure to the right, the interface is just below the **Traveler Members**, again on the Setup tab.

- Select all **Queues, Processors, Conveyors** (Straight and Curved), **Entry Transfers, Combiners, Exit Transfers,** and **Separators**. These are the objects that are used by A* and the objects that the Operator needs to avoid.
- By selecting an object category, e.g. Queues, all objects in that category are selected. In this case, by selecting the Queues category, the objects ContainerStorage, StoreCompA, and StoreCompB are added to the FR Members list.

Multiple object categories can be selected at a time by holding down the Shift-key when selecting categories.

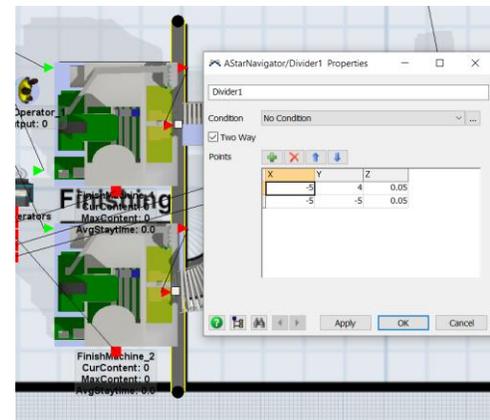
All of the object types in the model are selected except Visual Tools, Sources, Sinks, Dispatchers, Operators, and Network Nodes – these are not considered to be barriers to the Operator’s travel.



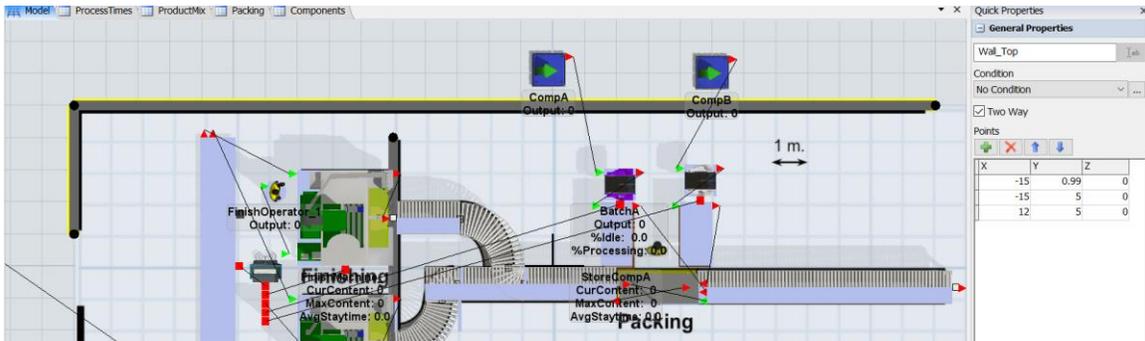
Besides the basic *FlexSim* modeling objects that are used in the model, the only other thing the Operator should avoid are the facility’s walls. While it is not necessary to include all of the walls on the layout, they are added in this case to get practice with the A* tool.

The Operator needs to avoid the walls in the system. The Divider tool is used to add walls to the model.

- It is best to zoom out so that the entire layout is visible on the screen. Also, it is suggested to Reset the view to the planar or two-dimensional view. (Recall, this is done by right clicking anywhere on the modeling surface and selecting **View**, then **Reset View**.)
- Select the Divider object from the Object Library and click on one end of the vertical wall to the right of the finish machines. (This wall divides the finish area from the packing area). Then click on the other end of the wall on the layout. Press the Esc key to stop drawing the divider. This results in a line segment with a circle at each end, as shown in the figure to the right.
- Adjust the shape as necessary by dragging a circle end point or the line segment. The wall can be located more precisely by double clicking the Divider object and modifying the table values, as shown in the figure to the right. Each row in the table is the x-y-z coordinate of one of the circles on the Divider. This table may also be accessed in Quick Properties when the object is selected.
- Change the name of the object from Divider1 to **WallBetweenFinishAndPack**.



- Select the Divider object again and click on the end of the vertical wall at the left end of the layout at the opening. Then, click on the wall corner (adding a circle) and finish the wall at the right-hand side of the layout. Press the Esc key to stop drawing the divider. Adjust the shape as necessary by dragging a circle end point or the line segment or modifying the table in Quick Properties, as shown in the figure below. Also, name the object **Wall_Top**.



- Again, similar to the step above, select the Divider object and click on the end of the vertical wall at the left end of the layout at the other end of the opening. Then, click on the wall corner (adding a circle) and finish the wall at the right-hand side of the layout. Press the Esc key to stop drawing the divider. Adjust the shape as necessary.
- On the Visual tab of the A* object, check the **Show Heat Map** box and the **Transparent Base Color** box.

The Operator will now be controlled using the A* algorithm and not the path network created earlier.

- Therefore, zoom in toward the Operator. Disconnect the operator from the network by holding down the Q key, selecting the operator then dragging and selecting the network node where the operator is connected to the network (red line). If this is done correctly, the red line between FinishOperator_1 and nn_Container Storage should be removed.

Note that barriers and dividers have no height; the z-value is only for location of the barrier/divider in the z direction. To represent a wall in a model, use the Walls object in the Visual section of the Object Library.

All of the network nodes for the path could be deleted, but in this case, they are left in the model.

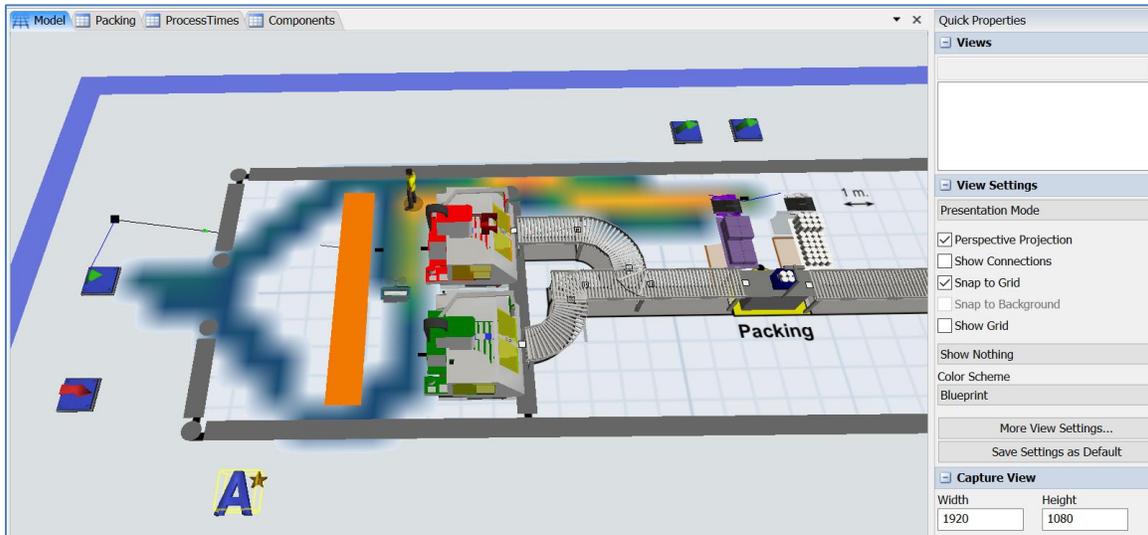
- **Reset** and **Run** the model. Verify that the Operator avoids all of the fixed resources, barriers, and dividers, as shown in the figure below.

➤

In the following figure, all of the connections are removed from view; this is accomplished by:

- Click anywhere on the modeling surface. In the Quick Properties window, change the **View Settings** property from **Working Mode** to **Presentation Mode**.

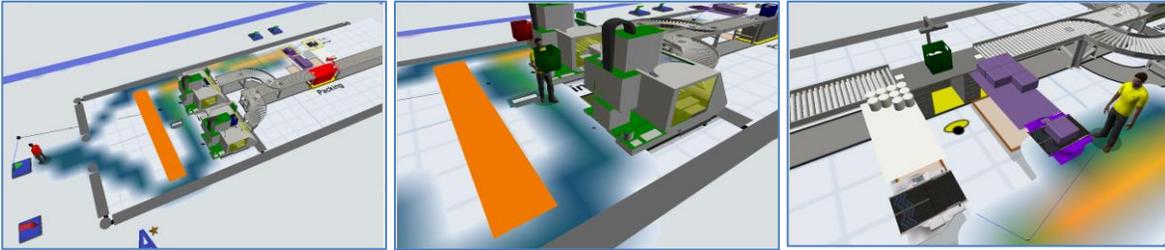
For clarity, you may want to uncheck all of the boxes on the Visual tab of the A* object. Of course this does not affect the underlying logic, it just makes the model a bit cleaner.



If you haven't already done so, save the model. Recall, it is good practice to save often.

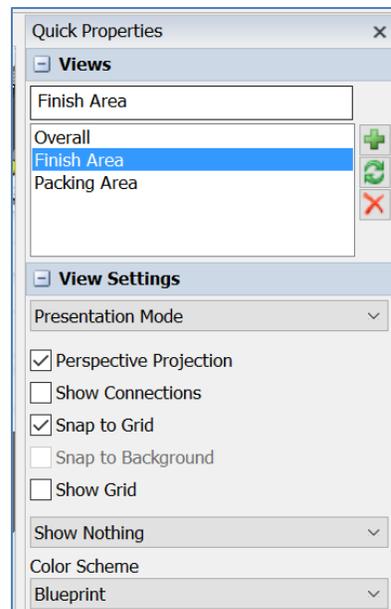
2.3 Saving model views

While not related to the A* algorithm, it is an opportune time to introduce the ability to set and save multiple model views. With saved views, various aspects of the model can be examined quickly. The following three views will be created and saved.



Model views are defined in the **Views** section of the Quick Properties window. As shown, in the figure to the right, the three model views that are listed – Overall, Finishing, and Packing – are for the views shown in the figures above.

- Define each of the three views by orienting the 3D view of the model to the desired look, then pressing the + button in the Views section of the Quick Properties window.
- Each view should be renamed to a meaningful name, e.g. Overall.



Views can be accessed anytime by clicking on the desired view in the Views section of the Quick Properties window.



If you haven't already done so, save the model. Recall, it is good practice to save often.

3 USING AN ITEM LIST FOR COMPLEXMODELING LOGIC

In the finishing area of the primer example, the current model uses the default logic in the Queue object to determine the order in which containers are processed. That default logic is first-come, first-served (FIFO); i.e., items are processed in the order in which they arrive. In other words, the items that have waited the longest are processed first. This is a common ways for queues to operate; many human systems operate this way because it is perceived to be the fairest.

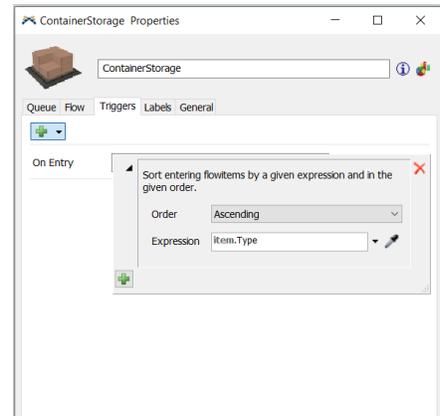
However, there are exceptions to this means of processing, such as when items are processed by priority. For example, patients in a hospital waiting room do not follow the FIFO rule - patients are triaged based on the seriousness of their situation. Also, customer orders are oftentimes processed before those that have waited longer, if they are designated as close to a due date or late, or if the customer paid a premium to get the order processed quickly.

3.1 Simple priority routing using an object trigger

In the primer example, the desire is to process containers by using the shortest processing time (SPT) rule – process the quickest items to process first. In production control theory, processing by SPT results in more throughput than by using FIFO. Based on the process times, the preferred order for finishing containers is: Type=1 first, then Type=2, and then Type=3. (Recall, finish times are 15.0, 20.0, and 30.0 minutes for types 1, 2, and 3, respectively.)

To represent this in *FlexSim* is quite easy. As shown in the figure to the right, the contents of the Queue can be reordered whenever an item enters.

- On the Triggers tab of the Queue ContainerStorage, add an **OnEntry** trigger via the + drop-down menu.
- On the OnEntry trigger, select the **Control** category, then **Sort by Expression** in the drop-down menu.
- As shown in the figure to the right, change the **Order** property from the default **Descending** to **Ascending** using the drop-down menu. Ascending is chosen because the Type=1 containers should be processed first, then Type=2, etc. ; i.e., items should be arranged in the Queue by increasing/ascending values of the label Type.
- The default **Expression** property, **item.Type**, is used as the sort criteria; i.e., the value of the label Type on the items in the Queue. If there are multiple items of the same type in the Queue, they are arranged in the order in which they arrived. Therefore, there are basically two ordering rules: first, order by type, then by age.



Note that this approach changes the order of the items in the Queue – the contents of the Queue is sorted each time an item arrives to the Queue and the item at the front of the queue is sent to the first available object (a Processor in this case). Thus, containers are finished (processed at the Processors) based on the SPT or priority rule. Many systems behave this way.

However, other systems maintain the order in which items arrive and then items are selected from the Queue based on the SPT rule. This would be the case if items arrive on a conveyor - items maintain the order in which they arrive since they cannot physically be reordered. While in many cases this is a subtle difference, it can affect performance. For example, in this case, if a priority container is the last to arrive, then the finish operator needs to travel to the end of the queue in order to load the container, rather than taking it from the front of the queue, resulting in a longer travel time. Of course, the level of detail is always an important modeling decision. In many cases, it might be best to use the quick change (**OnEntry** trigger to reorder the Queue), make and note an assumption that this does not significantly impact performance, and then revisit the assumption later.

The use of the List tool, described in the next section, does not reorder the queue, it just identifies the next item to process based on a specified rule.

3.2 Multiple-criteria routing using Lists

One drawback of using the SPT rule in this example is that the Type 3 containers may wait a very long time. This may occur because if there are any Type 1 or 2 items in the Queue when a finish machine becomes available, then they will be processed before any Type 3. This issue is addressed by using the following rule: *process containers based on Type unless some items have waited “too long,” then process them first.* Of course, “too long” must be specified, such as one hour; this is referred to as the *wait threshold*. The List tool is used to implement this more complex rule.

Prior introducing Lists, two new constructs, “pull” logic and “push” logic are defined.

- **Push logic.** In the current model, items are “pushed” from the Queue to the Processors (from ContainerStorage to FinishMach_1 and FinishMach_2); i.e., the routing decision is in the Queue and it pushes items to the first available Processor by default or by a rule specified in the **Send To Port** trigger on the Flow tab of the Queue. The default rule is **First Available**, but many others are available on the drop-down menu, such as **Random**, **Round Robin** (alternate between machines), **By Expression** (using the value of a label, such as Type), etc.
- **Pull logic.** The routing decision between the Queue and Processors is moved from the Queue to the Processors. Processors pull items based on a specified criteria, rather than having items pushed from the Queue. The pull logic is enabled on the Processor’s Flow tab by checking the **Pull** box in the **Input** section of the tab. The values for **Pull Strategy** and **Pull Requirement** are then be specified.



Begin with the basic model from the previous section, named Primer_3-4, and **Save As** Primer_3-5. Basically make a copy of the model so that it can be customized and the original model is retained.



[Section 6-2] Using Lists for decision making

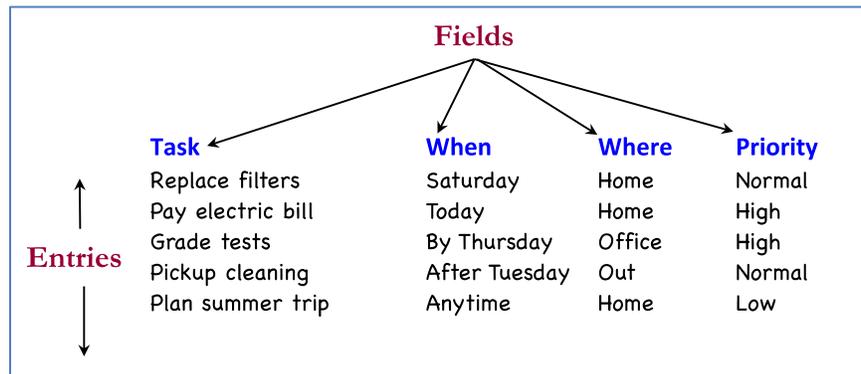
Before proceeding with the model, it may be better to turn off the heat map from the A* algorithm.

- Uncheck the **Show Heat Map** on the Visual tab of the A*Navigator object.

Lists, also called Global Lists, have many uses in modeling. Lists are tools that can create complex flows between objects. There are many types of lists available in *FlexSim* - Fixed Resource List, Item List, Task Sequence List, Task Executer List, and General List. However, in this example, an Item List is used to implement the routing logic specified above.

Information is *pushed to* and *pulled from* lists when events occur. Typically, an event causes an **entry** to be placed on a list and another event causes an entry to be removed from a list. The information about an entry is contained in **fields**. The overall structure of a list is like a table.

As a simple example, consider the simple To-Do list in the figure below. An entry is something that needs to be done and is a row in the list. Entries are put on (pushed to) a list as a result of an event occurring. For example, in the To-Do List, when we think of something or are told to do something, we make an entry onto, or push to, the list. Entries are typically added at the bottom of the list. Entries are removed (pulled) from a list, again when an event occurs. For example, in the To-Do list, an entry is removed, or pulled from, the list when it is completed or deemed no longer necessary to be done.



The columns in the list are the *fields* and represent characteristics of entries, such as in the To-Do List, a brief description of what needs to be done (Task), when it needs to be done (When), where it needs to be done (Where), and how important it is (Priority). Thus, each entry has a *value* associated with each field.

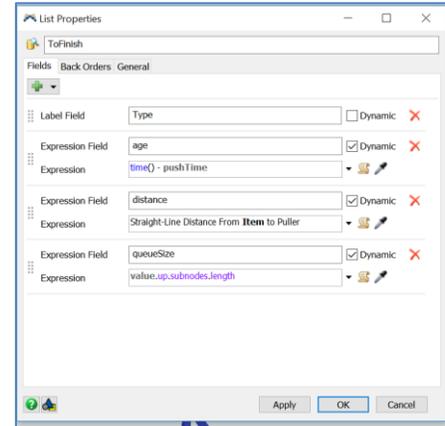
In the case of the primer example, as items enter the Queue, the Queue *pushes* information about the item to a List, i.e., a **list entry** is made. When available, Processors scan all of the list entries and *pull* an item to process based on a specified criteria. The scanning of the list and selecting an item is referred to as a **query**.

The threshold value for the longest time an item should wait is stored in a Global Table called Parameters. This way the threshold value can be easily changed. Actually, simulation experiments with this model will be used to set the threshold value in order to find the right balance of maintaining the SPT rule, but not allowing items to wait too long.

- Set up a new Global Table named **Parameters**. For now, it will contain just one cell; also, assume the threshold value is 30 minutes. Change the Row Heading value from Row1 to Wait Threshold.

- Create an **Item List** from the Global List tool in the Toolbox and name it **ToFinish**, as shown in the figure to the right.

Fields must be defined for the List and how their values are determined for each entry; i.e., what information about the item is put into the list and how that information is determined. Fields may be of different types, e.g. a label value on the item or calculated using an expression that is defined in each field's **Expression** property.



By default, *FlexSim* provides four fields, as shown in the figure to the right and defined below. Fields are either **Dynamic** (updated as affected states change in the simulation) or not (use and maintain the current value).

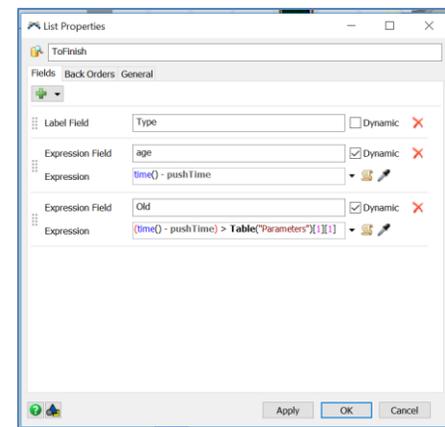
- *Type* is the value of the item's label Type at the time it is placed on the list (not Dynamic).
- *age* is how long an entry has been on the list. The value is calculated based on the **Expression** defined as: **time() – pushTime**. This means that the age of an item on this list is the current simulation time, time(), minus the simulation time when the entry was put onto the list, pushTime. Since the **Dynamic** box is checked, the value is constantly updated while the entry is on the list.
- *distance* is how far (straight-line distance) the item currently is from the object that will pull the entry from the list. The value is calculated based on the **Expression** defined as a drop-down menu option. Since the **Dynamic** box is checked, the value is constantly updated while the entry is on the list.
- *queueSize* is how many items are in the object where the item is currently located. The value is calculated based on the **Expression** defined as a drop-down menu option. Since the **Dynamic** box is checked, the value is constantly updated while the entry is on the list.

In the primer example, only the first two fields are needed. Therefore, the last two can be deleted. Of course, there is no problem leaving them on the List.

- Delete the **distance** and **queueSize** fields by using the red x next to the **Dynamic** property for the field.

In this example, additional information, beyond the default fields, is needed. The routing logic is based on whether an item has waited longer than a threshold value, not just how long it has waited, which is what the **age** field contains.

- Using the + on the List Properties window, just under the Fields tab name, create a new **Expression** field that will indicate if an item is “old,” i.e., has been on the List longer than the threshold value that is specified in the Global Table named Parameters. As shown in the figure to the right:



- Replace the default field name, fieldName in the **Expression Field** with the name **Old**.
- Check the **Dynamic** box.
- Replace the **0** in the **Expression** box with the following: **(time()-pushTime) > Table("Parameters")[1][1]**
 - The expression to the left of the > (greater than sign) can be copied and pasted from the age expression above. Be sure the expression is within parentheses, (...).

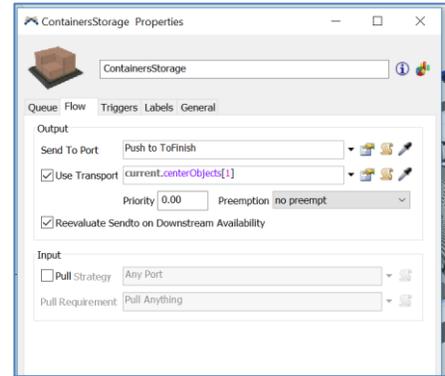
- The context-sensitive editing in *FlexSim* enables the table name to be selected from a list of all Global Tables in the model as the expression is being entered

The “Old” expression checks to see if the amount of time that the entry has been on the List exceeds the threshold value, which is stored in the first row and first column of the Parameters table. If the time on the List exceeds the threshold, then the comparison is true and the field will have a value of 1. If the entry has not be on the List longer than the threshold value, then the comparison is false and the field will have a value of 0.

Now that the List is defined in terms of the information that it contains, the model needs to push information to the List and pull information from the List in order to decide what to process next.

Push an item’s information to the List from the Queue (ContainerStorage), as shown in the figure to the right.

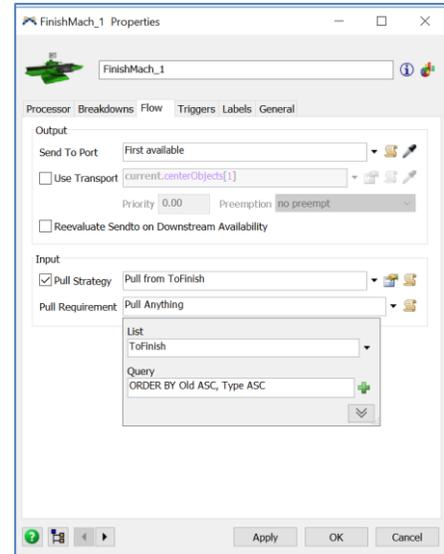
- On the Flow tab, change the Send To Port trigger from **First Available** to **Use List**, and then **Push to Item List**. Make sure the **ToFinish** list is shown in the menu.
- Check the box for **Reevaluate Sendto on Downstream Availability**.
- While in the Queue object, use the outer X icon to remove the **On Entry** trigger **Sort by Expression**. When doing so, a dialogue box will appear asking “Remove this trigger and its logic?”; press OK. The List logic is now controlling the item selection from the Queue so the Queue no longer needs to be sorted when items enter.



The finish machines will use information in the List to decide what to process next. It will:

- query the current List,
- select an entry based on the specified criteria,
- pull the information from the List, and
- pull an item from the Queue based on the information.

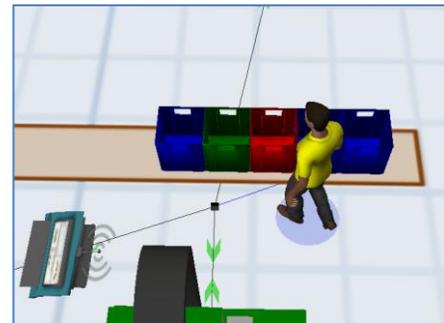
- The above logic, as described below and as shown in the figure to the right, is implemented on the Flow tab of each of the finish machines (Processors).
- Check the **Pull** box on the Input section.
- For **Pull Strategy**, select **Use List**, then **Pull from Item List** from the drop-down menus.
 - Be sure the **List** property value is set to **ToFinish**.
 - Enter two sort queries using the +. The query is to first sort the List based on the Old field, then by the Type field. To do this:
 - Select **Order By (Sort)** from the drop-down menu, then **Old**. Change **ASC** to **DESC**.
 - Again using the +, select **Order By (Sort)** again from the drop-down menu, then **Type**. Leave the default **ASC**.
 - The resulting syntax shown in **Query** should be: **ORDER BY Old DESC, Type ASC**



This command will sort the List and place the entries in the specified order. The List is first sorted in descending (DESC) order of the values in the field Old. The values with Old=1 (indicating they have been on the List longer than the threshold) are placed at the top of the list and those with Old=0 are at the bottom of the List. The List is sorted again by the values of item's Type and they are put in ascending (ASC) order of Type. As a result of the query, any entries in the list that are beyond the threshold (Old=1) and a high priority (Type=1) will be at the top of the list.

The finish machine will pull the flowitem from the Queue that is associated with the entry at the top of the List and the entry is subsequently removed from the List.

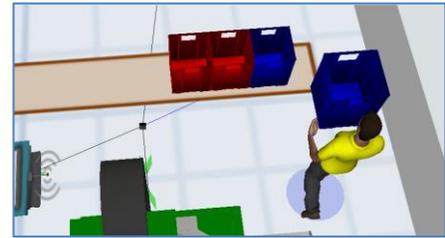
- Be sure the above pull logic changes are repeated on the other finish machine.
- **Reset** and **Run** the model. Notice, as in the figure to the right, the operator does select the intended item in the queue. The operator is moving to the red container (Type 1) for processing even through there is an item (blue) in front that has waited longer.



- To view the current contents of the List, double click on the ToFinish List in the Toolbox. On the General tab, select **View Entries...** In the example to the right, based on the criteria, the next one to be processed is the Type 1 even though one other item has been in the queue longer.

value	Type	age	Old
/ContainersStorage/Container 3	3	14.20	0
ContainersStorage/Container~2 1	1	5.67	0

In the figure to the right, the operator correctly selects the first item in the queue, even though there are higher priority items in the queue. This is because the blue item has been in the queue longer than the 30-minute threshold.



In the example to the right, based on the criteria, the next one to be processed is the Type 3 item even though another item has a higher priority. However, the Type 3 item has waited longer than the specified 30-minute threshold, as indicated by its value for the field Old is 1.

value	Type	age	Old
/ContainersStorage/Container 3	3	31.98	1
ContainersStorage/Container~2 3	3	18.12	0
ContainersStorage/Container~3 1	1	0	0

The **View Back Orders...** option on the General tab of the List Properties displays the resources that are awaiting flowitems. In figure below, the view on the left is for Backorders and the view on the right is for Entries. In the case below, the finish machines are on the Backorders List because they are idle – they want to pull items from the List, but there are none to pull. The Entries list is empty, no items are in the queue awaiting processing.

puller	query	requested	required	fulfilled
/FinishMach_1	ORDER BY Old DESC, Type ASC	1	1	0
/FinishMach_2	ORDER BY Old DESC, Type ASC	1	1	0

value	Type	age	Old



If you haven't already done so, save the model. Recall, it is good practice to save often.

There is not a lot of queuing or waiting in the model, at least so far. Therefore, it may be difficult to find good test conditions, especially for the routing rules between the Queue and Processors. One way to test the logic is to temporarily change the Process Times (column 1) values in the Process Times Global Table; e.g., doubling the times. Then the model can be run and the logic tested. Of course, the changed parameter(s) need to be reset to their intended value(s).

4 EXPERIMENTATION IN *FLEXSIM*

Simulation models are created for analysis. Analysis is based on the output of a simulation model and is used to support decision making and problem solving.

In the models considered so far, output has been limited to a single run. Also, output has also been limited to basic object statistics that are displayed on the object and in the Quick Properties window. While these are good for testing a model's behavior, they are insufficient for analysis.

Analysis must be based on multiple runs of the model, called *replications*.

Analysis also needs to be focused on the *performance measures* that are important to the decision being made or the problem being addressed.

Analyzing simulation models requires a foundation in probability and statistics and includes a variety of methodologies. It is beyond the scope of this primer to discuss simulation analysis - it only introduces the means in *FlexSim* to carry out experiments and obtain output from a simulation model. For more information on simulation analysis, see:



[Chapters 10 and 11] Fundamentals of Output Analysis and Multi-scenario Experimentation and Optimization

Two examples of experimenting with the current simulation model are discussed below.

4.1 Effect of buffer size on performance

The first Experimenter example is used to assess the affect on performance of the size of the containers' buffer (ContainersStorage object) that is located prior to the finishing area. The affect is measured by the number of containers that have to be redirected to another area because there is no space in the buffer for an arriving container.

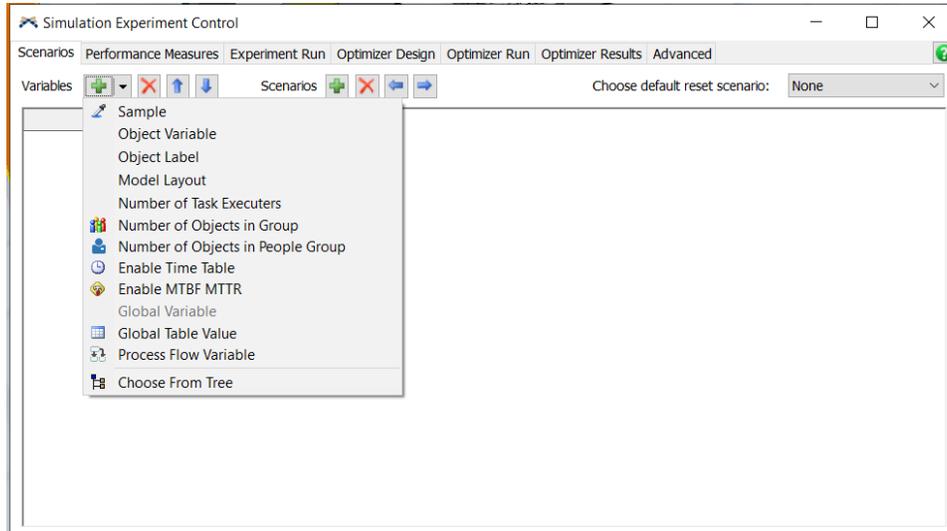


Begin with the basic model from the previous section, named Primer_3-5, and **Save As** Primer_3-6a. Basically make a copy of the model so that it can be customized and the original model is retained. It is denoted as 6a since two experiments are defined below.

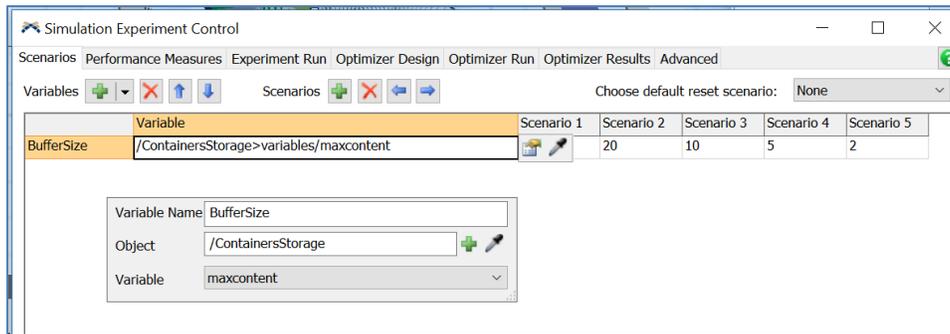
- Launch the **Experimenter** by selecting it from the Statistics drop-down on the Main Menu or from the Statistics section in the Toolbox. This opens the **Simulation Experiment Control** window that contains multiple tabs. Only the first three tabs – Scenarios, Performance Measures, and Experiment Run – are discussed here.

The first tab, Scenarios, is for specifying what alternatives are being considered. These typically represent a combination of values for the decision variables. In the first example, there is one decision variable, the size of the incoming container buffer, i.e., the capacity of the queue ContainersStorage.

- To add a decision variable, click the drop-down menu next to the + in the **Variables** section at the top of the interface, as shown in the figure below.

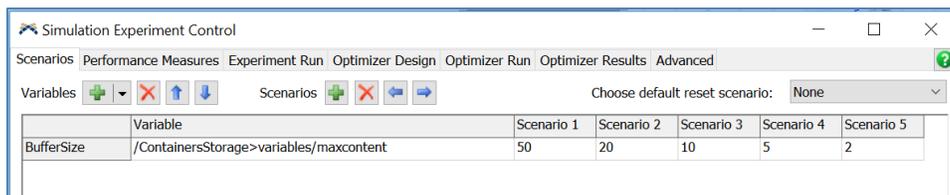


- Select **Object Variable**. And define the properties as follows and as shown in the figure below.
 - Change the **Variable Name** from Variable 1 to **BufferSize**.
 - Set the **Object** property by using the + , then select **ContainersStorage** from the Queues category.
 - Use the default value for the **Variable** property, **maxcontent**. This references the *FlexSim* variable that is defined by specifying **Maximum Content** on the Queue tab of the selected Queue object.



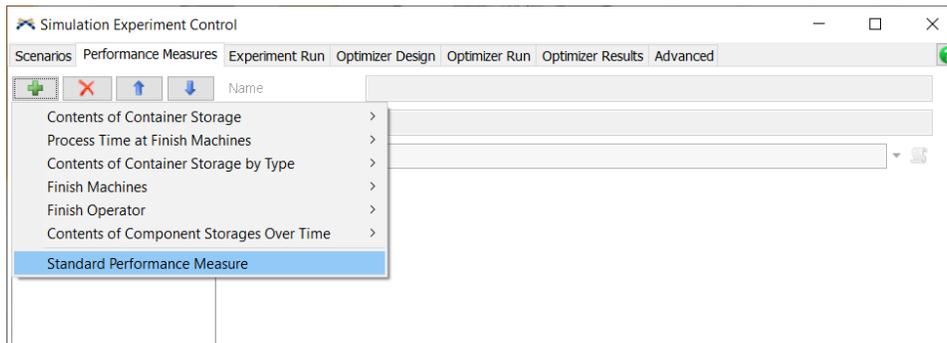
As shown in the figure below, the **Variable** property for each decision variable is a path, defined by *FlexSim*, to reference the object.

- To add and define scenarios, click the + button on the **Scenarios** section at the top of the interface, as shown in the figure below. In this case, create five scenarios and enter the values 50, 20, 10, 5, 2, respectively. Simulations will be run for each of these scenarios in order to evaluate the effect of different buffer sizes. Recall, the currently assumed buffer size is 50.



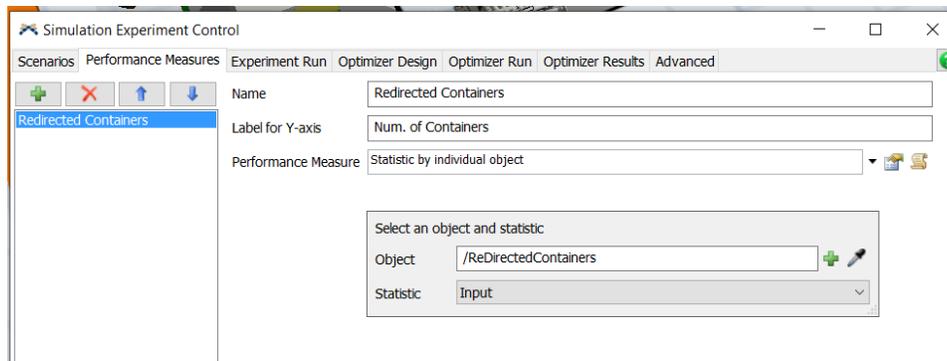
The second tab, Performance Measures, is for specifying how the alternatives are evaluated in order to either (1) understand the effect on the system of changes in the value of the decision variable or (2) to help select the best option. In this example, there is only one performance measure, the number of containers that arrive for which there is no place to store them in the buffer. The measure is the number of items sent to the Sink named ReDirectedContainers.

➤ To add a performance measure, on the Performance Measures tab click the + button and then select from the drop-down menu. As shown in the figure below, select **Standard Performance Measure**, which is at the bottom of the list. Note that the options above it are measures that were created on the Dashboards; thus these measures can be used as Performance Measures in the Experimenter.



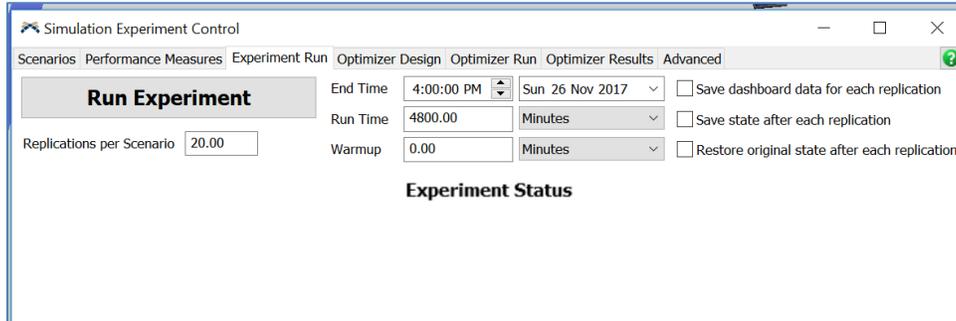
Define the properties for the performance measure as follows and as shown in the figure below.

- Change the **Name** of the measure from PFM1 to **Redirected Containers**.
- Change the **Label for Y-axis** property from the default Value to **Num. of Containers**. On any plot this is the text that describes the units of the measure.
- For **Performance Measure**, select the option **Statistic by individual object** from the drop-down menu. On the resulting interface:
 - Change **Object** from the default Operator1 to the Sink named **ReDirectedContainers** by using the +, then selecting ReDirectedContainers from the Sinks category.
 - Change the property **Statistic** from the default value, Output, to **Input** using the drop-down menu.. Thus, the measure is the total number of items redirected from the buffer that enter the Sink.



The third tab, Experiment Run, is for specifying the run conditions. In this case, just change the following properties, as shown in the figure below.

- **Run Time** to **4800** (minutes).
- **Replications per Scenario** to **20**.

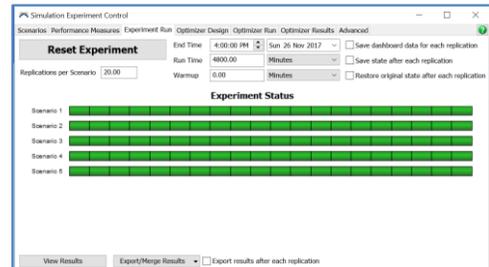


The Experimenter will run each of the five scenarios defined earlier 20 times, with each run being 4800 minutes. Since there are five scenarios and 20 replications of each, 100 80-hour simulations will be performed.

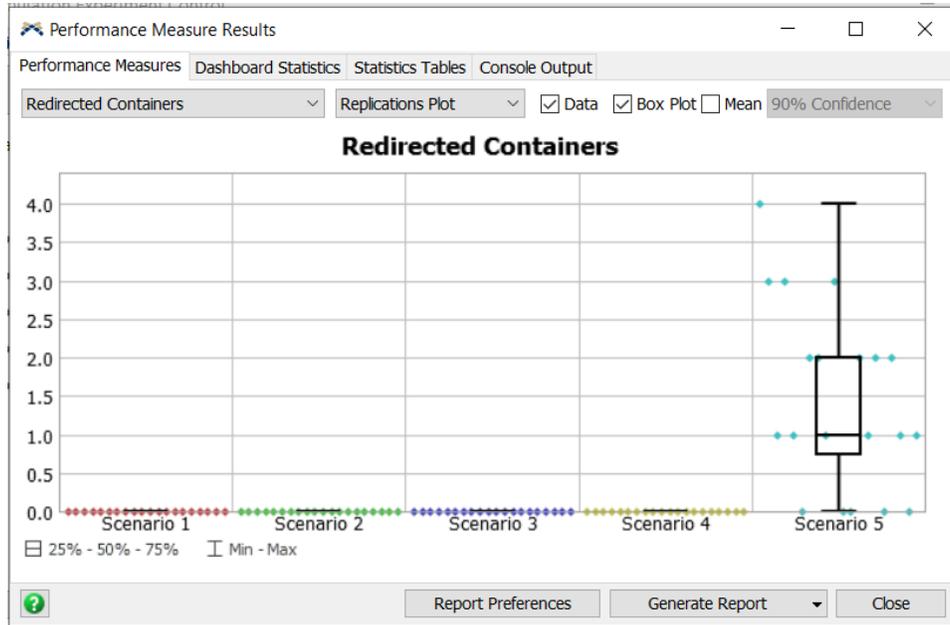
- Press the **Run Experiment** button.

As shown in the figure to the right, as each replication of each scenario is run, its cell in the interface changes from red to green. When all of the cells are green, the results can be viewed.

- To view the result, press the **View Results** button in the lower left portion of the interface.



As shown in the figure below, one output from the Experimenter is a plot of the performance measure value for each scenario and for each replication within the scenario. Note that in this case, items are diverted to the ReDivertedContainers sink only in the fifth scenario, where the Buffer Size is 2. In this scenario, items were diverted due to a full storage area fifteen times in 20 replications. Also in this case, of the 20 replications, no items were diverted five times, one item was diverted six times, two were diverted five times, three were diverted three times, and one item was diverted four times.



Therefore, setting the buffer size to two should not be a problem since only a few of the approximately 240 arriving containers in 80 hours will be diverted. The 240 estimated arrivals is obtained by dividing the simulation run time (80 hours) by the average inter-arrival time (20 minutes). However, in this example, to be a bit conservative, we'll use a buffer size of five,

- Change the **Maximum Content** of the ContainersStorage Queue from 50 to **5**. Recall, this property is on the Queue tab.



If you haven't already done so, save the model. Recall, it is good practice to save often.

4.2 Effect of product mix on performance

The second Experimenter example is used to assess the effect on performance of the mix of products (percentage of Type 1, 2, and 3 containers). The effect is measured by the throughput of containers – the number of containers that leave the packing area per 80 hours (length of a simulation run).



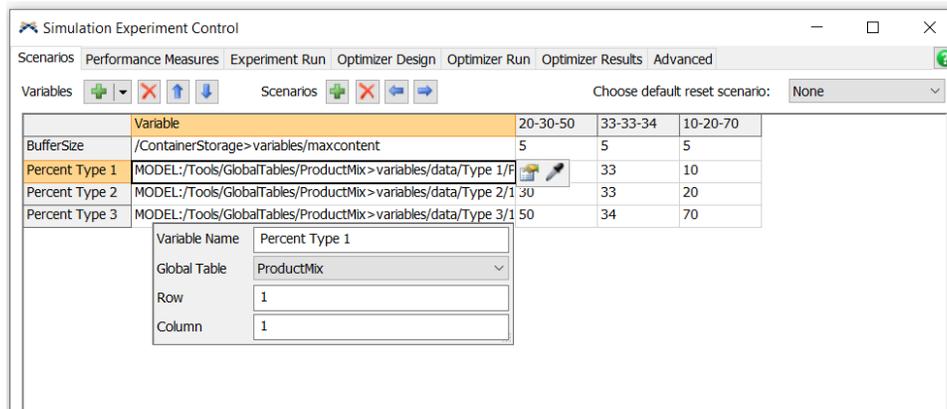
Begin with the basic model from the previous section, named Primer_3-6a, and **Save As** Primer_3-6b. Basically make a copy of the model so that it can be customized and the original model is retained.

Three levels of product mix are considered in this example. The first is the current mix: 20% Type 1, 30% Type 2, and 50% Type 3 (denoted as 20-30-50). The other two mixes are: equal mix, 33-33-34 and more Type 3, 10-20-70. Therefore, three decision variables are added to the Scenarios tab, one for the percent mix of each container type.

Set the properties of the Scenarios tab as shown in the figure below. Recall, the values for the percentage of each type of container are stored in the Global Table ProductMix. Therefore, the Experimenter needs to change the table value for each scenario.

- Add three variables, one for each of the three products. As in the previous example, use the drop-down menu for **Variables** that is just to the right of the + button. However, in each case, select the **Global Table Value** option. For the first type of container:
 - Change the value of **Variable Name** from **Variable 2** to **Percent Type 1**.
 - For the **Global Table** property, select **Product Mix** from the list of tables on the drop-down menu.
 - Leave the default values for **Row** and **Column** at **1** and **1** since the percent value for container type 1 is located in cell (1, 1).

The resulting Variables and **Variable** value for **Percent Type 1** should look as follows.



- Repeat the above steps for container types 2 and 3 as follows.
 - Change **Variable Name** to **Percent Type 2** and **Percent Type 3**, respectively.
 - Set **Row** to **2** and **3**, respectively.
 - The properties **Global Table** and **Column** remain **Product Mix** and **1**, respectively.

As defined above, in this case there are only three scenarios.

- Using the X button to the right of the **Scenarios** section, delete Scenario 4 and Scenario 5.
- Rename the scenarios from **Scenario 1**, **Scenario 2**, **Scenario 3** by changing the column headers to **20-30-50**, **33-33-34**, and **10-20-70**, respectively. This is just text and does not affect the experiment, but does improve the readability of the plots.

Based on the analysis from the previous section, the buffer size is set to 5; and, thus is now the same for all three scenarios. Therefore, this Variable could be deleted; however, it will be retained in case it is changed in further analyses.

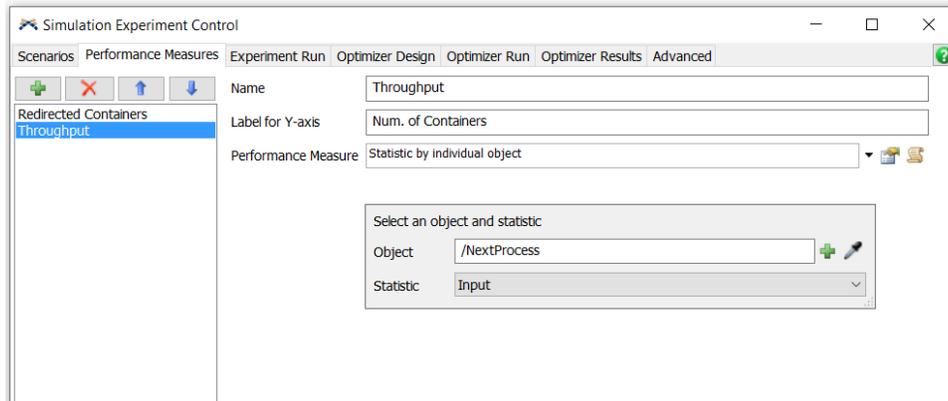
- Change the **Buffer Size** variable to a value of **5** for all scenarios.

The values for each variable are set as follows and as shown in the figure below.

- For the Scenario 1 column, **20-30-50**, the values for the rows **Percent Type 1**, **Percent Type 2**, and **Percent Type 3** should be **20**, **30**, and **50**, respectively.

Defining the properties for the new performance measure, *throughput*, involves basically the same steps as described for the first measure in the previous section. Throughput is defined as the number of items exiting the system, i.e., entering the Sink NextProcess.

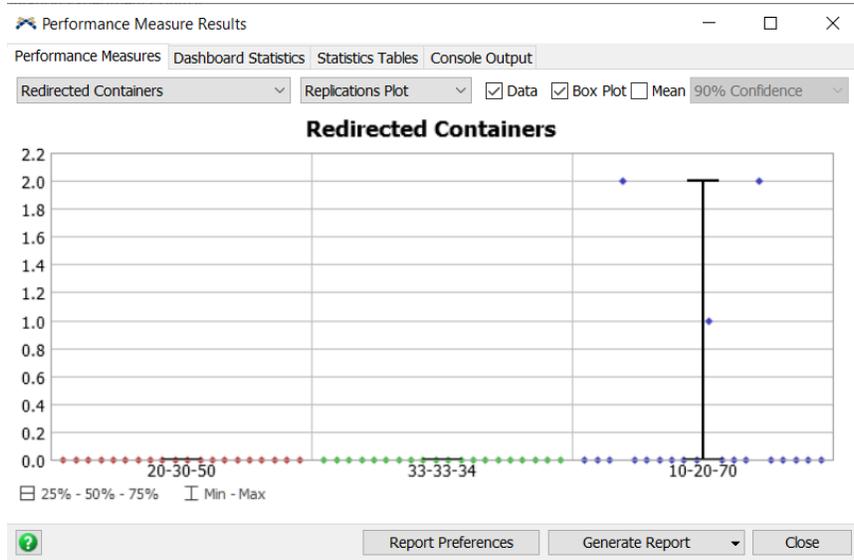
- To add the performance measure, on the Performance Measures tab of the Experimenter, click the + button and then select **Standard Performance Measures** from the drop-down menu.
- Change the **Name** of the measure to **Throughput**.
- Change the **Label for Y-axis** property from the default Value to **Num. of Containers**. On any plot this is the text that describes the units of the measure.
- For **Performance Measure**, select the option **Statistic by individual object** from the drop-down menu. On the resulting interface:
 - Change **Object** from the default Operator1 to the Sink named **NextProcess** by using the +, then selecting NextProcess from the Sinks category.
 - Change the property **Statistic** from the default value, Output, to Input using the drop-down menu.. Thus, the measure is the total number of items that leave the packing area and enter the Sink.



The third tab, Experiment Run, is not changed from the previous experiment, each scenario is replicated 20 times, with each replication being 4800 minutes.

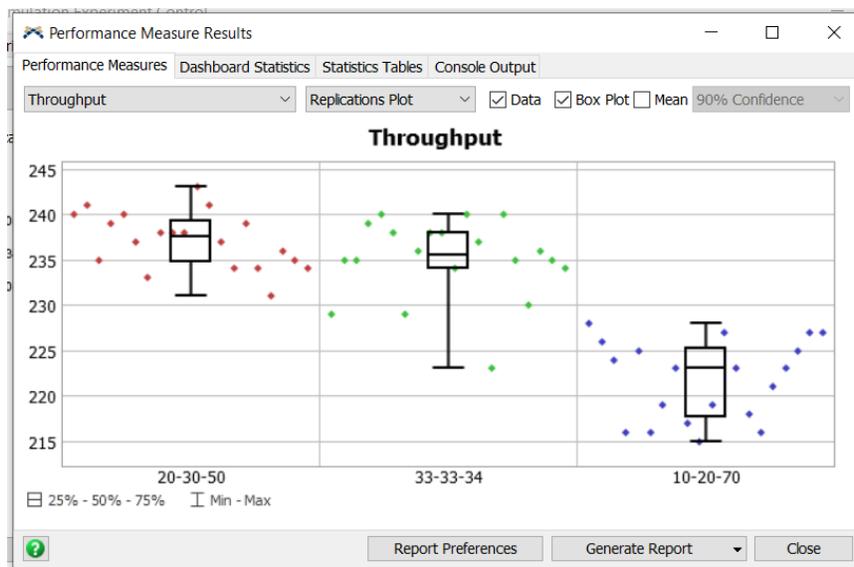
- On the Experiment Run tab, press **Reset Experiment**, then **Run Experiment**, and when all of the cells are green, view the results by pressing the **View Results** button in the lower left portion of the interface.

As shown in the figure below, the product mix with the most Type 3 containers impacts the number of items diverted to the ReDivertedContainers sink. A buffer size of five is used for all cases. In the third scenario, items were diverted in three of the 20 replications, with the most diversions over the 80-hour simulation period being two.



However, throughput is the main concern.

- To see the impact on throughput, click on the drop-down menu in the upper left corner of the interface to change the plot from **Redirected Containers** to **Throughput**. As shown in the figure below, there is not much difference in average throughput between the first two scenarios. However, the third scenario, with the largest percentage of Type 3 containers, results in about a 6% reduction in average throughput.

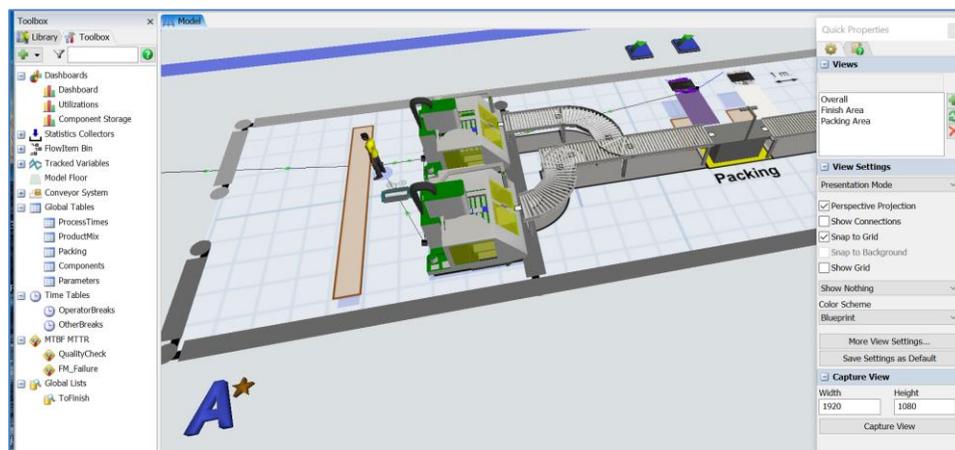


If you haven't already done so, save the model. Recall, it is good practice to save often.

5 SUMMARY OF THE PRIMER FOR *FLEXSIM* 3D

Even though the current model, Primer_3-6b as shown in the figure below, is not very large, it demonstrates many of the key features of *FlexSim* simulation software. The model was built by customizing basic 3D objects in *FlexSim*. and utilizing a number of model-building tools (shown on the left side of the figure below). The model:

- is built natively in 3D.
- is constructed on an imported layout and thus is built to scale in terms of sizes and distances.
- uses standard pre-built *FlexSim* objects that provide basic functionality, e.g., Processors, Operators, Conveyors, and Combiners. These objects:
 - represent fixed and mobile resources.
 - are customized by specifying properties via drop-down menu options and tools.
- considers multiple types of items that have different characteristics; these characteristics affect various aspects of operations, such as processing time, batch size, routing, etc. Labels on objects and items are used to represent the characteristics.
- uses fixed resources to include various types of activities: processors for planned delays, queues for unplanned delays, conveyors for transport, combiners for grouping, separators for ungrouping, etc.
- uses mobile or shared resources for transportation and tasks (unbatching items).
- uses two different methods for controlling the path of mobile resources, such as Operators: path network and the A* algorithm.
- includes planned downtime (e.g. operator breaks and quality checks).
- includes unplanned downtime (e.g. machine failure).
- includes randomness in the time between arrivals of the base item (container) and in the type of arrival.
- includes scheduled arrivals (of components) – time and batch size.
- utilizes both simple and complex routing rules; uses Lists to incorporate complex rules.
- includes imported custom graphics to make the model more realistic.
- utilizes tables for effective data management.
- uses color to facilitate model validation – different colored product and component types and different colored states (down, on break, etc.).
- uses charts and graphs on dashboards to illustrate the dynamic behavior of the system.
- is used for analysis and deciding among alternatives via *FlexSim*'s Experimenter.



This page is intentionally blank.

PART 4 – INTRODUCTION TO PROCESS FLOW

This fourth part of the Primer focuses on developing complex logic using *FlexSim's* logic builder called Process Flow. Process Flow enables users to define and specify logic via a flowchart-like, drag-and-drop interface and link the logic to 3D objects.

After introducing the basic concepts of Process Flow, and defining its modeling environment, this section describes modeling in Process Flow via two examples, both building upon the basic Primer model (finishing and packing containers). The two examples:

1. Implement a reorder-point inventory system for the components that are packed into containers.
2. Create a custom task sequence for the finish operator.

Both examples are not that complex, but they are complex enough that they cannot be implemented via the drop-down menu options on the 3D objects. In the past, and with many other simulation software, the only way to model these system behaviors is via custom coding. In the case of *FlexSim*, this would be via FlexScript. While the logic can still be implemented via FlexScript, Process Flow enables the modeling without coding and makes the logic more transparent. Transparency is enhanced through Process Flow since all of the logic is contained in one place, on the Process Flow interface, and not dispersed in 3D objects. In addition, the logic is more transparent because the logic is represented in an easy to understand flowchart-like format.

Process Flow, like the 3D aspects of *FlexSim*, is a very robust and powerful modeling environment. Therefore, since this primer is intended only to provide a basic introduction to *FlexSim*, only a few features and capabilities are described. However, this should provide a good starting point and a solid foundation for modeling with Process Flow.

While the focus is on Process Flow, in both examples various aspects of the 3D model are revisited and additional 3D features are introduced.

1 BASIC CONCEPTS

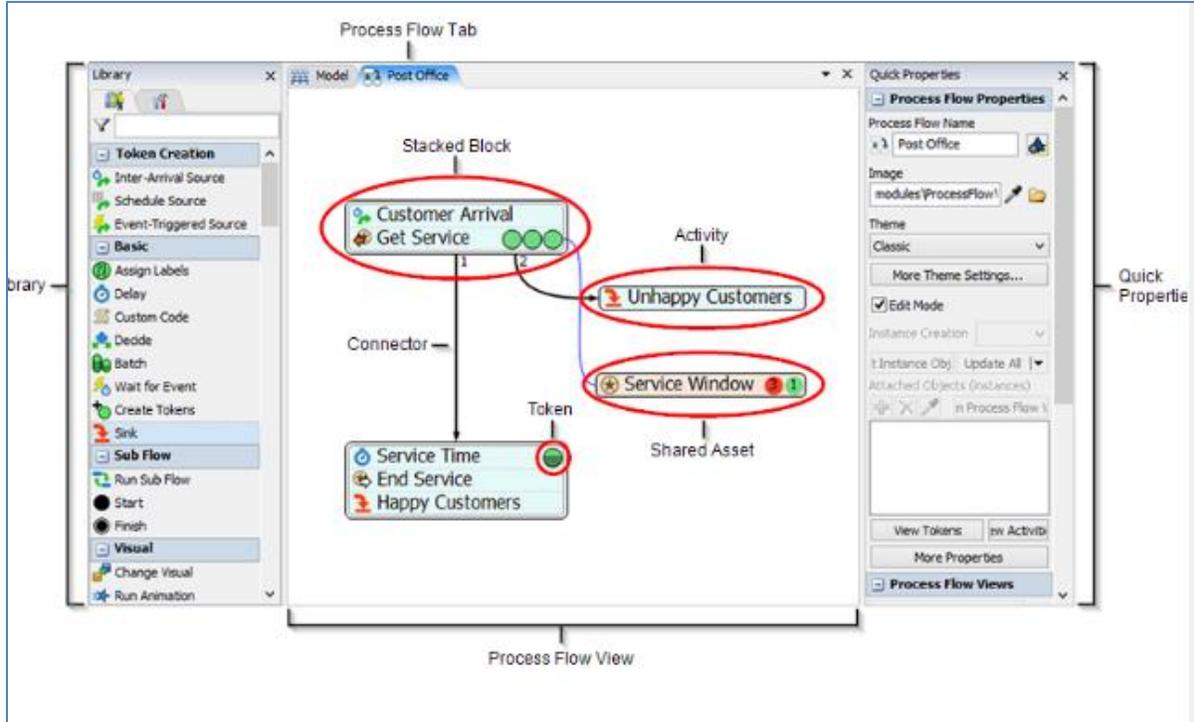
This first section introduces the basic elements of Process Flow and its modeling environment. While the interface and library of modeling tools are different than in 3D, it operates in a similar manner. As with 3D objects, Process Flow activities are dragged onto a modeling surface and are connected to represent the way a system behaves. This section also describes the basic operation of a reorder-point inventory system.

1.1 Basic Process Flow concepts and modeling environment

Process Flow is a part of *FlexSim* that is used to add complex processing logic into a simulation model. The logic is used to define interactions among *FlexSim* objects and other elements as a simulation runs. While *FlexSim* provides a lot of logic-modeling capability through its drop-down menu options on 3D objects and tools, there is a limit as to how much can be pre-specified. Typically, custom logic requires scripting or writing computer code to implement the more complex logic. This can be done through *FlexSim*'s programming language, called FlexScript, a subset of C++. However, Process Flow, reduces the need for programming by using a set of logic-building activities that can be defined and combined via a flowcharting type of paradigm.

As with any modeling endeavor, it is always best to build the logic in steps.

Process Flow has its own logic-building interface, analogous to the 3D modeling surface, and its own set of modeling objects, referred to as *activities*. The basic components of the Process Flow modeling environment and their terminology are shown in the figure below. The Process Flow View is accessed via its icon in the Main Menu or via the Toolbox. The logic is built by selecting and combining activities.



Activities are the building blocks of Process Flow - they are logical operations or steps in a logical process. Activities are analogous to objects in 3D since they are dragged from their library onto a Process Flow modeling

surface or workspace. Also like 3D objects, activities have properties that define their behavior. Activities are grouped into the following categories. This primer only addresses a subset of the activities, but the full list is provided so that the reader is aware of the capabilities of Process Flow.

- *Token Creation* activities are analogous to a Source in 3D. Tokens can be created by inter-arrival times, schedule, and by “listening” for specific events to occur in a 3D model.
- *Basic* activities include assigning labels, implementing a delay in a flow, waiting for an event to occur in a 3D model, making decisions, destroying tokens, etc.
- *Sub Flow* activities are used to create a separate process flow that contains logic that may be used by multiple objects or activities; it is analogous to a function or subroutine in computer programming.
- *Visual* activities change the appearance of an object in the 3D model or triggers an animation on a 3D object.
- *Object* activities are used to create, move, or destroy objects such as flowitems, fixed resources, task executers, etc. in a 3D model.
- *Task Sequences* activities are used to build custom task sequences that are assigned to task executers in a 3D model, including travel, load, unload, delay, etc.
- *Shared Assets* activities are used to define, access, and manage Lists, Resources, Variables, and Zones (keep statistics for a group of activities and restrict access to activities based on token properties).
- *Coordination* activities manage relationships between multiple tokens in a process flow, including splitting, joining, and synchronizing tokens.
- *Preemption* activities are used manage interruptions in a token’s flow.
- *Display* activities are used to annotate a process flow with text, arrows, and images; they do not affect the logic.
- *Flowchart* activities are shapes that can help organize and visualize a process flow; like Displays, they do not affect the logic.
- *People Activity Sets* are preconfigured activities that are bundled together that model a basic People Module task, such as Walk then Process, Escort then Process, Wait then Process, etc.
- *People Basic* are activities that create or delete a Person flowitem or process a token.
- *People Resources* are activities that facilitate acquiring and releasing resources (Locations, Staff, Transports, and Equipment) in a People-based model.
- *People Sub Flows* are activities that are tied to pre-built Sub-Flow objects.

A **token** is the basic component of Process Flow logic. Analogous to flowitems in 3D, tokens flow through activities as a simulation runs. *Tokens are typically more abstract than flowitems since they usually represent logic flow, rather than physical flow.* Each token has a unique ID and a set of labels or characteristics. Tokens are depicted as green circles in Process Flow as a simulation runs; they may change color depending on their use and state.

Tokens move from activity to activity either by **connectors** or through **blocks**. A block is a set of activities that have been “snapped” together; i.e., activities that form a single sequence of process flow steps.

Tokens have **Labels**, which are an important part of modeling with Process Flow. Labels store information that is used in the logic. As in the 3D objects, each label has a name and value; the value can be of any data

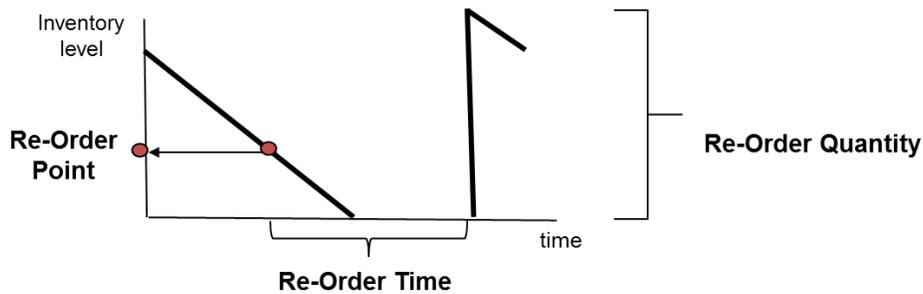
type, e.g., numbers, text, arrays, object references, etc. Token labels are specified in the dot-notation format token.LabelName, e.g. token.Type.



Begin with the basic model from the previous section, named Primer_3-6b, and **Save As** Primer_4-1. Basically make a copy of the model so that it can be customized and the original model is retained.

1.2 Using Process Flow to model inventory policy

Continuing the primer example from the previous section, Process Flow is used to model a reordering system for supplying components to the packing area. Rather than having components delivered in batches on a schedule, as in the current model, components are ordered when their inventory level falls below a specified reorder point. As shown in the figure below, when the inventory level falls below the specified reorder point (ROP), an order is triggered for a specified quantity (ROQ) of the component and that quantity is delivered as a batch after a specified time (ROTime).



Therefore, three new parameters must be specified for each component: ROP, ROQ, and ROTime.

- ROQ is analogous to the batch size.
- ROTime needs to include ordering, fulfilling, and delivery times.
- ROP is specified as a quantity, level of inventory, and not time.

Another parameter that is considered in the model is the initial inventory (IInv), the number of components on hand when a simulation starts. It can be any value above zero. However, if the initial value is below the reorder point, the logic would need to ensure an order is placed when the simulation starts. For simplicity, *assume the initial inventory value is above the reorder point when a simulation starts.*

These additional parameters need to be added to the model.

➤ Update the Global Table named Components as shown in the figure to the right.

- The time between batches will no longer be used since it will be replaced by the reorder point system. However, leave the entry in the table in case there is a need to return to the scheduled-order system. In fact, some components might use the scheduled-order system and some might use the reorder-point system.

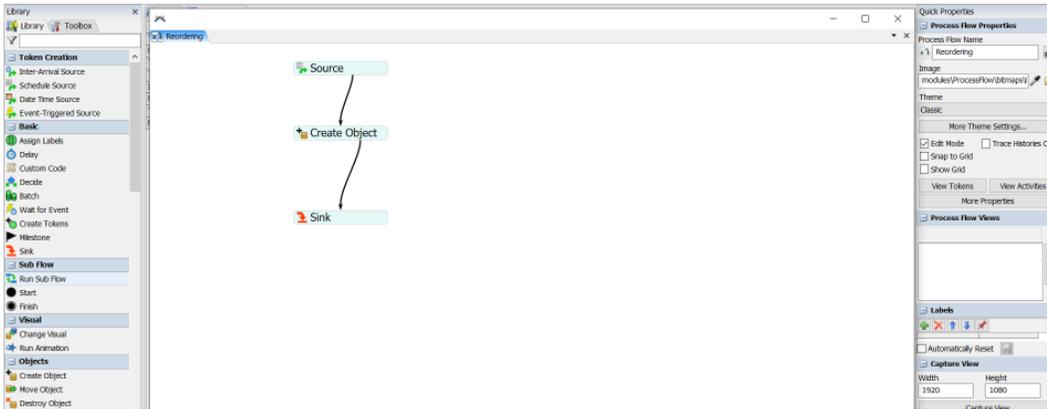
	CompA	CompB
Time between batches	60	30
Batch size (Order Quantity)	24	60
Time to unpack a batch - fixed (minutes)	1	1
Time to unpack a batch - per item (minutes)	0.05	0.05
Initial Inventory	12	30
Reorder Point	6	20
Time to receive an order	120	120
Flowitem ID	10	11

- The former batch size (row 2 in the table) will now be the component’s order quantity. Change the values of the Batch size from their previous values of 3 and 5 to **24** and **60**, respectively. Update the text in the row header to indicate the values are batch sizes.
- Four new rows are added to the table for: initial inventory, reorder point, time to receive an order and flowitem ID. Each of these are explained when used in the sections below; therefore, just enter them in the table for now, as shown above.

2 MODELING INITIAL INVENTORY

A very simple extension of the model is used to introduce modeling with Process Flow.

- Through the Main Menu icon, or via the Toolbox, select **Add a General Process Flow**, and change its name from the default ProcessFlow to **Reordering**.
- As shown in the figure below, drag out three activities onto the Process Flow modeling surface:
 - Schedule Source (from the Token Creation section of the Activity Library)
 - Create Object (from the Objects category)
 - Sink (from the Basic category)
- Link the activities using the Connector – move the mouse over an activity until the mouse cursor changes to a chain, then holding down the left mouse button, drag the arrowed-line connector to the other activity. The activities may be positioned by selecting them with the left mouse button and dragging them to the desired position.



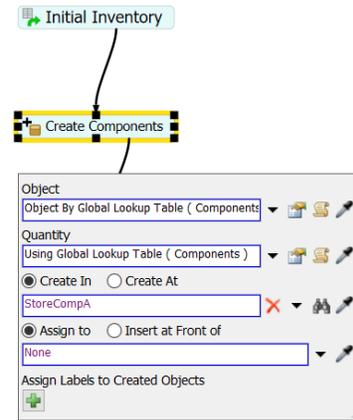
For now, only changes are made to the properties of the Create Object activity.

- However, change the name of the Schedule Source activity from Source to **Initial Inventory** and change the name of the Create Object activity from Create Object to **Create Components**. To do this select the activity and change its property values in the **Activity Properties** section of the Quick Properties window.

Activity properties are changed either by: (1) selecting the activity and editing in the Quick Properties window or (2) clicking on the icon for the activity located in the left-hand portion of the activity’s flowchart shape.

The Create Object activity creates flowitems that represent the components and puts them into their Queue in the packing area. *Note that the flowitems are created directly in the Queue and do not come from the Source.* As a result, these flowitems do not pass through the Queue's OnEntry trigger. Therefore, any logic on the OnEntry trigger would not be invoked for these flowitems.

- Change the Create Object activity as described below and as shown in the figure to the right.
 - The value of the **Object** property is obtained by selecting the drop-down menu option **Flowitem By Global Table Lookup**.
 - **Table Name: Components** is selected from the drop-down menu.
 - **Row: 8**, based on the revised Global Table above.
 - **Column: 1**; this is only for the first component - it will be changed later to a more general setting.



In this case, Row 8, Column 1 of the Global Table contains the value 10; this references the 10th entry in the Flowitem Bin, which is the flowitem created earlier named CompA.

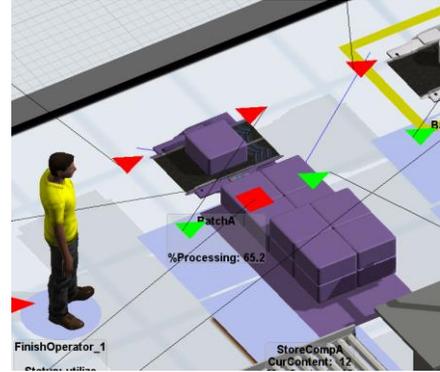
- The **Quantity** property in the Create Object activity is obtained using the drop-down menu option **By Global Table Lookup**. Values for the properties are:
 - **Table: Components** from the drop-down menu
 - **Row: 5**, since the initial inventory values are stored in this row in the table
 - **Column: 1**, for the first component; as indicated earlier, this will be changed later to a more general setting.

The **Creates In** property in the Create Object activity needs to refer to the component's Queue. The default model() is changed to the Queue by using the Sampler tool (pipette or eyedropper icon).

- Select the Sampler and use it to select the **StoreCompA** object in the 3D view. A more general means of setting this will be described later.

The **Assign to** property can be left as is or its value deleted, resulting in the value **None**.

- **Reset** and **Run** the model. At time 0 the inventory for CompA should appear as in the figure to the right – there are 12 purple boxes in the StoreCompA object and one batch of components at the Separator for the Operator to unpack. To obtain this view, you must be very quick on clicking between Run and Stop on the Execution toolbar. Otherwise, the Operator will have unpacked the batch and the Queue will have an additional 24 items.



The **Step** control, next to **Stop** on the Execution toolbar, can be used to move through a model’s execution one event at a time. This is better than trying to start and stop a model quickly. It will likely take several clicks of **Step** for the items to appear since *FlexSim* executes multiple events at simulation time zero.

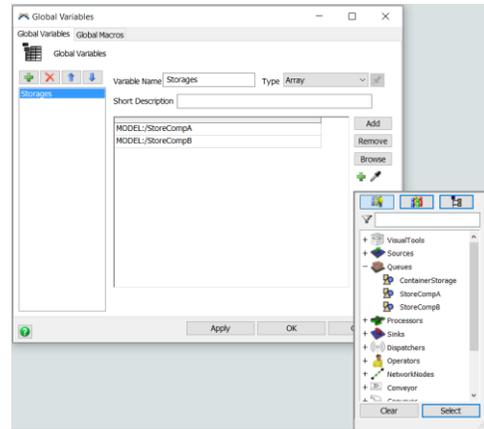


If you haven’t already done so, save the model. Recall, it is good practice to save often.

The modeling steps defined above need to be repeated for CompB. However, initializing the inventory can be generalized so that it occurs for any number components that are included in a model. To do so, the *FlexSim* tool Global Variables is introduced.

Global Variables are user-defined information that can be accessed from anywhere in a *FlexSim* model. The variables can be of various types such as Real (e.g., 123.45), Integer (e.g., 12), String (e.g., ABCdE), etc. In this case, the Global Variable is of the type Array, which means the variable contains an indexed ordered set of values. For example, for the arrayed variable $myVAR = (12, 25, 17)$, $myVAR(1) = 12$, $myVAR(2) = 25$, etc.

- Create an arrayed Global Variable named Storages via the Toolbox – first select **Modeling Logic**, section of the drop-down menu, then select **Global Variable**. As shown in the figure to the right:
 - Change the **Variable Name** from variable1 to **Storages**.
 - Using the drop-down menu, change the **Type** property from the default Integer to **Array**.
 - The elements of the array can be added by using the + drop-down menu or the Sampler, both tools are located below the Browse button on the interface. The figure to the right shows the object selection menu. By either method, add the two components’ storage objects, **StorageCompA** and **StorageCompB**.

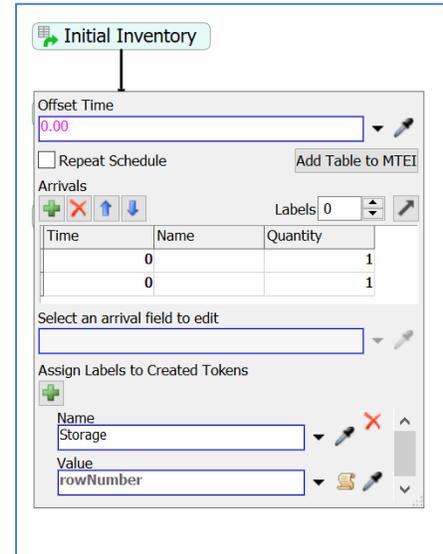


The data have been structured so that the i^{th} component in the packing area is represented as both the i^{th} column in the Global Table Components and the i^{th} element in the Global Variable Storages. Therefore, initial inventory

can be created for each component in a general manner by cycling through all of the components. This concept is now implemented in the Initial-Inventory Process Flow activity.

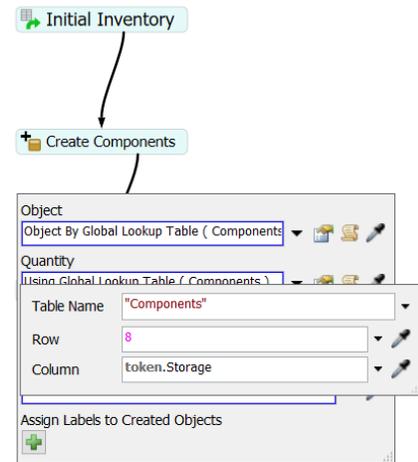
The Schedule Source activity is modified as follows and as shown in the figure to the right.

- The **Offset Time** remains at the default value **0.0** since the initial inventory is added at the start of the simulation.
- One arrival to the process flow is needed for each component; i.e., a token is created for each component that is used in the packing area. Therefore, a row is added to the **Arrivals** table in the Initial Inventory activity. This is done with the + button. Both arrivals are the same; they occur at **Time** value **0.0** and the **Quantity** is **1**, one token per component.
- A label is added to each token by clicking the + button at the bottom of the interface.
 - Change the **Name** property from the default labelName to **Storage**.
 - Change the **Value** property from the default 0.0 to **Arrival Row Number**, an option selected from the drop-down menu. When selected, *FlexSim* enters the command **rowNumber** in the **Value** cell. Therefore, each token will have a label that corresponds to the row number in the arrival table above. This will be used as an index for both the Global Table and the Global Variable. In this case, the indexes are 1 and 2 for CompA and CompB, respectively.



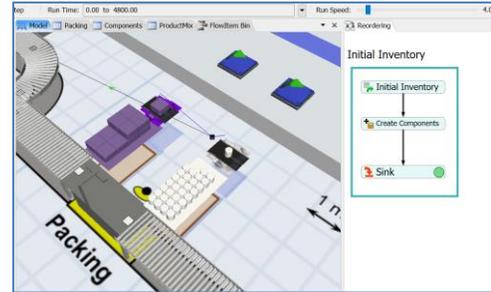
The index is used in the Create Components activity for each component. The activity is modified as follow:

- For the property **Object**, its **Column** value is changed to **token.Storage**; i.e. the column in the Global Table is the index or the number of the component. As shown in the figure to the right, this is accomplished through drop-down menu options in the property **Columns** by first selecting **Labels**, then **Storage**.
- Similarly, for the property **Quantity**, its **Column** value is also changed to **token.Storage** in the same manner as above. The column in the Global Table is the index of the number of the component.
- For the property **Create In**, use the Global Variable Storage and replace the single specific object reference with **Storages[token.Storage]**. There is no drop-down menu option for Global Variables so this needs to be typed in, but *FlexSim*'s context-sensitive help facilitates the entry.
- Create a label on the component flowitem by using the + button in the **Assign Labels to Created Objects** section of the interface. The property values are:
 - Change the **Name** from the default labelName to **Type**.
 - Change the **Value** from the default 0.0 to **token.Storage**, obtained from the drop-down menu **Token Label**.



Tidy up the Process Flow logic. Select the **Process** shape from the Flowchart category on the Activity menu and click near the three activities. The shape's box will surround the activities. As shown in the figure below, reshape the box border and name the shape Initial Inventory. This section of the Process Flow logic can be easily moved as a group.

- **Reset** and **Run** the model. At time 0 the inventories for CompA and CompB should appear as in the figure to the right – there are 12 purple boxes in the StoreCompA object and 30 white cylinders in the StoreCompB object, both of which are from the initial inventory logic just implemented in Process Flow. Note in the Process Flow logic in the right panel, the second token is about to exit.



If you haven't already done so, save the model. Recall, it is good practice to save often.

Now, for each component that might be added in the future, all that needs to be changed are to add a(n):

- column in the Global Table Components with the property values specified,
- row in the Arrivals table in the Initial Inventory Process Flow activity, and
- element in the arrayed Global Variable Storages referencing its Queue.

The above logic only executes once, at the beginning of a simulation whenever a model is Reset. However, it provides the basis for the reorder-point inventory system that is modeled in the next section.

3 MODELING INVENTORY REORDER POLICY

This section defines the reorder-point inventory system logic and describes in detail how to implement it using Process Flow. However, before doing so, a few preparatory changes are made in the 3D model. Why these changes are being made is explained as they are used in the next section.

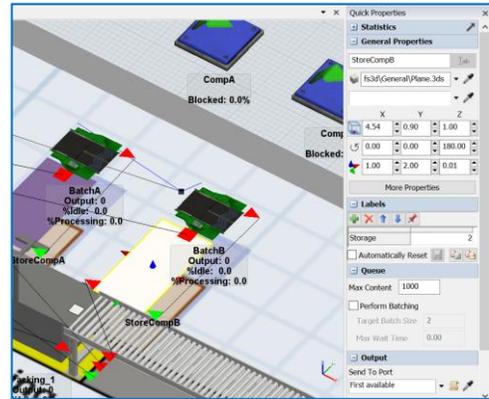


Begin with the basic model from the previous section, named Primer_4-1, and **Save As** Primer_4-2. Basically make a copy of the model so that it can be customized and the original model is retained.

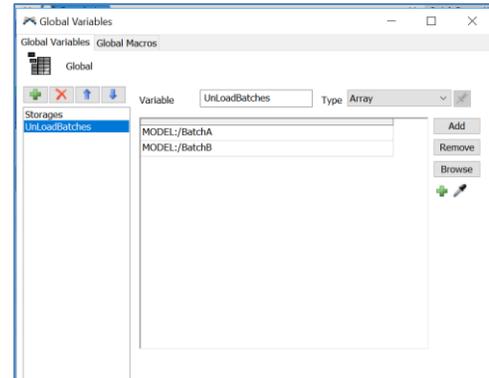
3.1 Changes in 3D objects for use in Process Flow

Before extending the Process Flow logic, the following preparatory changes are made to the model.

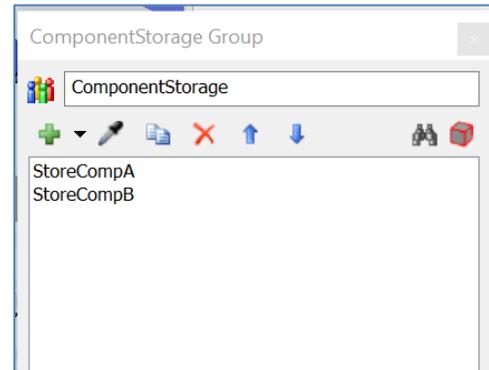
- Create and set a number label for each component’s Queue that indicates what component it stores. Name the label **Storage** and set its value to **1** for StoreCompA and **2** for StoreCompB. Therefore, the item (component) with the label Type=1 is stored in the Queue with the label Storage=1, the component with the label Type=2 is stored in the Queue with the label Storage=2, etc. The label is shown in the Quick Properties view for StoreCompB in the figure to the right.



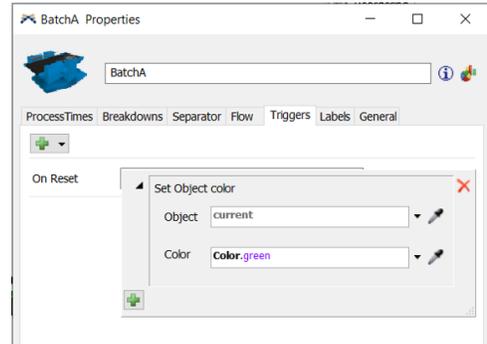
- Create another arrayed Global Variable, named **UnLoadBatches**. As shown in the figure to the right, it contains pointers to the Separator objects, BatchA and BatchB. It is created in the same way as the Global Variable Storages.



- Create a Group, as shown in the figure to the right. As the name indicates, the Group tool is a way to process similar items in a like manner.
 - Create the Group either via the Toolbox or by right-clicking on an object and add it to a current group or add it to a new group. In this case, use the Toolbox to create a Group.
 - Change its name from Group1 to **ComponentStorage**.
 - Add the two component storage Queues, StoreCompA and StoreCompB by using either the Sampler or the + button and then Select Objects menu.



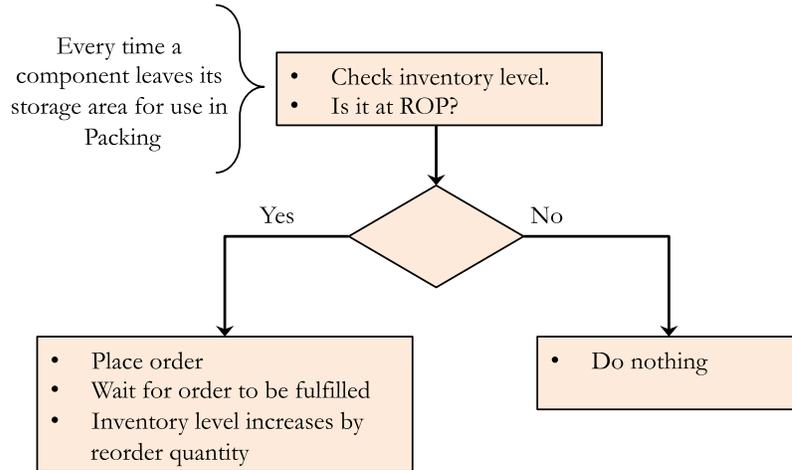
- Set the color of the Separators, where the batches are unloaded to green. Later they will be changed to red when they are in they are awaiting an order. As shown in the figure to the right:
- On the Triggers tab, add an **On Reset** trigger, on both objects BatchA and BatchB.
 - Select the **Set Object Color** trigger option and change the properties as follows:
 - Change the **Object** property value from the default **item** to **current** by using the drop-down menu.
 - **Change the Color** property value from **Color.random()** to **Color.green** by using the drop-down menu.
- Disconnect both component Sources. Since the Sources are being replaced by Process Flow logic, they could be deleted, but for now just disconnect them. As indicated earlier, in future models some of the components may be supplied based on a schedule; if so, the sources would be used. For this primer, all components are supplied via a reorder-point inventory system.



If you haven't already done so, save the model. Recall, it is good practice to save often.

3.2 Implementing reorder-point inventory logic using Process Flow

The basic logic for a reorder-point inventory system is shown in the figure below. The logic is contained in three sections, as denoted by the shaded boxes. Basically, every time a component is used in Packing the inventory level is checked. If the inventory level is not at the reorder point, no action is taken. However, if the inventory level equals the reorder point, the reorder quantity is ordered. There is a delay until the order is fulfilled; then, the level of inventory is increased by the number of items in the reorder quantity.



This logic is now implemented in *FlexSim* via Process Flow in three basic sections: Check for Reorder, Reorder, and Do Not Reorder.

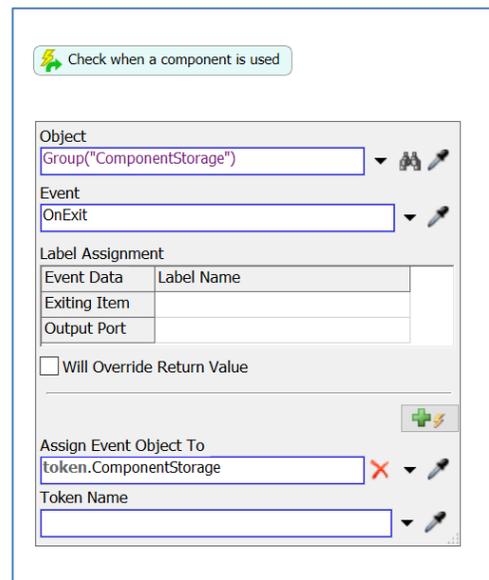
3.3.1 “Check for Reorder” logic

The first set of logic involves three activities: Event-Triggered Source in the Token Creation section of the Library and Assign Labels and Decide in the Basic section of the library.

- Therefore, drag out these three activities onto the Process Flow workspace named Reordering

The first activity, Event-Triggered Source, is used to “listen” for any On Exit events that occur in any of the Queues that are included in the Group named ComponentStorage. This is implemented as described below and as shown in the figure to the right.

- Select the activity, and in the Quick Properties window, rename the activity **Check when a component is used**.
- The activity box can be resized as needed to make the text more readable by dragging one of the handles (small black squares) on the perimeter of the shape.
- The **Object** property is set by selecting **Group** from the drop-down menu, then selecting **ComponentStorage**. *FlexSim* writes the command **Group(“ComponentStorage”)** that obtains the property’s value.
- The **Event** property is **OnExit**, selected from its drop-down menu.



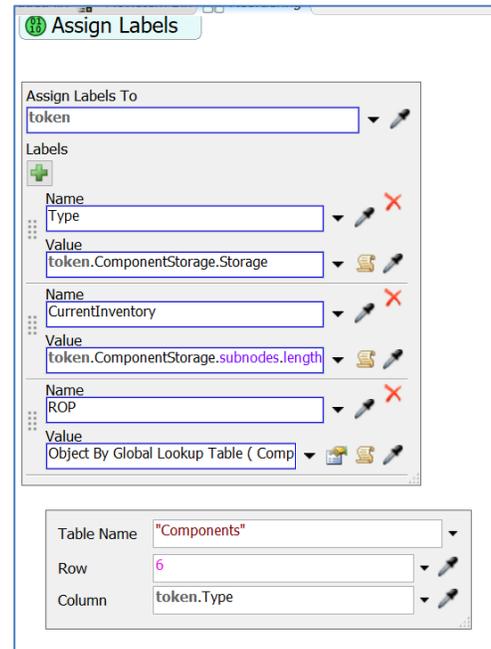
- The value of the property **Assign Event Object To** is set to **token.ComponentStorage**. This action creates a label on the token that points to the Queue object from where the flowitem exited.

The next activity, Assign Labels, is used to obtain some information about the container that just exited the Queue and store it on the token. Each label is added by pressing the +. All of the labels are shown in the figure to the right.

- The first label's **Name** is **Type**, and its **Value** is **token.ComponentStorage.Storage**. This expression can be typed in or the token.ComponentStorage portion of the expression can be obtained from the drop-down list of **Token Labels** and the ".Storage" portion can then be added at the end.

This label contains the value of the label named Storage that is on the Queue object from where the flowitem just exited. Recall that ComponentStorage points to the Queue object where the item/container was stored.

- The next label is given the **Name** of **CurrentInventory** and its **Value** is set to **token.ComponentStorage.subnodes.length**. This label contains the value that is the number of items in the object from where the flowitem just exited (current contents of the object). *subnodes.length* is a *FlexSim* method that provides this information - how many items are currently in an object.

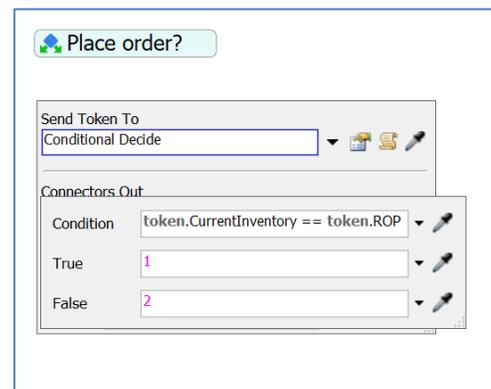


- The final label's **Name** is **ROP** and its **Value** is set by using the drop-down menu options **Table**, then **By Global Table Lookup**. The property values are set as follows and are shown at the bottom of the figure:
 - **Table** is set to **Components** using the drop-down menu.
 - **Row** is changed from the default value **token.labelName** to **6**.
 - **Column** is changed from the default value **1** to **token.Type** using the Labels drop-down option.
 This label stores the reorder-point value that would trigger an order; its value is obtained from the Components table and is based on the type of component.

The final activity in the Check for Reorder section is Decide, which is located in the Basic section of the Process Flow Library. Its properties are set as shown in the figure to the right.

- Name the activity **Place order?**
- Set the property **Send Token To** to **Conditional Decide**.
- Set the **Condition** to:
 - token.CurrentInventory == token.ROP.**
 This can be typed in or the token.CurrentInventory == 1 option can be selected from the **Labels** drop-down menu and then edited; i.e., change **1** to **token.ROP**.

This statement compares the number of items in the Queue where the flowitem/component just exited to the reorder point value for that type of component. If the two values are equal, the condition is True and the token will take branch 1 from the Decide activity and



an order will be placed. If the two values are not equal, the condition is False and the token will take branch 2 from the Decide activity and no further action is taken.

Note the use of == in the Condition expression. Since this is a comparison, the == operator must be used. A single = is an assignment operator that assigns the value of the expression to the right of the = to the variable named on the left side of the =.

Also, the connections from the decide activity to the two alternative blocks will be made after the blocks are created.

As shown in the figure to the right:

- Move the three activities so that they snap together in a sequence, or block.
- Using the Text activity (in the **Display** section of the Process Flow Library), annotate this section of logic to indicate what it does, e.g. **Check for Reorder**.



If you haven't already done so, save the model. Recall, it is good practice to save often.

3.3.2 “Reorder” logic

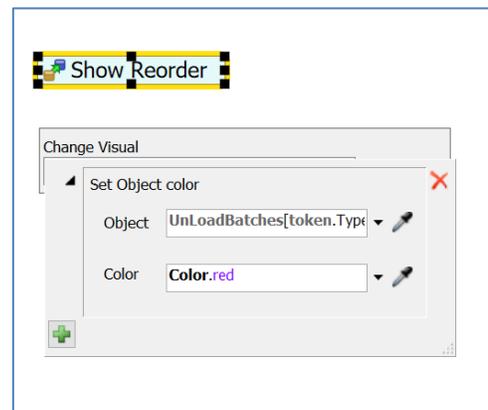
If a component's inventory level is at its reorder point, then:

1. a delay is incurred that simulates the order replacement time, (
2. an order is created, and
3. the resulting order is unloaded by the finish operator.

The logic below implements this and changes the color of the batch area to red to indicate that an order has been placed. The color is reset to green once the order is fulfilled.

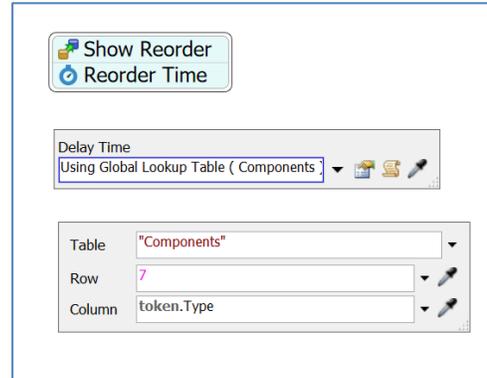
The Change Visual activity sets the appropriate Separator's (BatchA or BatchB) color to red to indicate that its component has been ordered. This is shown in the figure to the right and described below.

- Select the Change Visual activity from the Visual section of the Process Flow Library.
- Name the activity **Show Reorder**.
- For the **Change Visual** property, select using the + button the **Set Object Color** option, then
 - Set the **Object** property to **UnLoadBatches[token.Type]**.
As you type this expression, *FlexSim* provides a list of possible values, one of which is the Global Variable named UnLoadBatches, which can then be selected. Then after the [token. is entered, then token.Type can be selected from the list of suggested items on the drop-down menu.
 - This value is an Object property that points to a Separator objects that is stored in the arrayed Global Variable named UnLoadBatches. The value is based on the component type, which is stored in the token's label named Type.
 - Set the **Color** property to **Color.red** which results from choosing Color.red on the drop-down menu.



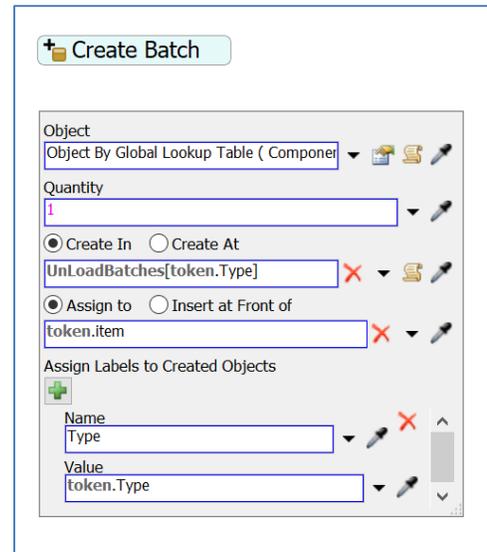
The **Delay** activity (from the Basic section of the library) is used to set the reorder time. The token is delayed by the value stored in the Global Table Components. As shown in the figure to the right:

- Name the activity **Reorder Time**.
- Set the **Delay Time** using the drop-down menu option **By Global Table Lookup** with the **Table** parameter set by selecting “Components” from the drop-down list of model tables, the **Row** parameter value set to 7, and the **Column** parameter set to the value stored in the token’s label that is named Type, i.e. **token.Type**. Thus, the reorder time is the value stored in row 7 of the Global Table Components and in the column that corresponds to the value of the token’s Type label.



The **Create Object** activity (in the Objects section of the library) creates a flowitem in the appropriate Separator that represents a component order. That flowitem will later be split into the number that is specified as the reorder quantity. This is very similar to the Create Object activity in the Initial Inventory section of the logic. As shown in the figure to the right:

- Name the activity **Create Batch**.
- The **Object** property is set using the **Flowitem by Global Table Lookup** drop-down menu option. The token creates a flowitem of the class that corresponds to its location (its number) in the list of the flowitems in the FlowItem Bin. In this case CompA and CompB are the 10th and 11th items in the list, respectively. These reference values are stored in the Components table. Therefore:
 - **Table** is set to **Components** from the list of Global Tables.
 - **Row** is set to 8.
 - **Column** is set to **token.Type**. (This value can be selected from list of tokens in the drop-down list **Labels**.) The column in the table depends on the value of the token’s Type label (type of component).
- The **Quantity** value is 1 (the default value). This activity only creates an order. The order size is handled by the Separator.
- The value for **Create In** is **UnLoadBatches[token.Type]**. The flowitem is created in the object referred to in the arrayed Global Variable UnLoadBatches and depends on the value of the token’s Type label (type of component).
- One label on the flowitem is created using the + button in the **Assign Labels to Created Objects** section of the interface.
 - Change the label’s **Name** from labelName to **Type**.
 - Change the **Value** property from 0.0 to **token.Type** by selecting **Token Label** from the drop-down menu and then selecting **Type**. In this case, the value is either 1 or 2.



Another Change Visual activity is used to reset the appropriate Separator’s (BatchA or BatchB) color, i.e., from red to green in order to indicate that its component’s order has been delivered. The activity is the same as the Show Reorder activity.

- Select the activity, Change Visual from the Visual section of the Process Flow library.
- The activity name is **Show Delivery**.
- Select Set Object Color and the properties to:
 - **Object** to **UnLoadBatches[token.Type]**.
 - **Color** to **Color.green** from the drop-down list of colors.
- Drag out a Sink activity (from the Basic section of the library).
- Ensure all five of the activities created above are snapped together to form a “block” of activities.

3.3.3 “No Reorder” logic

This branch includes the activities that are executed by a token when the use of a component in the packing area does not trigger a reorder event. The branch could be composed of just a Sink, but an arbitrary 10-minute delay is added so the user can see the token route to this branch. There is no effect on the operation or performance of the model.

- Drag out a Delay activity and set the **Delay Time** property to **10**.
- Drag out a Sink activity and connect it to the Delay activity created above.

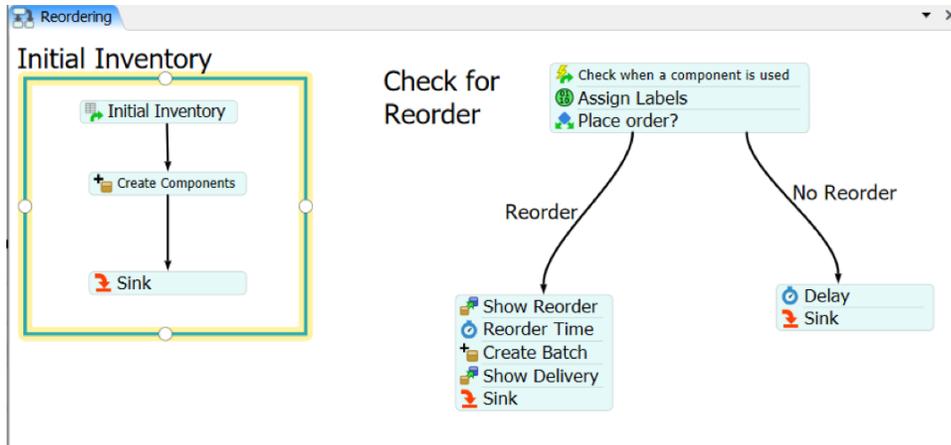
The “Reorder” and “No Reorder “ blocks of activities are alternative responses to the “Place order?” Decide activity. Therefore, they must be connected to the activity, but the “Check to Reorder” block must be connected first.

- Click the bottom of the “Check to Reorder“ block (the last activity in the block is “Place order?”). The cursor changes to a chain and as the mouse is moved an arrowed line connector appears. Complete the link by clicking on any side of the “Reorder block.”
- Repeat the above step for the “No Reorder” block of activities.

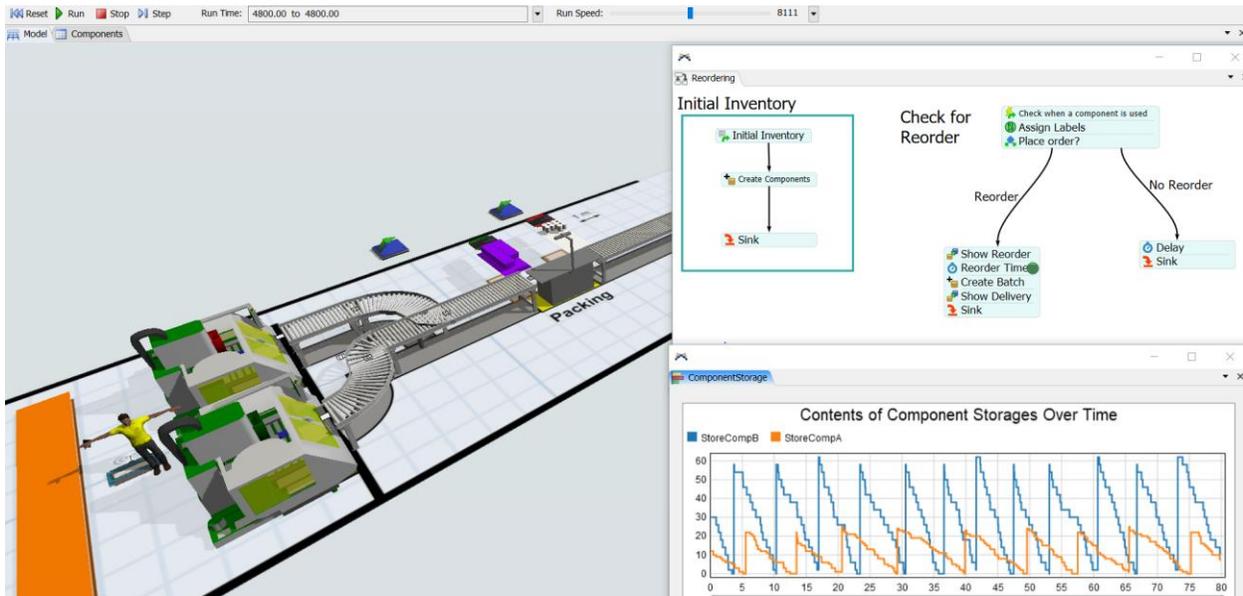
Notice the 1 and 2 labels at the bottom of “Check for Reorder” block. If the condition in the “Place Order?” activity is True (current inventory is equal to the reorder point for that component), its value is 1 and the “Reorder” block of activities are executed. Similarly, if the condition in the “Place Order?” activity is False, its value is 2 and the “No Reorder “ block of activities are executed.

- Double-click the arrows and use the text box to label the connectors – **Reorder** and **No Reorder**, respectively. The size, color, and font of the text can be edited through the Quick Properties interface. Also, the text boxes can be moved for better placement.

The final Process Flow logic should look like the figure below.



The figure below is a screenshot of the current model running near the end of the prescribed Run Time. It includes the Process Flow logic and a Dashboard showing the contents of the two component Queues over time.



The Process Flow logic can easily be resized to fit the window size by right-clicking on the Process Flow workspace, selecting **View**, then selecting **Size to Fit**.



If you haven't already done so, save the model. Recall, it is good practice to save often.

4 CUSTOM TASK SEQUENCES IN PROCESS FLOW

This section provides a second extension of the primer model using Process Flow. In this case, a custom task sequence is developed using Process Flow to add a task to the finish operator's basic activity of moving containers from storage to finish machines for processing. The new task is that the operator must log information at a kiosk after a container is loaded onto a finish machine.

Prior to developing the logic in Process Flow, some modifications are made to the base model. Also, a simple study model is used to understand how to customize task sequences before implementing the logic in the main model.

4.1 Base model modifications

Some of the modifications in this section are to set up the change in tasks on the finish operator. Other modifications provide an opportunity to review and be introduced to new features in the 3D portion of *FlexSim*, such as:

- increase the production rate to the finish area,
- change from the A* algorithm back to using path networks to control the finish operator's travel paths,
- integrate the kiosk into the network, and
- hide the underlying layout that was used to build the model.



Begin with the basic model from the previous section, named Primer_4-2, and **Save As** Primer_4-3a. Basically make a copy of the model so that it can be customized and the original model is retained.

- Be sure the queue ContainersStorage's **Maximum Content** property is set to 5 (based on the experimentation in Section 4 of Part 3).

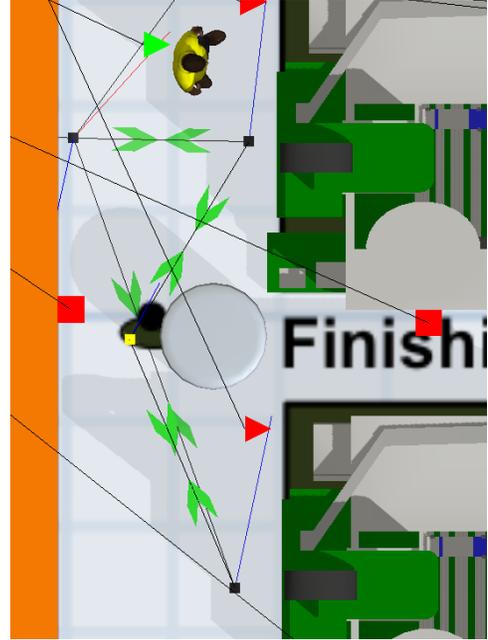
Based on running the simulation model, the utilization of the finish operator is only about 20% and the finish machines' utilizations are only about 60-65%. Therefore, it has been decided to increase the production rate by 25%.

For the production rate to increase by 25%, the average time between arrivals must decrease by 25%. The current inter-arrival time distribution is **triangular(10, 35, 15)**, which has an average of 20 minutes. Therefore, the mean needs to decrease to 15 minutes. Assume the revised distribution is **triangular(5, 30, 10)**, which has an average time between arrivals of 15 minutes.

- Update the triangular distribution's parameters in the Source, ContainersArrive.
- Confirm that the planned product mix is 20% for Type 1, 30% for Type 2, and 50% for Type 3 (recall, these are in the Global Table ProductMix).

Another task is going to be added to the operator – after the finish operator loads a container onto a finish machine, information needs to be logged at a kiosk that is located between the two machines. This task is added in the next section, but the kiosk object is added here, as described below and as shown in the figure to the right. It may be easier to add the object in planar view. (Recall, this is done by right-clicking on the modeling surface, selecting **View**, then **Reset View**.)

- To represent the kiosk, use the default Shape object, in the Visual section of the object Library. Retain its default size, a 1-meter high cylindrical object that is 1 meter in diameter.
- Name the object **Kiosk**.
- Position it between the two machines, say at $x = -8.5$, $y = -0.25$, and $z = 0$.
- Add the kiosk object as a barrier in the A* Navigator object in the **Members** portion of the Setup tab. Select FR Members, then use the + button to add the Kiosk, which is in the Visual Tools section of the object list.
- Hide the A* components, since a path network will be used. To do so, on the Visual tab of the A*Navigator, be sure all boxes are unchecked except for **Show Barriers** so that only the walls are displayed. If A* is used, the objects are still considered barriers even if hidden; not showing them just hides the blue shadowing under the object that indicates it is a barrier.

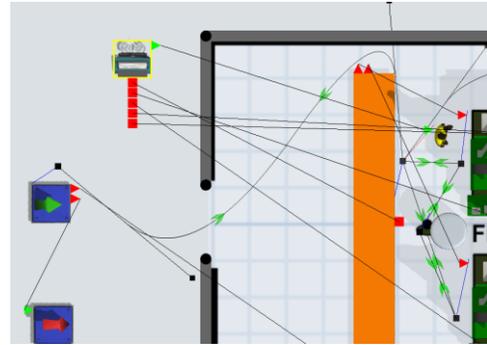


In this version of the model the path network will be used rather than the A* algorithm to control the finish operator's travel. Once the operator is connected to the network, it takes precedence over A*.

Make the following modifications to the path network.

- It may be easier if the Dispatcher object is moved out of the way; its placement does not affect the model.
- Also, it will be easier to do the remaining steps if the network is made visible again. Therefore, right click on the visible Network Node near the Queue ContainerStorage, then select **Network View Mode**, and then **Show All**.
- Add the operator to the network with an A-connection to the node near the Queue, ContainersStorage. (Recall, after the connection, a red line should show the connection between the two objects.)
- Change the Operator's property **Use navigator for offset travel** to **Travel offsets for load/unload tasks**.
- Integrate the kiosk into the path network.
 - Add a Network Node for the kiosk; name it **nn_Kiosk**.
 - Connect the Node to the kiosk object.
 - Disconnect the path between the Nodes at the finish machines, **nn_FinMach_1** and **nn_FinMach_2**.
 - Connect **nn_Kiosk** to **nn_FinMach_1** and connect **nn_Kiosk** to **nn_FinMach_2**, as shown in the figure above.

- Right-click on one of the green arrows on the network edge that leads to the Break Area. From the popup menu, change the edge property to **Curved**. Adjust the path around the Queue and through the door, as shown in the figure to the right, by dragging the handles. This is the same process that was used earlier to create the path to the packing area.



Additional nodes could be used to form the path, but the edge between two nodes is a Bézier curve and is quite flexible.

- Hide all of the network nodes except the one by ContainersStorage. To do so, as before, right-click on that node, select **Network View Mode**, then select **None**. All nodes and edges will be hidden except for the one by the Queue. The network can be viewed again by following the same process except selecting either **Edges** (which only shows the edges; no nodes are shown) or **Show All** (all nodes and edges are displayed).
- Hide the layout on which the model was built. To do so, open the Tree and then double-click on the Layout object to open the user interface. In the Flags section of the General tab, uncheck the **Show 3D Shape** box. This way the layout can be displayed again by reversing this process.



If you haven't already done so, save the model. Recall, it is good practice to save often.

4.2 Simple study model of task sequences in Process Flow

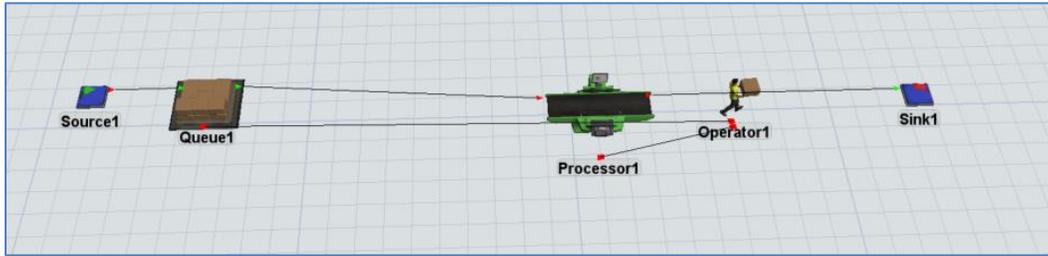
Before adding the information-logging task to the finish operator in the main model, develop a simple study model in order to understand some basic concepts of working with Task Executors. As has been stressed in this primer, it is good modeling practice to test out new concepts and more advanced logic in a study model before implementing it in the primary model. This allows the modeler to focus on the salient aspects of the change.



Start with a new FlexSim model, accept the default model units, and name it Simple TE Process Flow.

Begin with the most basic model, as shown below, with all default values. The operator is used to transport items from the Queue to the Processor and from the Processor to the Sink.

- Drag out all of the required objects and connect them as shown in the figure below.
 - In addition to A-connecting the Fixed Resource object, be sure to S-connect (Center Connect) the Queue and Operator and the Processor and Operator.
 - Also, be sure to check the **Use Transport** box on the Flow tab on both the Queue and Processor.



For comparison, two similar models are considered. To facilitate this, we will make a copy of the above model. This is a convenient way to duplicate sets of objects.

- Select all of the objects – hold down the Shift key and use the left mouse button to encompass all of the objects; this results in a gray box that contains the objects. When the mouse button is released, all of the objects should be selected; i.e., the outline of all objects should be red.
- Copy, using Ctrl-C, the selected objects, then click on the modeling surface somewhere below the current model, and using Ctrl V, paste the set of objects. The selected set can be moved as a group to position them on the modeling surface. Once set, unselect all of the objects by holding down the Shift key and clicking with the mouse anywhere on the modeling surface (not on any objects).

The two models in the file are completely independent since there are no flows, resources, events, or activities that are shared. In this exercise the upper model will remain as is; it is there just for comparison. The lower model will be modified to use Process Flow to control some of the operator's tasks. The models can be used to compare task sequences using the default 3D approach and customizing them through Process Flow.

A Shape object, like the kiosk in the main model, is used in this example. The operator will carry each flowitem to it once processing is finished at the Processor. At the Shape object, the operator and flowitem are delayed for additional processing. After the delay, the operator carries the flowitem to the sink and unloads it.

Note that the transport from the Source to the Processor remains unchanged. It is managed by the 3D model (in the Flow tab of the Source) and not by Process Flow logic. There is no need to modify it since the default logic is fine.

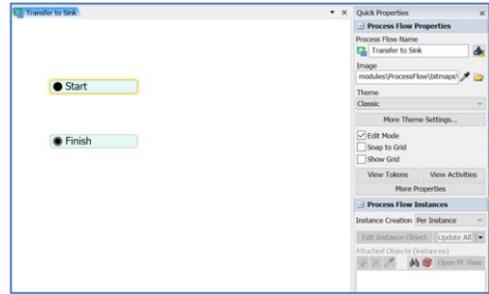


If you haven't already done so, save the model. Recall, it is good practice to save often.

- Add a Shape object, from the Visual section of the Library, anywhere in the second model, somewhere between the Processor and Sink. Again, this is like the kiosk in the main model.

In order to have the operator travel to and be delayed at the Shape object before being unloaded at the Sink, Process Flow is used to replace the basic task sequence for transport with a custom task sequence.

Start the Process Flow logic as shown in the figure to the right and as described below.

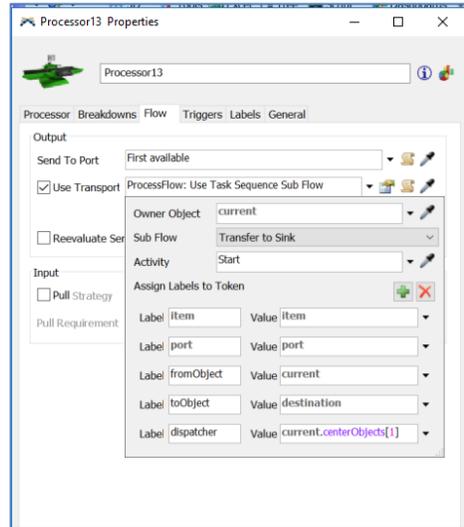


- From the Process Flow icon on the Main Menu, or via the Toolbox, add a **Sub Flow** to the model.
- In the **Process Flow Properties** section of the Quick Properties window, change the **Process Flow Name** from SubFlow to **Transport to Sink**.
- In the **Process Flow Instances** section of the Quick Properties window, change the **Instance Creation** property value from **Global** to **Per Instance**.
- From the Sub Flow section of the Process Flow Library, drag out Start and Finish activities onto the Process Flow workspace.

FlexSim's default transport logic using a Task Executer, as was described in Section 3 of Part 2, has the Operator perform the following uninterrupted sequence of tasks. In this primer, this set of four tasks is referred to as the Transport Task Sequence.

1. Travel to the object where the flowitem is located; the object is referred to as fromObject.
2. Load the flowitem from the fromObject.
3. Travel with the flowitem to its destination object that is referred to as toObject.
4. Unload the flowitem to the toObject.

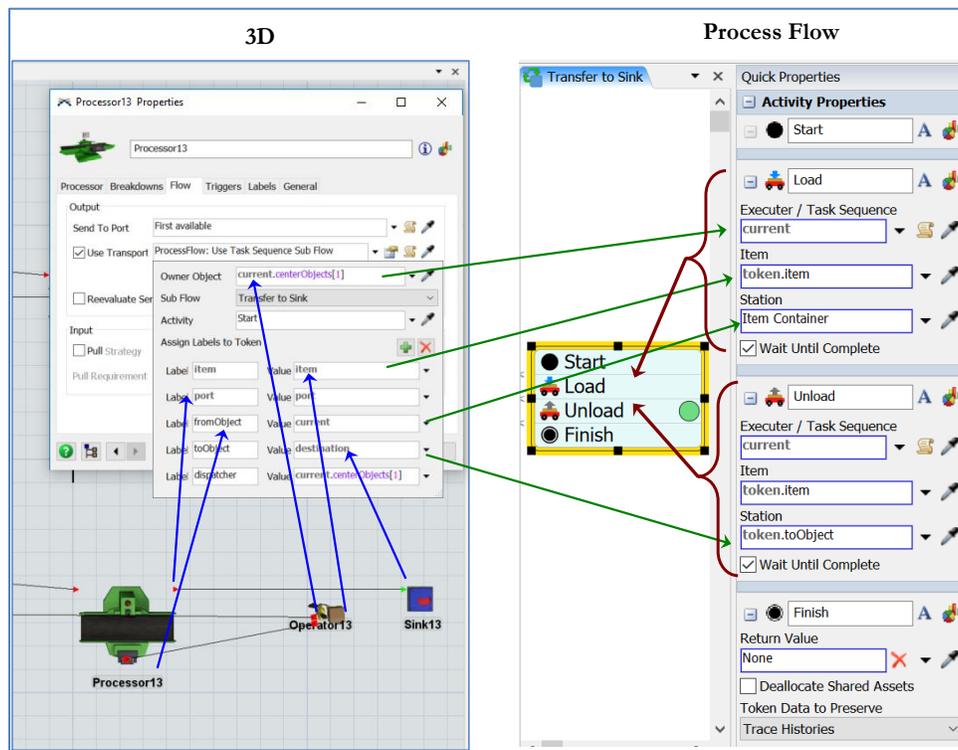
These four tasks will now be handled by Process Flow so that they can be easily modified and customized. To pass the responsibility for transporting a flowitem from the Processor, the **Use Transport** property on the Flow tab of the Processor is changed, as described below and shown in the figure to the right.



- On the Processor's Flow tab, the Use Transport box should already be checked. Change the default `current.centerObjects[1]` to **ProcessFlow: Use Task Sequence Sub Flow** via its drop-down menu.
- Change the resulting interface as follows:
 - Change the **Owner Object** property from **current** to `current.centerObjects[1]` by selecting it from the **Connected Objects** option on the property's drop-down menu.
 - All of the remaining properties are the default values. However, be sure the **SubFlow** property is **Transfer to Sink** (the name of the newly created Process Flow Sub Flow) and the **Activity** property is **Start**. The remaining items in the interface is information that is sent to Process Flow about the Processor object and the flowitem that needs to be transported.

As shown in the following figure, information needed to perform the transport in the 3D model is captured in the Processor's Flow tab and sent to Process Flow. The sending of this information creates a token in Process Flow and that token contains all of this information in its labels. The token flows through a set of activities that models the transport operation. The token's labels are defined as:

- token.item flowitem that needs to be transported.
- token.fromObject object that the flowitem is transported from; i.e., Processor 13 for the case shown in the figure.
- token.toObject object that the flowitem is transported to; i.e., the Sink for the case shown in the figure.
- current the task executer that will perform the transport. It is the object that is connected to the center port of Processor13, current.centerobject[1]; i.e., Operator13 for the case shown in the figure.



As shown in the figure above, the basic transport is accomplished in Process Flow by using two activities Load and Unload, both found in the Task Sequences section of the Activity Library. In the 3D portion of the figure, the Operator is carrying the item to the Sink and correspondingly the token in Process Flow is in the Unload activity.

The Load activity is actually a small task sequence, carried out by, in this case, Operator13. The task sequence contains the first two tasks in the Transport Task Sequence that was defined above – travel to the fromObject and load the item. Similarly, the Unload activity is also a task sequence, again carried out by Operator13 in this case, that contains the last two tasks in the Transport Task Sequence defined above – travel to the toObject and unload the item. In both models, the task sequence is considered complete with

the unloading of the flowitem at the Sink; therefore, the Operator waits at the Sink for the next task sequence to be assigned.

Add the Load and Unload activities in the Sub Flow and modify their properties as shown in the figure above.

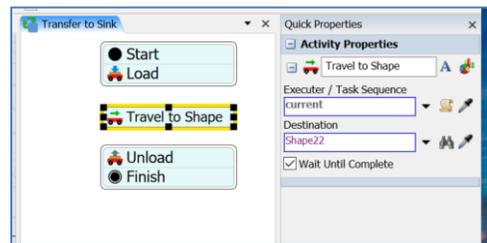
- Drag a Load activity from the Task Sequences section of the Process Flow Library.
 - Change the **Executer / Task Sequence** property to **current** by selecting **current (Instance Object)** from the drop-down menu.
- Similar to the previous step, drag an Unload activity from the Task Sequences section of the Process Flow Library.
 - Change the **Executer / Task Sequence** property to **current** by selecting **current (Instance Object)** from the drop-down menu.
 - Change the **Station** value from **token.destination** to **token.toObject**.
- Join the four activities to form a logic block that is the sequence: Start, Load, Unload, Finish.
- **Reset** and **Run** the model. Confirm that both model segments operate similarly. One model uses *FlexSim's* default task sequence for transporting items by a task executer; the other model uses a custom task sequence created in Process Flow.



If you haven't already done so, save the model. Recall, it is good practice to save often.

The new task, to travel to the Shape object, is added as shown in the figure to the right and described below.

- Select the block of activities and click on the “scissors” symbol between the Load and Unload activities in order to break the block so a Travel activity can be inserted.
- Drag a Travel activity to Process Flow from the Task Sequences part of the Library, and:
 - Change the **Executer / Task Sequence** property to **current**. As before, select **current (Instance Object)** from the drop-down menu.
 - Change Destination property to the name of the Shape object, Shape22 in this case. (Note that the number in the object names may be different since in this study model, the default *FlexSim*-generated names are used.) The Shape object can be selected either by:
 1. the property's drop-down menu **FlexSimEventHandler** and selecting the desired object or
 2. clicking on the object in 3D using the property's Sampler tool.
- Change the name of the task from the default Travel to **Travel to Shape**.
- Insert the Travel activity into the logic block between Load and Unload. Snap all activities together to form one block.

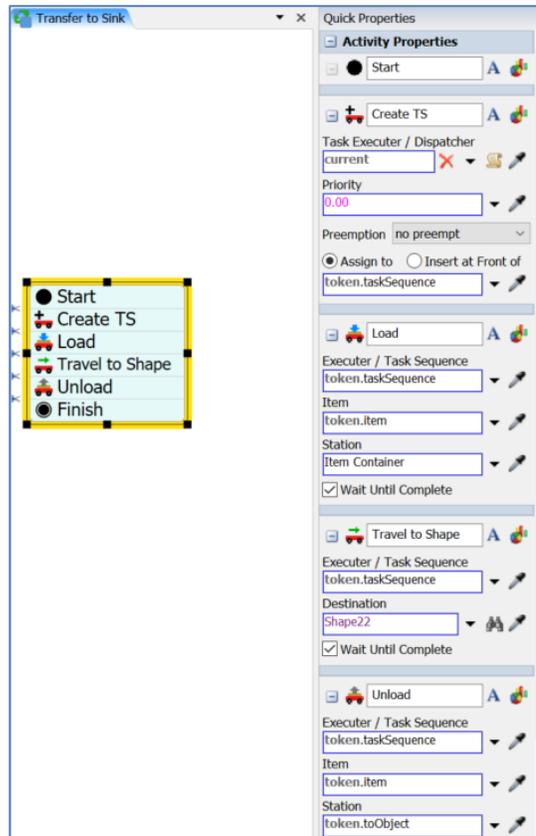


- **Save, Reset, and Run** the model. Check to be sure to model works as planned. Note that it does **not** work as planned! Here’s what happens after an item is finished at the Processor:
 1. Operator travels to the Processor and loads an item → Correct.
 2. Operator travels to the Shape object → Correct.
 3. Operator travels to the Source and loads another object; the operator now has two objects → Incorrect.
 4. Operator travels to the Processor and unloads the item from the Source → Correct.
 5. Operator travels to the Sink and unloads the object → Correct.

The problem is there are now four task sequences: the default Transport Task Sequence (from the Source to the Processor) plus the three Process Flow sequences: , Load (after Processing), Travel (to Shape), and Unload (at Sink). As flowitems enter the system and finish processing their resulting tasks are sent to the Task Executer and are queued in a First-In, First-Out manner. The Operator is processing tasks in the order in which they are received; however, this results in an operation that is not what is desired.

To remedy the problem, all three Process Flow task sequences need to be processed as one and not as separate task sequences. The Process Flow logic is modified as shown in the figure to the right and described below.

- “Cut” the logic block and add the Create Task Sequence activity (name is Create TS) between the Start and Load activities. Rejoin the activities, again into one block.
- Change the Create TS activity’s **Task Executer / Dispatcher** property to **current**. Reference to the task sequence is stored in the token’s label named **taskSequence** (token.taskSequence).
- Change the **Task Executer / Dispatcher** property in all of the remaining activities (Load, Travel, and Unload) from **current** to **token.taskSequence**. Each of these can be selected from the drop-down menu **Token Label**.



- **Save, Reset, and Run** the model. Verify that the model works as planned. It should! The model is correct, if the Operator transports only one flowitem at a time; and, after the operation at the Processor is complete, the Operator should take the item to the Shape object and then to the Sink, where it is unloaded.



If you haven’t already done so, save the model. Recall, it is good practice to save often.

4.3 Custom task sequences in the primer model

The concepts developed in the previous section are now applied to the main primer model. The Process Flow logic is very similar to that described in the simple study model.



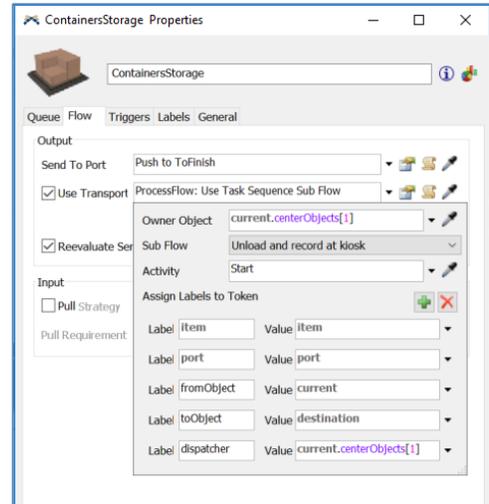
Begin with the basic model named Primer_4-3a, and **Save As** Primer_4-3b. Basically make a copy of the model so that it can be customized and the original model is retained.

Start the Process Flow logic as described in the previous section.

- From the Process Flow icon on the Main Menu, or via the Toolbox, add a **Sub Flow** Process Flow logic to the model.
- In the Quick Properties interface, change the **Process Flow Name** from SubFlow to **Unload and record at kiosk**.
- In the Quick Properties interface, change the **Instance Creation** property value from **Global** to **Per Instance**.
- From the Sub Flow section of the Process Flow Activity Library, drag out Start and Finish activities.

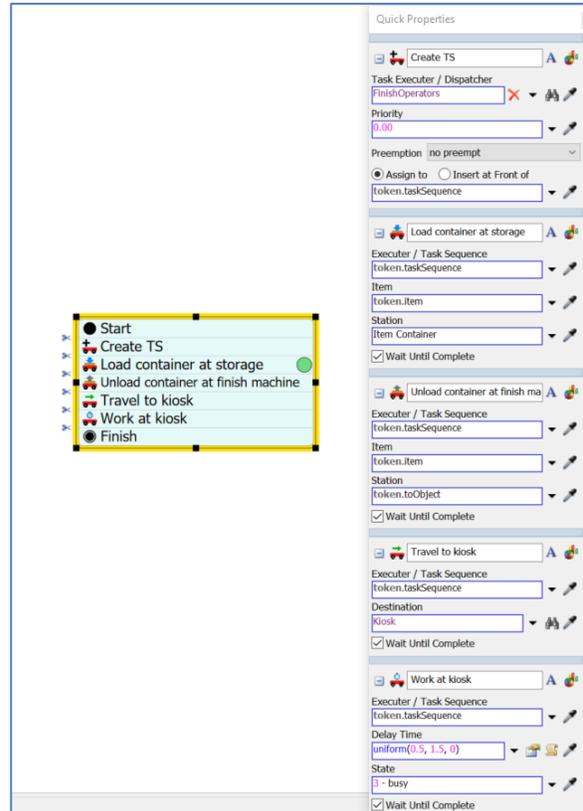
The responsibility for defining the tasks to transport containers from the Queue ContainersStorage to a finish machine is changed from the 3D object to Process Flow logic so that the finish operator can travel to the kiosk after unloading a container at the finish machine.

- On the Queue's Flow tab, the Use Transport box should already be checked. Change the default **current.centerObjects[1]** to **ProcessFlow: Use Task Sequence Sub Flow** via its drop-down menu.
- Change the properties on the resulting interface as follows:
 - Change the **Owner Object** property from **current** to **current.centerObjects[1]** by selecting it from the **Connected Objects** option on the property's drop-down menu.
 - All of the remaining properties are the default values. However, be sure the **SubFlow** property is **Unload and record at kiosk** (the name of the newly created Process Flow Sub Flow) and the **Activity** property is **Start**. The remaining items in the interface are information that is sent to Process Flow about the Queue object and the flowitem that needs to be transported.



Define the activities for the Process Flow and as each activity is defined add it to the logic block, as shown in the figure to the right.

- So that all of the tasks are performed as a group, start the logic flow with the Create Task Sequence activity.
 - Its default name, **Create TS**, is okay.
 - Change the **Task Executer / Dispatcher** property to the operator's Dispatcher object named **FinishOperators**. This can be selected from the Dispatcher section of the drop-down menu.
 - The default **Assign To** label name – **token.taskSequence** – is okay.
- Use the Load activity for the Operator to travel to the Queue and pick up a container.
 - Change its name to **Load container at storage**.
 - Change the **Executer / Task Sequence** property to **token.taskSequence**. from the Token Label section of the drop-down menu.
 - Change the **Station** property to **token.fromObject**. Recall this is passed from the Queue when requesting a travel resource.
- 1. Use the Unload activity for the Operator to travel, with the flowitem, to a finish machine and unload the container.
 - Change its name to **Unload container at finish machine**.
 - As in the previous activity, change the **Executer / Task Sequence** property to **token.taskSequence**.
 - Change the **Station** property to **token.toObject**
- Since longer, more descriptive activity names are used, it is advised that the size of the activities be changed for readability. To do so, select a side handle on the logic block and drag it horizontally to the desired width. All subsequent activities that are added to the block will be resized to this larger width.
- Use the Travel activity for the finish operator to travel to the kiosk. Change its properties as follows:
 - Change its name to **Travel to kiosk**.
 - As in the previous activity, change the **Executer / Task Sequence** property to **token.taskSequence**
 - Change the **Destination** property to **Kiosk** either by the drop-down menu option **FlexSimEventHandler** or by using the Sampler tool.
- Use the Delay activity for the finish operator to spend time at the kiosk. Be sure to use the Delay activity in the Task Sequences section of the Library and not the one in the Basic section. Change the activity's properties as follows.
 - Change the Delay activity's name to **Work at kiosk**.
 - As in the previous activity, change the **Executer / Task Sequence** property to the token label **token.taskSequence**



- Change the **Delay Time** property from the default value of 10.0 to **Statistical Distribution**, then **Uniform** by the drop-down menu. In the resulting interface, set the distribution's parameters: **Minimum** to **0.5** and **Maximum** to **1.5**. Therefore, the operator is delayed at the kiosk for a randomly-generated amount of time that is between 0.5 and 1.5 minutes. Note, the average delay time is one minute.

- **Reset** and **Run** the model. Verify that the finish operator travels to and is delayed at the kiosk each time after a finish machine is loaded, as shown in the figure to the right.



If you haven't already done so, save the model. Recall, it is good practice to save often.



A close examination of what happens during the simulation, which can be noted early on in the simulation via the **Step** feature on the Model Execution Toolbar, is that the Operator unloads a container at a finish machine, goes to the kiosk, then returns to the machine if a setup is required. This is because there are two task sequences at play here:

1. the load, unload, and kiosk work, as defined in the section above in Process Flow and
2. the setup operation using the Operator, if needed. This may be the way the real system works, or it may not.

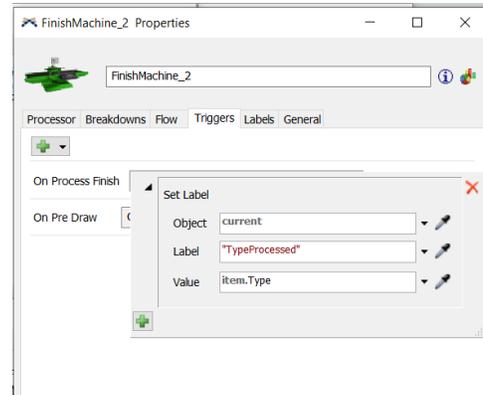
In the case of this example, this is not the way the system works. The Operator should load a flowitem at the Queue, travel to the desired finish machine, unload the item, perform the setup operation, if needed, and then travel to, and work at, the kiosk. Therefore, the Process Flow logic needs to be modified to include the setup operation; i.e., the setup logic on the two Processors (finish machines) that was included in the model earlier in the primer via drop-down menu options needs to be replaced by Process Flow logic.



Begin with the basic model named Primer_4-3b, and **Save As** Primer_4-3c. Basically make a copy of the model so that it can be customized and the original model is retained.

Prior to modifying the Process Flow logic, a new object label needs to be added to both Processors that tracks the type of the flowitem/container that was most recently processed. This is used to determine if a setup is required or not - no setup is required if an arriving item is of the same type as the one that was just processed.

- On the Labels tab of each Processor (FinishMach_1 and FinishMach_2), use the + button to **Add Number Label**.
 - Change the name from labelName to **TypeProcessed**.
 - Check the box at the bottom of the interface **Automatically Reset Labels**. Whenever, the model is Reset, the label value will be set to 0. This will ensure that a setup always occurs when a simulation starts because the arriving item Type value will never be 0. (In this example, the value will be 1, 2, or 3.)
- On the Triggers tab of each Processor, use the + button to add an **On Process Finish** trigger.
 - Using the + button on the trigger, select **Data** and then **Set Label** on the drop-down menus.
 - Set the property values on the **Set Label** interface as follows and as shown in the figure to the right.
 - Change the **Object** property from the default item value to **current** using the drop-down menu. That is, set the label value on the machine, not on the flowitem.
 - Change the **Label** property from the default “Type” value to “TypeProcessed” using the drop-down menu.
 - Change the **Value** property from the default value to **item.Type** by selecting the value from the **Labels** section of the drop-down menu. This sets the value of the Processor’s label named TypeProcessed to the value of the item’s label named Type that just finished processing.



As was defined earlier in the primer, each finish machine (Processor) handles the logic for:

1. determining if a container requires a setup or not,
2. if a setup is required, determine the setup time, and
3. also if a setup is required, obtain an Operator to perform the setup.

All of this logic now needs to be removed from the 3D objects because it will now be handled through Process Flow logic.

- On the Processor tab of each Processor:
 - Change the **Setup Time** property’s value that had been added earlier in the primer to **0** by selecting **No Setup Time (0)** from the drop-down menu.
 - Uncheck the **Use Operator(s) for Setup** box.

Define and incorporate the setup logic from the Processors into the recently created Sub Flow named **Unload and record at kiosk**.

- Modify the name of the Processor Flow to reflect the new logic; i.e., **Unload, setup, and record at kiosk**.
- Split the logic block after the activity **Load container at storage**.
- Add an Assign activity from the Basic section of the Process Flow Library. Its properties are set as described below and as shown in the figure to the right.

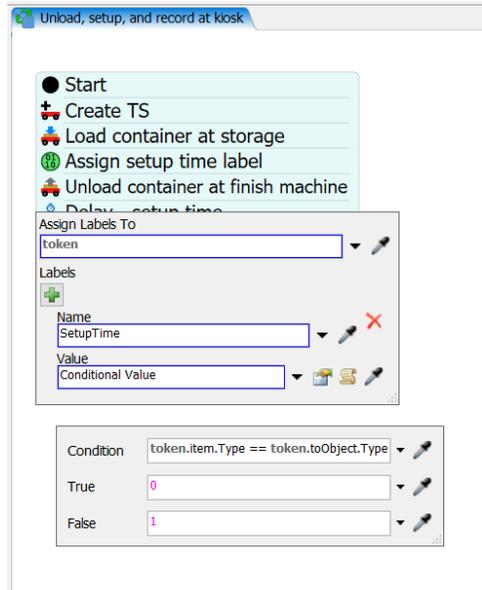
- Name the activity **Assign setup time label**.
- Change the token label **Name** to **SetupTime**.
- Set the **Value** property to **Conditional Value** and change the property values on the resulting interface.
- Set the **Condition** property value to:

token.item.Type == token.toObject.TypeProcessed.

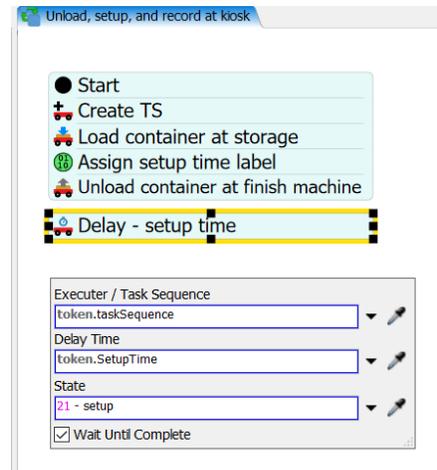
This statement compares the type of container/item that needs to be moved from the Queue to the type of container that was last processed at the finish machine where the container/item is being moved to.

If the two values are equal, then the value specified in the value of **True** property is the setup time. Conversely, if the values are not equal, the setup time is the value specified in the **False** property.

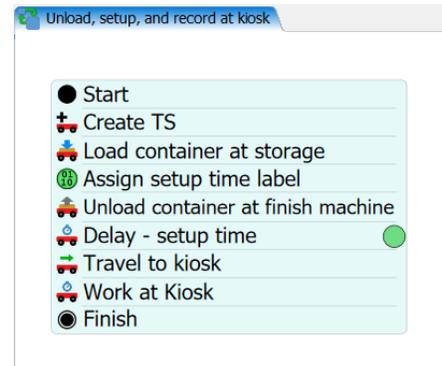
- Set the value of the **True** property to **0** (no setup time is needed since the items are of the same type) and set the value of the **False** property to **2**. (setup time is 2 minutes).



- Add a Delay activity after the Unload activity, as shown in the figure to the right and as described below.
 - The value of the **Executer / Task Sequence** property is the token label **token.taskSequence**.
 - The value of the **Delay Time** property is the value of the token label that was set earlier in the Assign activity, **token.SetupTime**.
 - The value of the **State** property is **21 – setup**, selected from the dropdown menu.



The final Subflow Process Flow logic is shown in the figure to the right. It controls the Operator for the set of tasks that include: transporting flowitems from the container Queue to the finish machine Processors, performing a setup operation, if needed, and performing work at the kiosk.



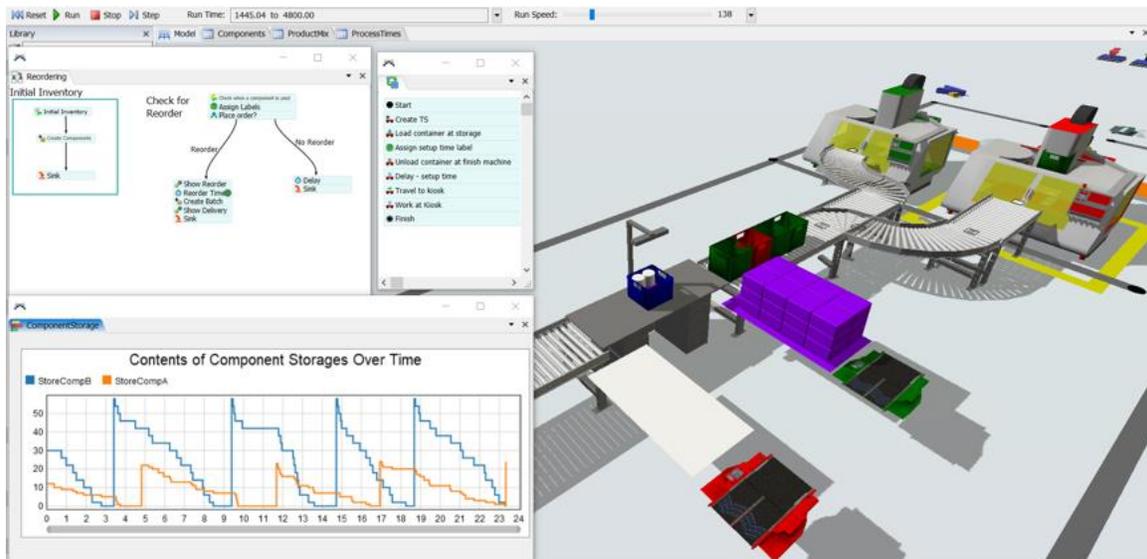


If you haven't already done so, save the model. Recall, it is good practice to save often.

- **Reset and Run** the model. Verify that the finish operator moves containers from the ContainerStorage object to the finish machines, performs a setup if needed, and then travels to and works at the kiosk before undertaking another task.

A screenshot of the final model as it is running is shown in the figure below; it includes:

- 3D view of the finishing and packing areas with a variety of fixed and mobile resources, downtimes, etc.
- Two applications of Process Flow - reorder-point inventory management and custom task sequence.
- Example Dashboard output, inventory levels of the two components; this clearly shows the dynamics of the system and the basic reorder-point means of managing inventory.



Note that this model can now be easily scaled up – add more finish machines, packing stations, operators, etc. These can be added by mostly copying and pasting the objects.

Also, if the modeling started with many finish machines or packing stations, then each change in the model would require the change be made in each of the objects. This is not only tedious, but subjects the model to potential errors. Therefore, *it is always good modeling practice to start small, add complexity as needed in small steps, test and verify, then scale up.*

This page is intentionally blank.

PART 5 – WAREHOUSING, ORDER PROCESS AND ADDITIONAL FEATURES

The final part of the primer extends the model that has evolved over the previous four sections and introduces some additional features of *FlexSim*.

The model's first extension is the addition of a small warehouse section that follows the Packing operation. That is, containers that have been finished and packed with components flow to racks in a warehouse where they are stored until they are used to fulfill demand. This introduces *FlexSim*'s Warehousing Module.

A second extension is the development of another model segment that adds a simple order-fulfillment process. The ordering process uses the containers that are finished and packed to meet arriving customer demand for packed containers.

In addition, a few of the remaining, but not all of the, basic objects (Multiprocessor, Transporter, Robot, and Crane) are introduced, as well as *FlexSim*'s Fluids Library.

The final section provides a textual summary of the model that has evolved over the course of the primer.

1. INTRODUCTION TO WAREHOUSE MODULE

FlexSim provides a robust set of tools for modeling warehouse operations. These tools can represent complex aspects of warehousing and storage systems. Since this is a primer, the warehousing capabilities are only introduced through a simple example. However, this should provide the basic concepts and a foundation for exploring the topic further. In any case, as has been stressed throughout the primer, it is always best to start simple, have a clear definition of the problem being addressed, and have clear operational objective(s) for developing the simulation model.

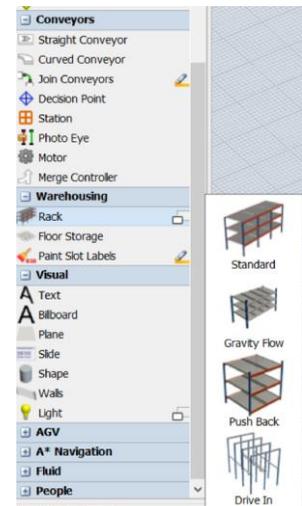
Objects are accessed through the Warehousing section of the Object Library. The most commonly used object, and the only one discussed in this primer is the Rack object. There are other types of racks in the Library for handling specific types of systems, e.g. Gravity Flow, Push Back, and Drive In Racks. While these differ from the Standard Rack, they are structured similarly.

The first time that a Warehouse object is dragged into a model, a Storage System object is added to the Toolbox. Most of the properties in the Storage System are for use in more advanced models; e.g., customizing schemes for addressing locations within a warehouse object, customizing visualization of objects, and triggers.



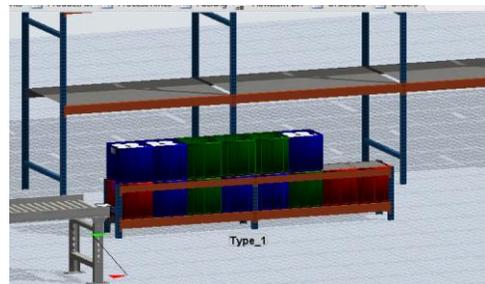
Begin with the basic model named Primer_4-3c, and **Save As** Primer_5-1. Basically make a copy of the model so that it can be customized and the original model is retained.

- Select the Standard Rack from the submenu in the Warehousing section of the object library, as shown in the figure to the right.
- Drag out a Rack object into the model.



In the figure to the right, the default Rack object is in the background and the customized Rack that will be used in this model is in the foreground. Obviously, among other things, the Rack will be resized for the primer model.

Also note in the figure, the Rack is storing all three types of containers.



The following steps customize the basic Rack to the Type_1 object used in this example.

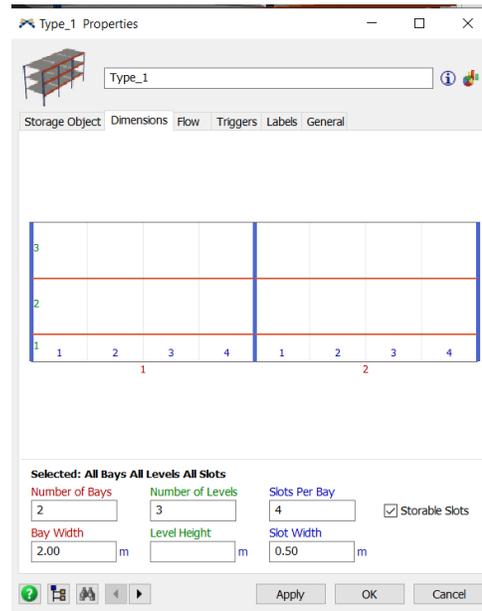
- Rename the general Rack object **Type_1**.
- On the General tab:
 - Set its color to red.
 - Change the size from the defaults of **x = 20.0**, **y = 1.5**, and **z = 8.40** to **x = 4.0**, **y = 0.5**, and **z = 1.25**. Recall that a container is a 0.5 meter cube. Therefore, the capacity of the rack, in terms of the number of containers, is eight containers in the x direction, one in the y direction, and two in the z direction. This reflects what is shown in the figure above.

All of the default values for the properties on the Storage Object tab are not changed.

The settings for the Dimensions tab is shown in the figure to the right and explained below.

However, prior to customizing the tab, a few terms need to be defined. *Bays* are sections of a Rack that represent storage areas along the horizontal axis. Bays are subdivided horizontally into *Slots*. Bays also have *Levels*, storage areas in the vertical dimension. Each bay/slot/level combination is referred to as a *Cell*. Each cell may be considered either storable or not.

- As shown in the figure to the right, update the Rack properties for Type_1.
 - Set the **Number of Bays** to **2**.
 - Set the **Number of Levels** to **3**.
 - Set the **Slots Per Bay** to **4**.
 - Set the **Bay Width** to **2.0**.
 - Set the **Slot Width** to **0.5**.



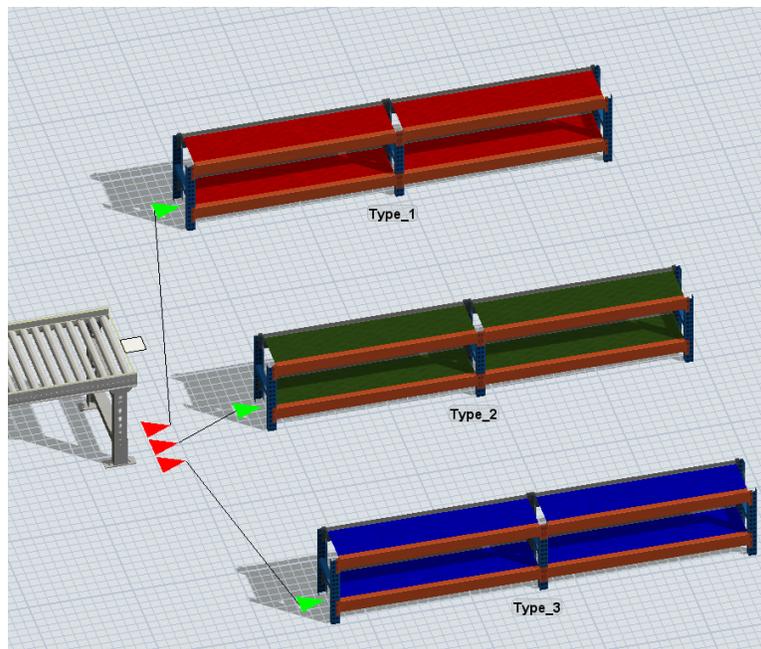
Now, what will likely result is what is shown in the diagram in the figure above, except that all of the **Level Height** values are the same, most likely 1.40. What we want is the bottom level to be 0.25 m. high and the top two levels to be 0.5 m. each. Each cell can be customized by selecting one or more cells in the diagram on the Dimensions tab to highlight them. While highlighted the **Slot Width** and/or **Level Height** properties can be changed.

- Select all of the slots on the bottom row/level and set **Level Height** to **0.25**. Also uncheck the **Storage Slots** box since the lower level is not for storage; it is just to get the items off of the floor. Of course, no containers would be stored there anyway since a container’s height exceeds the height of the cells in that level.
- Select all of the slots on the top two rows/levels and set **Level Height** to **0.5**.

Thus, the 4 x 0.5 x 1.25 meter rack has **2** bays, **4** slots (in each bay), and **3** levels (2 of which can store items). Since a container is a 0.5 m. cube, the rack can store up to 16 containers. By default, containers are placed in the rack starting at Bay 1, Slot 1, Level 2. They are filled as follows: all slots in Bay 1 Level 2, then all slots in Bay 1 Level 3, then all slots in Bay 2 Level 2, and finally all slots in Bay 2 Level 3.

- Copy the Type_1 Rack object twice for the other two types of container.
 - Rename one **Type_2** and set its color to green.
 - Rename one **Type_3** and set its color to blue.
- Position the racks on the layout and connect the Conveyor's Exit Transfer to each Rack in the order of Type; i.e., Rack Type_1 should be connected to the Transfer's port 1, Type_2 to port 2, etc.
- Change the **Send to Port** property on the Exit Transfer to **By Expression** using the drop-down menu. The default value **item.Type** is what we want, to transfer from the Conveyor to a Rack based on the value of the container's label named Type.

The warehousing section of the model should resemble the figure below.



Since there is no Sink in the flow, if the model is run, then the racks will fill, containers will back up on the conveyors blocking the packing area and the finishing machines, and finally the containers' Queue at the beginning of the model will fill and all subsequent entering containers will be redirected elsewhere. The next section adds the logic to represent order fulfillment, where containers are removed from the Racks to meet demand.

2. ORDER PROCESSING SUBMODEL

Packed containers are pulled from the rack storage area to meet demand. Demand for containers of different types are based on incoming orders. The following is a brief summary of a simple representation of the ordering and fulfillment processes. This is followed by a description of how to model this in *FlexSim*.

2.1 Definition of the order-fulfillment process

Orders are released to production twice per 8-hour shift, halfway through (240 minutes) and at the end (480 minutes). This repeats for as long as the simulation runs.

Assume five orders are released at a time. The size of each order is a random variable, assuming there is a request for one container in an order occurs 10% of the time, two containers 15% of the time, three containers 50% of the time, four containers 15% of the time, and five containers 10% of the time. This is referred to as the order-size distribution. Of course, this distribution may change. Based on this distribution, an order consists of one to five containers and the average order size is 3.0 containers.

The mix of containers in an order is assumed to be the same as the previously assumed product mix: 20% are for Type 1 (red), 30% for Type 2 (green), and 50% for Type 3 (blue).

Once all of the containers are available for an order, the time to complete the order is normally distributed with a mean of 3 minutes and a standard deviation of 0.5 minutes.

For simplicity, no transport activities are modeled.

2.2 Implementation of the order-fulfillment process

The order size distribution is stored in a Global Table for easy access in the model and for editing and experimenting.

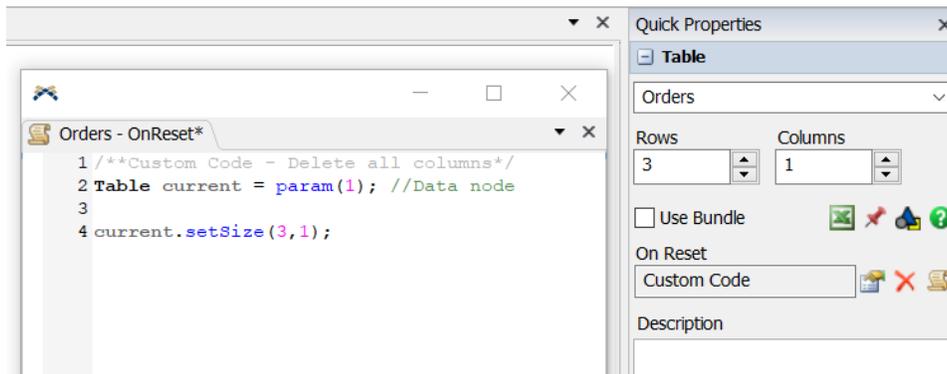
- Create a **5-row, 2-column** Global Table named **OrderSize**, similar to the ProductMix table created earlier. In the model, order sizes will be randomly generated using the discrete-empirical distribution; again, in the same manner as the container's Type is generated at its Source. The first column contains percentages: **10, 15, 50, 15, and 10**, which correspond to the order sizes **1, 2, 3, 4, and 5**, which are stored in the second column of the table.

Another table is used to store the contents of each order. Each column corresponds to an order and each row contains the quantity of each type of container that is in that order. In other words, this file will contain a running history of all of the orders that occur during a simulation run. How this is used is discussed later, just create the base table now.

- Create a **3-row, 1-column** Global Table named **Orders**. The default values of **0** in each cell are fine.

However, the Order table needs to be reset whenever the model is reset; i.e., all columns must be removed. In *FlexSim*, a table must have at least one column. Therefore, there is no option for this in the **On Reset** property in the table's Quick Properties window. Notice there is an option to **Delete All Rows, Clear All Cell Data**, etc., but no Delete All Columns option. Therefore, the logic needs to be specified using custom computer code.

- Select the scroll-like icon to the right of the textbox for the **On Reset** property. This will open a coding window as shown below.
- The first statement, **Table** current ..., provides a reference to the table. It does not need to be modified.
- Add the statement **current.setSize(3,1);** and be sure to enter the statement as shown, including the ending semicolon. Each coding statement must end with a semicolon. This statement is a *FlexSim* command that sets the size of the table to three rows by one column whenever the Reset button is pressed.



When the coding window is closed the **On Reset** property will show **CustomCode**. The code can be edited by clicking on the scroll-like icon to the right of the property.

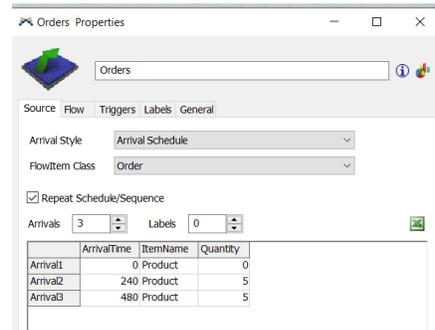
While not needed for this example, the interested reader can learn more about custom coding in *FlexSim* either via the *FlexSim User Manual* (accessed via the Help item on the Main Menu) or by another primer written by this author, entitled *A Primer of Coding in FlexSim 2017*.

Orders are represented as pallet-like flowitems and the containers are placed onto each order pallet as the order is fulfilled.

- Create a new flowitem for the order by duplicating the existing Pallet in the Flowitem Bin. All of the default values are fine, except:
 - Name the flowitem **Order**.
 - Add two number labels, **OrderNum** and **OrderSize** with default values of **0**.

Orders are generated based on a repeating schedule, not randomly, from a Source object.

- Drag out a Source object from the Library and name it **Orders**.
- Change the properties on the Source tab as follows and as shown in the figure to the right.
 - Change the **Arrival Style** from the default **Inter-Arrival Time** to **Arrival Schedule**.
 - Check the box **Repeat Schedule/Sequence**.
 - Create an arrival schedule table with **Arrivals** (number of rows) set to **3**.
 - Change the **Arrival Time** and **Quantity** as shown in the table to the right. Five orders arrive at time 240 and 480. Since the table repeats, further orders arrive again at 720, 960, etc.



The **Arrivals Schedule** works as follows. The three rows are executed at the designated times and then Row 1 repeats 0 time (value of that row’s Arrival Time) after the last row, then all rows are executed and Row 1 again executes 0 time after the last row, and the process repeats.

- On the Source’s Labels tab:
 - Create a number label named **OrderNum** that has a value of **0**. This label’s value will be the total number of orders created and is updated each time an order leaves the Source.
 - Check the box **Automatically Rest Labels**. Whenever, the model is Reset, the initial value of the label, **0**, is reset; i.e., zero orders are created at Reset.

- On the Source’s Triggers tab, create an On Creation trigger, actually there will be three On Creation triggers that set labels, as shown in the figure to the right.

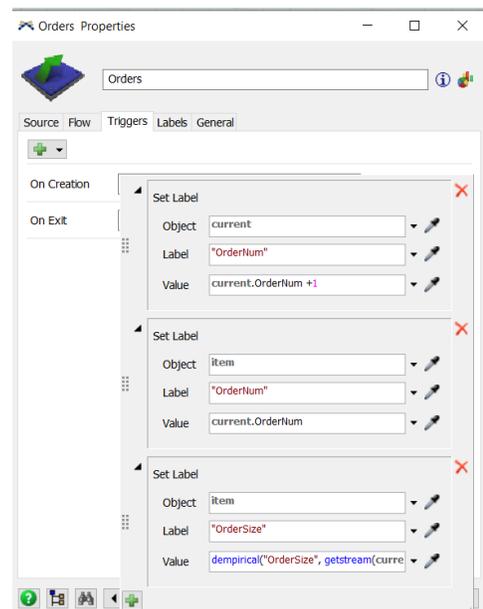
- Select the **Data** category, then the **Set Label** option. As shown in the figure to the right and using the drop-down menu options, set the **Object** property to **current**, the **Label** property to **OrderNum**, and **Value** to **current.OrderNum + 1**.

This increments the total number of orders created.

- Using the green + button, add the second **Set Label** trigger (selecting **Data** then **Set Label**). Using the drop-down menu options, select **item** for the **Object** property and **OrderNum** for **Label**. The **Value** is set to **current.OrderNum** from the **Labels** drop-down menu option.

This sets the created flowitem’s order number; its value is the total number of orders created so far.

- Using the green + button again, add the third **Set Label** trigger (selecting **Data** then **Set Label**). Using the drop-down menu options, select **item** for the **Object** property and **OrderSize** for **Label**. The **Value** is set using drop-down menu option **Statistical Distributions**, then **dempirical**. Change the parameter “MyTable” to “OrderSize”.

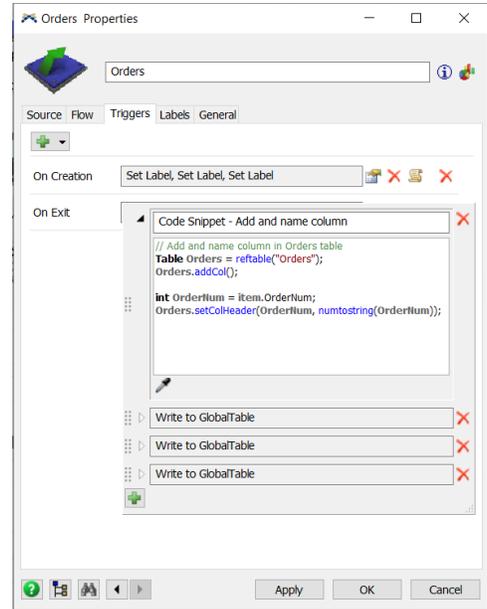


This sets the created flowitem’s order size to a randomly-selected value that is based on the probability distribution defined in the Global Table OrderSize.

Again on the Source’s Triggers tab, create an On Exit trigger, actually there will be four On Exit triggers.

➤ The first On Exit trigger, as shown in the figure to the right, is another small snippet of computer code. Every time a flowitem leaves the Source, it adds a column to the Global Table Orders with the header being the order number. Each line of code is briefly defined and explained.

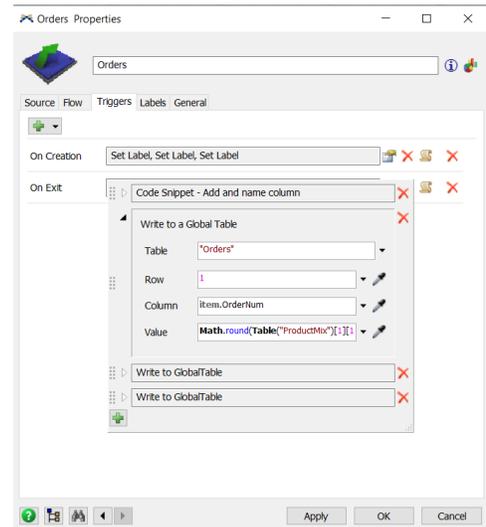
- **// Add and name column in Orders table** is a comment statement that briefly describes what the snippet does. It is not a part of the logic, it is only included for clarity.
- **Table Orders = reftable("Orders");** sets a reference to the table that is being modified.
- **Orders.addCol();** is a *FlexSim* method that adds a column to the Orders table.
- The blank line is only for readability only.
- **int OrderNum = item.OrderNum** declares a local variable as an integer (**int**) and sets its value to the value of the flowitem’s label that is being released from the Source. The label contains the total number of orders released so far.
- **Order.setColHeader(OrderNum, numtostring(OrderNum));** uses a *FlexSim* method to name the column just created to the item’s order number. The header must be text so the order number must be converted from an integer to a string; this is done by the function **numtostring(...)**.



As mentioned above, a complete understanding of the coding is not needed; coding is a more advanced topic. However, the interested reader can learn more about custom coding in *FlexSim* either via the *FlexSim User Manual* (accessed via the Help item on the Main Menu or by another primer written by this author, entitled *A Primer of Coding in FlexSim 2017*).

The Source's On Exit trigger is also used to populate the Orders table. This is done by using three similar triggers, one for each item/container type.

➤ Using the green + button, add the second On Exit trigger by first selecting **Data** and then **Write to Global Table**. The interface is shown in the figure to the right and setting the property values are described below.

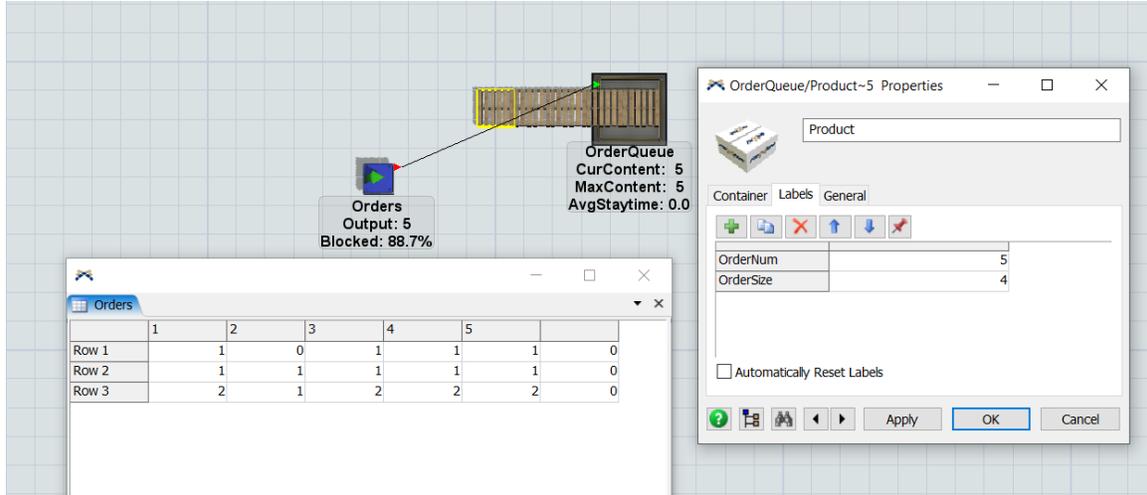


- For the **Table** property, select the table **Orders** from the drop-down menu.
- For the **Row** property, use the default value **1**. This is for item/container Type 1.
- For the **Column** property, select the table **item.OrderNum** from the **Labels** section of the drop-down menu. The column number in the table is the item's order number
- For the **Value** property, type in the following:

Math.round(Table("ProductMix")[1][1]*item.OrderSize/100.0)

This sets the number of item/containers of Type 1 that is in the order. It is the product of the percentage of Type 1s that is in the ProductMix table and the size of the order. For example, the percentage of Type 1s is 20% and if the order size sampled from the OrderSize distribution is 3, then the number of Type 1s in the order is $20 \times 3 / 100 = 0.6$. Since the number of items in an order must be an integer, the Math.round(...) method rounds the value to 1 in this case.

- Repeat the above steps for each other types of containers. In this case, create two more **On Exit, Write to Global Table** triggers. All of the values are the same except:
 - The **Row** value is **2** or **3**, depending on the item/container type.
 - The first **[1]** in the Math.round statement is changed to **[2]** or **[3]**, depending on the item/container type
- Drag out a Queue object and connect it to the Source.
 - Name it **OrderQueue**.
 - Set the **Item Placement** property, in the Visual section of the Queue tab, to **Horizontal Line**. This makes it easier to visualize the orders awaiting fulfilment
- **Reset** and **Run** the model. Since many changes were made, verify that the Source is working properly by checking the following. The model segment should resemble the figure below. In this case, the screenshot was made just after time 240 minutes, soon after the first set of orders was generated.
 - Check the labels on the orders/pallets – **OrderNum** should be sequential and **OrderSize** should vary between 1 and 5.
 - The Orders table should have a column for every order with header being the order number. The rows should be the number of containers of each type in that order.
 - Resetting the model changes the table back to a single column.

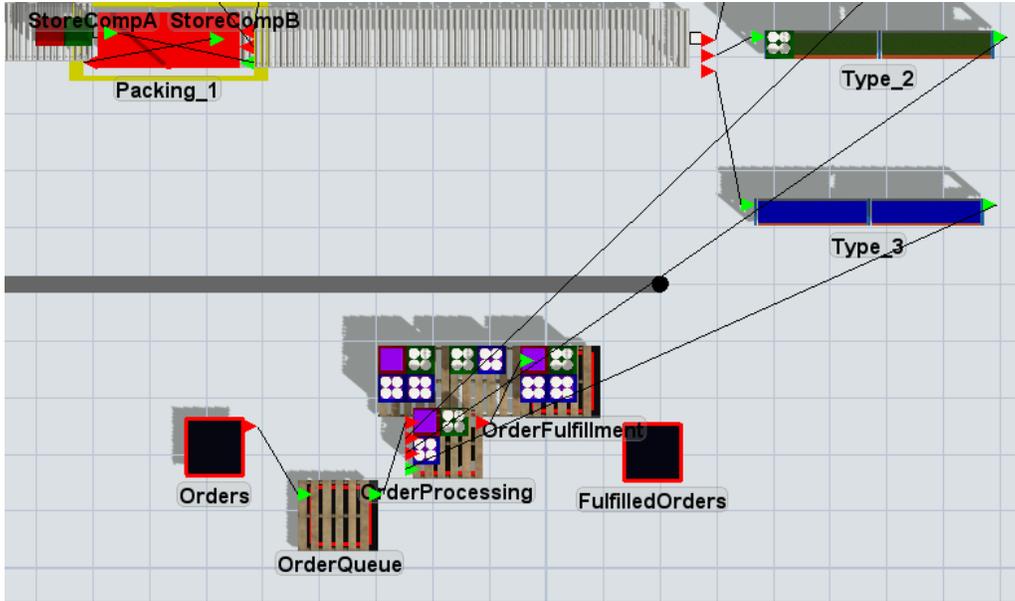


Note in the figure above the fifth order is selected in the 3D view. By double clicking on the selected pallet, the Labels tab shows that the value of the **OrderNum** label is **5**. The other label shows the size of the order to 4 items. Column 5 in the Orders table shows the breakdown of the four items in the order: one Type 1, one Type 2, and two Type 3s.

The Combiner object is used to fulfill the orders. Components from the Racks are combined onto an order pallet. The mix of components is determined from the Orders table.

- Drag out a Combiner object and connect it to the Queue (i.e., the submodel's flow is: Source → Queue → Combiner).
 - Name it **OrderProcessing**.
 - Set the **Process Time** property using the **Statistical Distribution** option, then select **Normal**. In the resulting interface set the **Mean** property to **3** and the **Standard Deviation** to **0.5**.
 - On the Triggers tab, use the + button to add an **On Entry** trigger. Then, select **Update Combiner Component List With Labels**. On the resulting interface:
 - Using the drop-down menu, for the **Table** property, select **Orders**.
 - Set the Label property to **"OrderNum"**.
- Connect each Rack to the Combiner in the order: Type_1 (red), then Type_2 (green), then Type_3 (blue).
- Drag out a Queue and connect the Combiner to it. This is only a temporary object to test the model - it will be replaced by a Sink. If the Sink is connected initially, we cannot visually verify that the orders are fulfilled correctly.
 - Name it **OrderFulfillment**.
 - Set the **Item Placement** property, in the Visual section of the Queue tab, to **Horizontal Line**. This makes it easier to visualize the orders that have been fulfilled.
- **Reset** and **Run** the model and verify that the order processing logic is working properly. Each item in the Order Fulfillment Queue should have the same mix of containers as in the Orders Global Table.

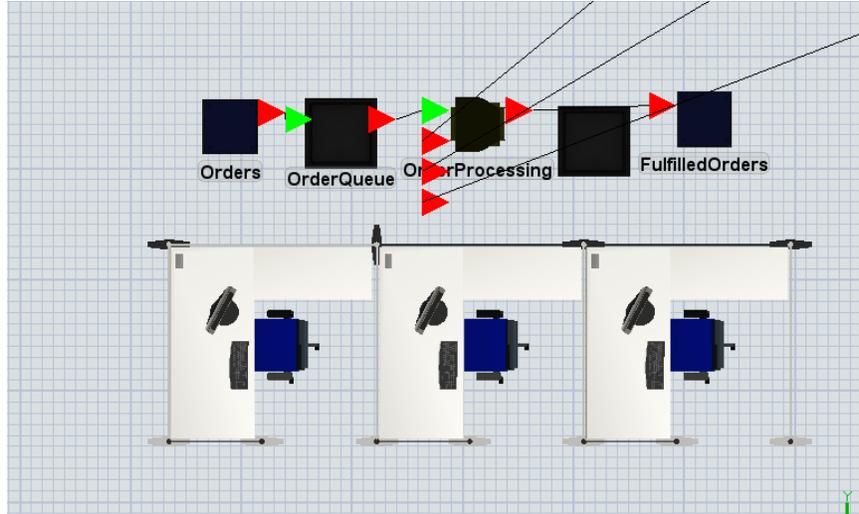
Note in the model shown below, the OrderFulfillment Queue is used to verify the model. At the time the screenshot was taken, three of the five orders that arrived at time 240 have been fulfilled (they are in the Queue), one is in process of being fulfilled and one is awaiting processing.



	1	2	3	4	5		
Row 1		1	0	1	1	1	0
Row 2		1	1	1	1	1	0
Row 3		2	1	2	2	2	0

Compare the Queue contents with the first three columns in the Orders table. For example, the first completed order in the OrderFulfillment Queue contains one red container, one green, and two blue; this corresponds to the first column in the table. Orders two and three in the Queue also match the orders in the table.

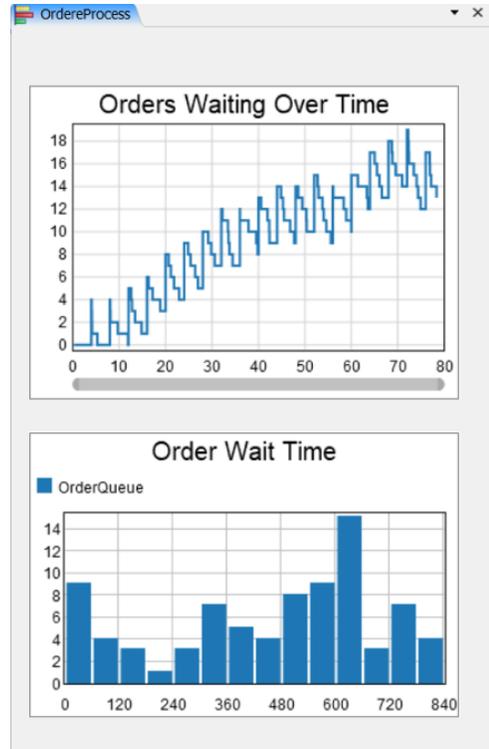
- Once the model is verified, drag out a Sink object and name it **FulfilledOrders**. Replace the connection from the Combiner to the OrderFulfillment Queue with a connection to the Sink.
- The order process can be placed anywhere in the model and the objects can be made small, or hidden, since the way order fulfillment is modeled here it is not a physical process. For example, the Source, Queue, Combiner, and Sink can all be set to 0.5 meter square by 0 meters high, as shown in the figure below.



The order process objects can be hidden by covering them with a visual object.

- Drag out an instance of the **Shape** object (gray cylinder) from the Visual section of the Object Library.
 - Rename the object **OrdersOffice**.
 - On the General tab. Select the drop-down menu for the **3D Shapes** property (fs3d\General\Cylinder.3ds) and **Browse**. Locate the Sketchup Files folder (should be in your FlexSim Models folder) and select the **Office.skp** file.
 - As shown in the figure above, resize the object so that it is 6m in the x direction, 2m in the y direction, and 2m in the z direction.
 - Hide the order process logic by covering it with the office shape.

- Create a Dashboard named **Order Process**, as shown in the figure to the right and as defined below.
- On the Dashboard, add a Content plot from the Content templates section of the Dashboard Library and name it **Orders Waiting Over Time**.
 - On the Options tab, use the green + button in the Objects section to select the object **OrderQueue**.
 - On the Settings tab, using the drop-down menu, change the **Time Axis Mode** to **Show Duration** and change the units from the default **Seconds** to **Hours**.
- On the Dashboard, add a Staytime histogram from the Staytime templates section of the Dashboard Library and name it **Order Wait Time**.
 - On the Options tab, use the green + button in the Objects section to select the object **OrderQueue**.
 - On the Settings tab, change **Bar Mode** to **By Bucket Width** and set the **Bucket Width** value to **60** so that each bar represents one hour of wait time.
 - On the Text tab, change the **Precision** property from **2.00** to **0**



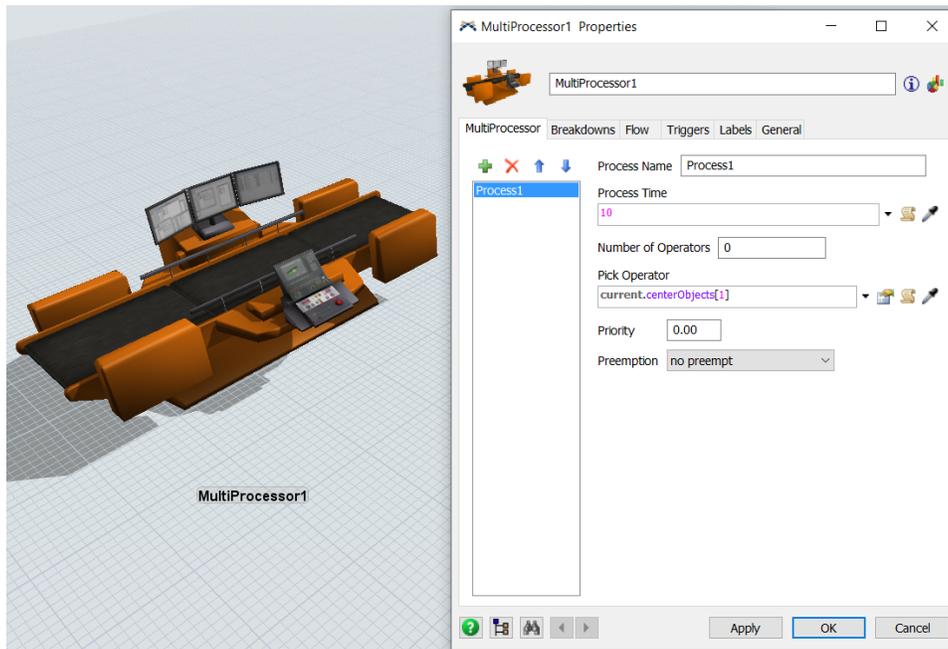
Note that based on the Dashboard plots, the system seems to be out of balance. This is evident in the time-series plot where the number of orders waiting is generally increasing over time. The histogram indicates that many of the order wait times are quite large - many are over ten hours (600 minutes)!

3 OTHER FLEXSIM OBJECTS AND CAPABILITIES

This final section introduces some additional modeling constructs that are used to model other system features than those described in the primer. These are certainly not all of the additional capabilities in *FlexSim*; they are a sampling that are intended to identify and introduce some other features. The additional constructs include other objects: a Fixed Resource object, the Multiprocessor, and three more Task Executer objects, the Transporter, Crane, and Robot. This section also includes another library of objects, the Fluids Library.

3.1 Multiprocessor object

The Multiprocessor is a Fixed Resource similar to the Processor, Combiner, Queue, etc. Its 3D shape and Multiprocessor properties tab is shown in the figure below.



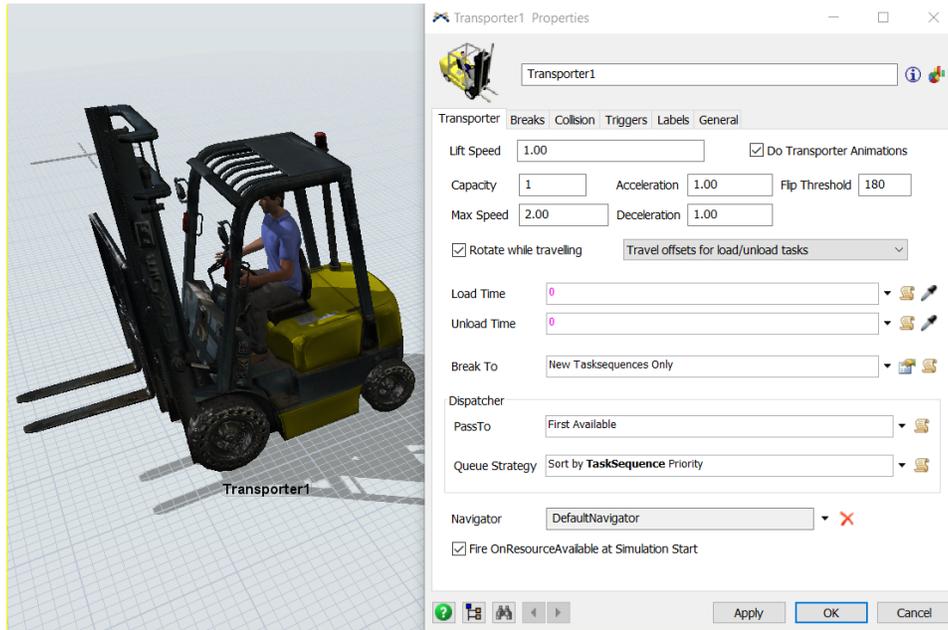
The object puts a single flowitem through a sequence of separate processes; thus, only a single flowitem is in a Multiprocessor at a time. Each process has its own Process Time, and means to set the process time, as well as use one or more Operators.

Different types of flowitems can be processed by different processes. For example, if a Multiprocessor has four process steps defined, say A through D, then flowitem Type 1 might use process steps A and D and flowitem Type 2 might use process steps B and C. In this case, the process time for process B and C would be 0 for Type 1; and, the process time for process A and D would be 0 for Type 2.

It is important to note that if when a flowitem completes a process step and then another flowitem can enter that step, then the Multiprocessor is not the object to use to model this situation. In this case, simply use several Processor objects connect in sequence.

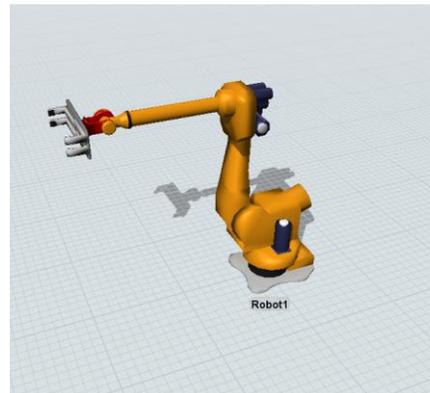
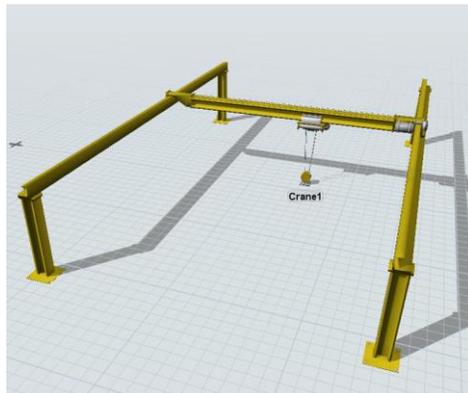
3.2 Transporter object

The Transporter object is a task executor, similar to the Operator object used in the primer. The basic 3D object, which resembles a fork truck, is shown in the figure below, along with the main tab on the object interface. Note the similarities between the Transporter properties and the Operator object properties. One difference between the objects is that the forks on the Transporter raise and lower to load and unload flowitems.



3.3 Robot and Crane objects

The Robot and Crane objects are both task executors, similar to the Operator object used in the primer and the Transporter introduced above. These basic 3D objects are shown in the figure below.



As mentioned previously, all task executors have the same functionality, primarily to interact with Fixed Resources and move flowitems.

The Robot object is a special transport that lifts flowitems from a starting point and places them at an end point. Generally the Robot’s base does not move, but it has six rotating joints that transport flowitems.

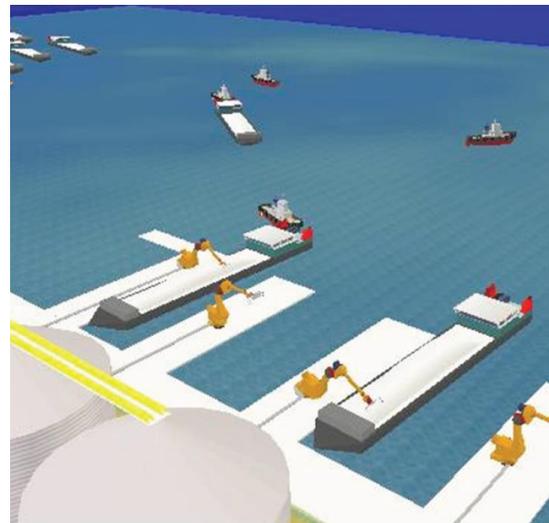
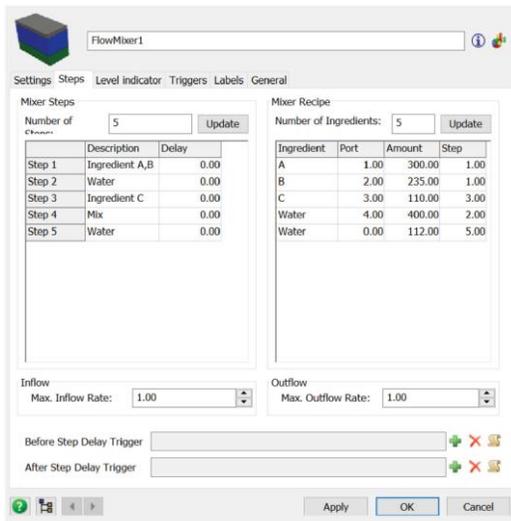
The Crane object is also a special transport that simulates the operation of rail-guided cranes, such as gantry, overhead, or jib cranes. Its functionality is similar to a Transporter, but with modified graphics. By default, the crane picker rises to the height of the crane object after picking up or dropping off a flowitem, then travels to the next location. Of course all of these actions can be customized.

3.4 Fluids Library

Fluid objects are found in the Fluids section of the Drag-and-Drop Library. These objects provide a means to model systems where at least some of the states of the system change continuously over time rather than at discrete points in time. These objects enable *FlexSim* to simulate continuous systems representing fluid flows (e.g., water, oil, and gases), particulate flows (e.g. powders, cereal, and potato chips), and high-speed operations (e.g. bottling). A fluid (or continuous) flow is usually described in terms of units per time, such as gallons per minute.

Example Fluid objects include: Fluid Generators, Fluid Tanks, Fluid Terminator, Fluid Mixer, and Fluid Conveyors. Many of the fluid object are analogous to Discrete objects that have been discussed at length in this primer, such as Sources, Queues, Sinks, Combiners, and Conveyors. There are also objects that convert fluid entities into discrete entities and visa versa. Models that include a mix of both discrete and continuous elements are referred to as hybrid models.

The following figure shows an example interface for the Fluid Mixer object on the left and a full model using Fluid objects on the right. The Mixer object uses both a Step and Recipe table to define batch-processing operations, which are common in most fluid systems. The full model is used to assess operations and resource requirements (e.g. docks and tugs) at a grain-loading facility where barges are loaded with grain from silos.



Congratulations! You have successfully completed the *FlexSim Simulation Software Primer*. Hopefully you now have a better understanding of how to build *FlexSim* simulation models to solve problems and make better decisions through the power of discrete-event simulation and the robust capabilities of *FlexSim*.

4 SUMMARY OF THE PRIMER MODEL

This final section provides a textual summary of the model that evolved throughout the primer. The final model is the result of a development process that progressed from a very simplified version of the system to one that more closely represents the real operations.

Recall that Dobre Products Limited (DPL) is planning to reuse an area in one of its production facilities to finish and pack containers for distribution. The new production area will finish various types of containers and then the containers are packed, where the contents depend on the type of container. The packed containers are then moved to a warehouse area where they are used to fulfill customer demand.

In this initial model, which would likely continue to evolve and expand as the design process continues, it is assumed that there will only be three types of containers packed with two types of components. Of course, based on the manner in which the model was developed, it is quite easy to expand the model to consider a larger number of products and components as well as increased production resources, people and equipment.

The basic units of measure are meters and minutes.

The key properties of the three products are summarized in the following table.

Property	Product/Container		
	1	2	3
Size (l x w x h)	0.5 x 0.5x 0.5	0.5 x 0.5x 0.5	0.5 x 0.5x 0.5
Percent of product mix	20	30	50
Color	Red	Green	Blue
Finishing Time, average	15	20	30
Packing Time, average	2.5	3.33	3.83
Quantity of Components A, B packed	2, 0	0, 4	1, 4

The key properties of the two components that are packed into the containers are summarized in the following table.

Property	Component	
	A	B
Size (l x w x h)	0.4 x 0.4 x 0.2	0.2 x 0.2 x 0.2
Shape	Box	Cylinder
Color	Purple	White
Batch Size	24	60
Initial Inventory	12	30
Reorder Point	6	20
Time to Receive and Order	120	120
Time to unload, per batch	1	1
Time to unload, per item	0.05	0.05

The containers arrive to the finishing area (upstream boundary of the model) with an average time between arrivals of 15 minutes. (Alternatively, the average arrival rate of four containers is per hour.) The times between arrivals are triangularly distributed with a minimum time of 5 minutes, maximum time of 30 minutes, and most-likely time of 10 minutes.

- There are no breaks in the arrival pattern; i.e., there is no downtime in the upstream operation.
- The type of container that arrives is random and is based on the percentage of product mix; i.e., there is no batching of containers in upstream operation.

Arriving containers wait to be placed on a finish machine in an area that has the capacity to store five containers; the size of the storage area is 1.1 meters by 8 meters. If when a container arrives and there is no available space, it is diverted to another area in the facility and is no longer considered in the model other than it is counted as a “diverted item.”

Containers are loaded onto one of two finish machines by a single finish operator. The operator uses the Shortest Processing Time (SPT) rule where the container that is the quickest to process on a finish machine is loaded first. However, Dobre does not want items to wait in the buffer too long; therefore, they use the SPT rule unless a container has waited more than 30 minutes or some other specified threshold value.

Currently, there are two identical finish machines that can process any type of container. Each machine is 3 meters square and 2.5 meters high. The process time depends on the type of container. A setup is needed before processing if the container being loaded is different than the one that just finished processing. The setup is performed by the finish operator and the time to perform the setup is 1 minute. Once loaded, the machine processes the container without the operator. After processing, containers move onto a conveyor that transports it to the packing area.

Finish machines are subject to two types of downtime. The first type of downtime is for a quality check where each machine pauses for 15 seconds every ten minutes to upload data. The second type of downtime is when there is some type of failure within the machine. The time between failures is a random variable that is exponentially distributed with a mean of two hours. The time to repair the problem is also a random variable that is uniformly distributed between 5 and 15 minutes. For the failure downtime, the finish operator performs the repair. Failures can only occur when a machine is running.

After the finish operator loads a container into the finish machine and performs a setup operation, if needed, the operator travels to a kiosk and records some production information. The recording time is uniformly distributed between 30 and 90 seconds. Once the information is logged in, the operator is then free to perform another task.

The finish operator travels at an average speed of 60 meters per minutes and takes 30 seconds to load or unload any item. The operator takes two 15-minute breaks per shift, one after two hours and one after six hours into the shift. The operator also takes a 30-minute meal break in the middle of the eight-hour shift. For all breaks, the operator travels to a break area outside of the finish and packing areas.

Once a container arrives at the single packing station, it is loaded with the number and type of components that are specified in the table above and then moved by conveyor to the warehouse. Once in the warehouse area, each container is placed on a rack; there is one rack for each type of container. The packing station follows the same break schedule as the finish operator. The speed of all conveyors is 90 meters per minute.

Components to be packed into the containers are delivered according to a Reorder-Point type of inventory system. That is, when the on-hand quantity of a component in the packing area drops to a specified reorder point, a batch of that component is ordered. When a batch of components arrives, the finish operator unloads the batch. The time to unload a batch is one minute plus three seconds per component.

Customer orders for containers arrive throughout the day, but are released to production twice per shift, once midway through the shift and once at the end. Five orders are released at each of those times. The number of containers per order averages three, but varies between one and five (10% of the orders are either one or five containers, 15% are two or four, and 50% of the orders require three containers). The type of container varies, but is the same as the product mix defined in the first table above.

Orders are fulfilled in the warehouse in the order in which they are released. Once all of the containers for an order are gathered, the time to complete an order is normally distributed with a mean of three minutes and a standard deviation of one-half minute.

The following are the key performance measures of interest to Dobre. Adjusting production parameters, such as reorder point and quantity, process times, equipment speeds, etc. should help them better design the facility.

- The time it takes to fulfill an order in production, from release to production to release to shipping.
- The number of orders waiting to be fulfilled in production.
- The number of components in inventory.
- Number of containers awaiting finishing and packing.
- Utilization of finish machines and finish operator.
- ...

Each simulation run is for 80 hours and when running experiments, each scenario is replicated 20 times.

EPILOGUE

This primer focused on the modeling and analysis of operations systems in order to understand and analyze their dynamic behavior and performance (operations dynamics). The reader is led through detailed, step-by-step instructions for building a comprehensive simulation model. A screenshot of the final model is on the next page. The model evolves and builds upon simpler models using a sequential model-development process. In addition to describing the mechanics of building a model, the primer demonstrates the power of using discrete-event simulation in general, and *FlexSim* in particular, to support the decision-making and problem-solving processes. As the model evolves insight and rationale are provided and good modeling and analysis practices are introduced – this is to demonstrate that model building is not just carrying out rote commands.

While all of *FlexSim*'s capabilities are not presented, the primer raises awareness of many of the more advanced features available in *FlexSim* without covering the details at that time. Successfully completing the primer should provide the reader with a solid foundation to explore topics in more detail in the *FlexSim User Manual*, tutorials, videos, blogs, *FlexSim Answers*, etc.

It was also beyond the scope of the primer to discuss, or even introduce, all of the topics and methods that are associated with simulation modeling and analysis. Therefore, it is suggested that the interested reader consult the following or other more general references on simulation.

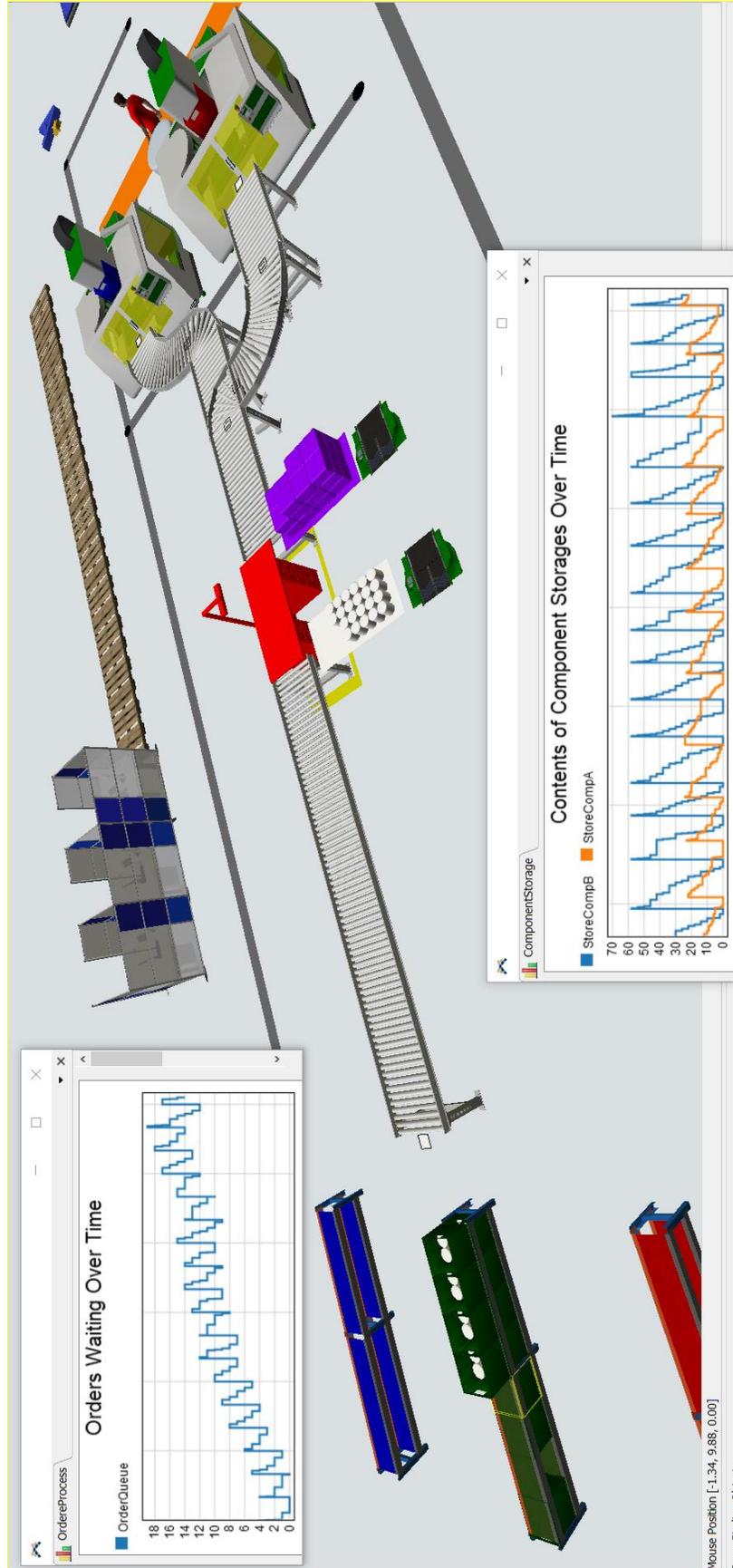
Greenwood, A. *Simulation Primer*, FlexSim Software Products, Inc., 2019.



Beaverstock, M., Greenwood, A., and Nordgren, W. *Applied Simulation Modeling and Analysis Using FlexSim*, 5th Edition, FlexSim Software Products, Inc., 2017.



Since improvements cannot occur in a vacuum, the author welcomes comments and suggestions; please send them via email to allen.greenwood@flexim.com.



APPENDIX - LIST OF MODELS

This appendix provides a list and brief description of the 23 models that are discussed in the primer. Most of them are used as part of a sequential model-development process that evolves from a very simple representation of the system being considered to a more complex, more realistic representation. A few of the models are smaller “study” models that are used to understand, test and/or validate a method or concept in isolation before incorporating it into the main model.

As has been mentioned several times, all of the models listed below, along with supporting files (a layout file and two 3D representations of work areas in the example), are available in a resources folder that can be obtained at <https://flexsim.com/primer-files>.

<u>Filename</u>	<u>Brief Description</u>
MyFirstModel	Part 1, Section 4. Basic objects and connections. Simple, single-server queueing system using all default values.
Primer_1-1	Start of the primer models. Same as MyFirstModel..

Note: All subsequent primer models begin with the previous model and add further customization to the model in order to more closely represent the system described in Part 1, Section 2.

Primer_1-2	<p>Part 1, Sections 5 (Basic Fixed Resource Objects and Customization) and 6 (Basic Model Output).</p> <ul style="list-style-type: none"> • Further description and customization of the following basic objects: <ul style="list-style-type: none"> ○ Source - inter-arrival times to triangular distribution; flowitem class to Tote; On Creation trigger to set Type label and color. ○ Queue – maximum contents and item placement properties. ○ Processor – process time based on product type; set-up time based on whether product changes. ○ Sink ○ Flowitem Bin • Run Time and Run Speed. • Object statistics – on-object stats, Quick Properties stats. • Dashboard – time-series plots of contents and staytime histogram.
Primer_2-1	<p>Part 2, Sections 1 (Changing 3D Graphics) and 2 (Alternative Routing)</p> <ul style="list-style-type: none"> • Importing graphics – layout and object shapes; changing size, location, rotation, and shape of objects; reference points. • Alternative routing rules, Flow tab.

- Primer_2-2
- Part 2, Section 3 (Task Executors)
- Dispatcher to control Task Executor (Operator)
 - Operator transports flowitems; Operator performs setup operation on Processor.
 - Operator properties: speed, load/unload times, location on model reset.
 - Creation and use of Path Networks to control Operator travel.
- Primer_2-3
- Part 2, Section 4 (Conveyors)
- Straight, Curved, and Join conveyor objects; Transfers.
 - Conveyor types and properties.
- Primer_2-4
- Part 2, Section 5 (Global Tables)
- Global table to store and use process times.
 - Global table to store and use product types (percent product mix).
- Primer_2-5
- Part 2, Section 6 (Downtime)
- Time Table to control time and duration of Operator breaks; change color and travel to break location.
 - Reliability, using MTBF/MTTR tool to represent downtime for a fixed-interval quality check activity that is clock-dependent and requires no repair resource.
 - Reliability, using MTBF/MTTR tool to represent downtime for a randomly occurring machine breakdown that is state-dependent and requires an Operator for repair.
 - Chart the impact of downtime using a pie chart of utilizations (Processors and Operators) on a Dashboard.
- Primer_3-1
- Part 3, Section 1 (Combining and Separating Flowitems)
- Create and customize flowitems in the Flowitem Bin.
 - Use scheduled arrival of batches at a Source, rather than inter-arrival times.
- Primer_3-2
- Part 3, Section 1 (Combining and Separating Flowitems)
- Fixed-recipe Combiner (default) to simulate packing operation.
 - Global Table to define a Combiner's combining recipe; i.e., by flowitem type.
 - Extend a Global Table to contain packing times by product type in addition to process times.

- Primer_3-3
- Part 3, Section 1 (Combining and Separating Flowitems)
- Time Table to set breaks for the packing station.
 - Global Table to set the inter-arrival time at a Source.
 - Object Label to identify the type of component it generates. The item's Type label is set to the same value as the object's label.
 - Separator to convert the arrival of a batch to the number of flowitems in the batch. Batch size is stored in a Global Table, along with other parameters.
 - Operator to unpack a batch of items.
 - Process Time, the time to unpack a reorder of components is based on the type of product and the batch size. The time is based on an expression of Global Table values. Introduces coding in *FlexSim*.
 - Extended path network, including curved paths.
 - Display the contents of a buffer over time in a Dashboard.
- A-StarStudy
- Part 3, Section 2 (Controlling Task Executer Travel Paths Using A*)
- A* is an alternative means to path networks for controlling task executer travel. This simple model illustrates the basics of how A* works and how to incorporate it into a model.
 - Illustrates the use of a study model to test a concept and understand a model's behavior in an isolated environment.
- Primer_3-4
- Part 3, Section 2 (Controlling Task Executer Travel Paths Using A*)
- Set up the model to use the A* algorithm in order to control Task Executer (TE) travel. A* is an alternative means to using Path Networks for TE travel.
 - Specify which model objects the TE needs to avoid; add Barrier and Divider objects.
 - Show travel paths via a Heat Map.
 - Define and access Model Views.
 - Re-order the contents of a queue when an item arrives based on an item label
- Primer_3-5
- Part 3, Section 3 (Using an Item List for Complex Routing Logic)
- Simple re-ordering rules for Queues. Sort Queue based on type of flowitem.
 - Basic introduction to, and definition of, the Lists modeling tool, including defining Fields, using Push To and Pull From Lists constructs, and viewing a List's contents and backorders.
 - In the example, a List is used to re-order a Queue based on two criteria – time in queue and item type.

- Primer_3-6a
- Part 3, Section 4 (Experimentation in *FlexSim*)
- Basic setup of the Experimenter – defining decision/independent variables, performance measures, and run time parameters; generating and analyzing standard output.
 - Assess the affect of buffer size on performance (number of redirected arrivals to insufficient storage).
- Primer_3-6b
- Part 3, Section 4 (Experimentation in *FlexSim*)
- An extension example to assess the affect of an additional variable, product mix (percentage of each product type), on another performance measure (system throughput).
- Primer_4-1
- Part 4, Section 2 (Introduction to Process Flow – Creating Initial Inventory)
- Basic Process Flow environment.
 - Simple Process Flow Model to create flowitems in all Queues (initial inventory) using Scheduled Source and Create Object activities. Parameters set using Global Table lookups.
 - Use an Array-typed Global Variable to generalize the above process so that inventory can be initialized in any number of Queues.
- Primer_4-2
- Part 4, Section 3 (Modeling Inventory Reorder Policy Using Process Flow)
- Create a label on each Queue denoting what component it stores.
 - Create another Global Variable that contains a pointer to the Separator object for each component.
 - Create a Group of Queues that store components. Use the Group tool to help develop a general process for managing inventory in any number of Queues.
 - Check for the need to reorder by “listening” for exits from Queues using the Event-Triggered Source activity. Using the Assign Labels activity to obtain and save information on the current state of the system. Compare the number in Queue with the re-order point using the Decide activity.
 - If the decision from the process described above, requires a reorder, use: the Delay activity to implement the reorder time, Change Visual to change the object’s color denoting a reorder has been placed, and the Create Object activity to place the ordered item in the model.
- Primer_4-3a
- Part 4, Section 4 (Custom Task Sequences in Process Flow)
- Addition of a 3D model object that represents a kiosk and including it on the Operator’s travel path. It is used later as part of an example that creates a custom task sequence in Process Flow to control an Operator’s behavior.
 - Increase incoming production rate by 25% by adjusting the Source’s inter-arrival time distribution parameters.
 - Miscellaneous model housekeeping actions: confirm parameter settings, revert the model back to using the path network for Operator travel that had been replaced by the A* algorithm, add curvature to some of the network paths to avoid objects, and hide the imported layout.

- Simple TE Process Flow Part 4, Section 4 (Custom Task Sequences in Process Flow)
- Create a simple custom task sequence in Process Flow and compare its behavior to the default travel task sequence that is used by Operators.
 - Illustrate the relationship between events in the 3D model and Process Flow.
- Primer_4-3b Part 4, Section 4 (Custom Task Sequences in Process Flow)
- Apply the concepts learned in the study model to the main primer model.
 - Add a custom task sequence to the finish operator that involves the operator entering information at a kiosk after loading a finish machine.
- Primer_4-3c Part 4, Section 4 (Custom Task Sequences in Process Flow)
- Add the Processor's setup operation to the custom task sequence created in Process Flow in the previous model. The setup task is added to the transport and kiosk work task sequence. Requires adding a label to the Processor that is the flowitem's type that just finished processing.
- Primer_5-1 Part 5, Section 1 (Warehouse Module) and 2 (Order Processing)
- Use customized Rack objects to store components.
 - Introduce a simple order-generating and order-processing model.
 - Orders have a randomly-generated mix of components based on an empirical distribution. Orders are fulfilled from the Racks. Complexity arises in writing each order's mix of components to a Global Table that is used by a Combiner to fulfill the order. Small custom code snippets are required.
 - A new dashboard of performance measures is added.
 - The order processing logic is "hidden" in the model using a Visual object with an imported office-like shape.

APPENDIX - GLOSSARY

This appendix provides definitions for some of the key terms used in the primer.

3D shape	A visual representation of an <i>object</i> that is defined through an imported graphics file. The size, location, color, etc. is defined through the object's General tab user interface.
A* Algorithm	A heuristic search method used to find the shortest path between objects considering barriers or obstacles and no-travel zones.
A* Module	A <i>FlexSim</i> tool for creating barriers, and identifying which fixed objects are barriers, in a model. The barriers influence and restrict Task Executer (TE) travel paths between objects. In contrast to <i>Path Networks</i> , which also controls Task Executer movement, A* tells the TE where <u>not</u> to travel, compared to path networks, which restricts where a TE can travel.
Activity (Process Flow)	Logical operations or steps in a logical process that are the building blocks of <i>FlexSim's</i> Process Flow. They are analogous to objects in 3D in that they (1) are dragged from their library onto a Process Flow modeling surface or workspace and (2) have properties that define their behavior.
Activity Set (Process Flow)	Preconfigured activities bundled together that model a basic People Module task, such as Walk then Process, Escort then Process, Wait then Process, etc.
Address (Rack)	Identifier of where an item is located/stored in a storage system. Typically, a combination of letters, numbers, and separators that uniquely identifies a <i>slot</i> in a storage system.
AGV	Automated Guided Vehicle. A means to transport items using portable robots.
Analysis	See <i>Simulation Analysis</i> .
Animation	A sequence of object movements that are triggered as a simulation runs.
Array	A data structure that consists of an ordered series of elements that are indexed.
Bay (Rack)	A section of a Rack object that represent storage areas along the horizontal axis.
Continuous simulation	A type of simulation where the states of a system change continuously over time, e.g. the filling of a tank with water or other fluid. This is compared to <i>discrete-event simulations</i> where states change at discrete points in time, e.g. a part arriving at a machine to be processed.
Conveyor	A mechanical means to move items between points in space. There are different types of conveyors, all of which can easily be modeled in FlexSim, including: <ul style="list-style-type: none">• Belt conveyor, sometimes referred to as a non-accumulating conveyor, behaves as follows. An item travels down the conveyor until the end and then stops if it cannot be removed (downstream object or transport is not

available). When one item stops, the belt stops and then all other items on the conveyor stop in their current location,

- Merge conveyor, combine items from multiple conveyor lines into a single line for further transportation.
- Power and free conveyor, uses a free rail and multiple drive rails to move items loaded on trolleys through a system.
- Roller conveyor, sometimes referred to as an accumulating conveyor, behaves as follows. An item travels along the conveyor until the end and then stops if it cannot be removed (downstream object or transport is not available). Subsequent items on the conveyor continue to flow on the conveyor and stop behind the one in front.
- Slug conveyor, a type of merging where a group of accumulated items are release for further transport as a group, thus items are released in “slugs.”.
- Sorting conveyor, transports items to different destinations based on stated criteria and item properties. Sorting typically involve conveyor logic objects such as decision points, stations, and photo eyes.

Dashboard	An area for displaying values of some variable as a model runs, typically the current state of a system or a summary measures (e.g. mean). The display may be in text form, but is typically a graph or chart that updates as a model runs. Dashboards provides a means to view the system dynamics.
Discrete-Event Simulation	The means to consider the dynamics of a system by creating and managing events at discrete points in time. Each event changes one or more states of a system. The resulting state change and modeling logic causes an action or activity to occur.
Domain expert	A person with specialized knowledge or skill in an area of interest.
Dot Notation	A notational convention in object-oriented programming where the elements of an object (variables and methods) are separated by dots, e.g. Processor.Type where Type is a variable attribute of the object Processor.
Downtime	The time period when a resource is not available to perform its basic functions. Downtimes are either planned or unplanned. <ul style="list-style-type: none"> • Planned downtimes are expected to occur at known times and typically recur on a regular basis. Examples include: shift schedules, operator breaks, and periodic quality checks. Planned downtimes are implemented in <i>FlexSim</i> via the Time Table tool. • Unplanned downtimes occur unexpectedly. Examples include: machine breakdowns and operator absenteeism, Unplanned downtimes are implemented in <i>FlexSim</i> via the MTBF/MTTR tool.
Event	An occurrence at an instance in time. When an event occurs, it causes one or more system states to change and possibly one or more actions.
Experiment	A set of <i>scenarios</i> and <i>performance measures</i> considered for analysis. The simulation model is run for a prescribed number of replications and duration and may involve excluding a <i>warm-up</i> period.

Experimenter	A tool for designing, executing, analyzing, and possibly optimizing simulation models. It provides a means for defining <i>scenarios</i> , <i>performance measures</i> , number of <i>replications</i> , simulated time or duration, and whether a <i>warm-up</i> period is considered. IA direct connection to <i>OptQuest</i> for optimization is provided.
Expert Fit	Software included with <i>FlexSim</i> , developed by Dr. Averill Law, to best fit data to probability distributions. The software is typically used to define input distributions to a simulation model from operational data, but may also be used to characterize output data from a simulation in terms of probability distributions.
Fixed Resource	Objects in a model that are fixed or stationary, e.g. Processor or Combiner. Typically, fixed resources process flowitems.
FlexScript	A scripting language, that is a subset of C++, within <i>FlexSim</i> that can be used to define logic and behaviors via custom coding. FlexScript includes many pre-built functions, or commands, for performing common programming operations.
Flowitem	As the name indicates, an item that flows through a simulation model. Movement is triggered by events and controlled by model logic. Similar to entities or transactions in other simulation software.
Flowitem Bin	A tool in the Toolbox for customizing and creating flowitem classes, including size, shape, graphic, labels, packing method, etc.
Global Table	A systematic means to store information in terms of rows and columns.
Global Variable	User-defined information that can be accessed and updated from anywhere in a FlexSim model at any time during a simulation. Each variable is typed as Integer, Real, String, Array, etc.
Label	A user-defined property, on an object or item, that can be defined, used, and/or updated at any time during a simulation.
Level	Storage spaces along the vertical axis of a <i>Bay</i> in a Rack object.
List	A tool for creating complex flows in a model. Basically, in a certain condition an object pushes information onto a list (referred to as a list entry) and under a different condition an object, using selection criteria, pulls an entry from the list, resulting in an action in a model.
Mobile Resource	see <i>Task Executer</i> .
Model	Representation or abstract of a system and its behavior. A model represents a system's components and the interaction among the components, variability inherent in the system, and the system's dynamic behavior.
Model Fidelity	The degree to which a model represents the real system. While it might appear that more is better; in this case, it is not. A model should only be as detailed as needed to answer the posed question(s). Of course, this is easier said than done; determining the right fidelity comes with practice and experience.
Modeling	See <i>Simulation Modeling</i> .

Modeling Surface	The interface where models are built in 3D space by dragging and dropping objects onto the gridded-surface and locating them in 3D space in terms of their x, y, and z coordinates. In <i>FlexSim</i> , the surface grid's unit of measure is specified as Length Units when a model is started; e.g., if the Length Unit is feet, then each grid unit represents one foot in the x and y directions. Similar modeling environments are available when defining model logic in Process Flow and creating charts and graphs in Dashboards; however, these are just dimensionless work surfaces for organizing work.
Monte Carlo simulation	A means to obtain numerical results based on repeated random sampling.
MTBF/MTTR	Mean Time Between Failures / Mean Time To Repair. In <i>FlexSim</i> , this tool provides the ability to model the reliability of an object in terms of a set of parameters and properties such as: the time between failures (operating time), downtime duration, resources needed, states affecting the time between failures (e.g. clock time versus processing time), etc. Typically, the time between failures and repair times are random variables.
Object	A pre-built, yet customizable, representation of actions commonly found in operations systems, such as planned and unplanned delays (e.g. processing, storage), transportation via fixed (e.g. conveyor) or mobile (e.g. operator) means, combining or separating objects, etc. Its behavior is defined through properties that are specified via its user interface, the <i>Process Flow</i> logic builder, or custom computer code using <i>FlexScript</i> .
Object Library	A list of available objects that can be selected and dragged and dropped onto the modeling or work surface. The primary <i>FlexSim</i> library contains Fixed Resources, Task Executors, Conveyors, Fluids, Modules, etc. User libraries of objects can be developed that contain special user-developed objects. Other libraries are available for the defining logic in Process Flow and creating charts and graphs.
Object Flow Diagram	A diagramming methodology, used during the conceptual design phase of model development, to identify and represent the various system elements and relationships that must be considered by the simulation.
Operational dynamics	The changing behavior and performance of an operations system over time.
Operations system	A collection of elements that transform input into output through a set of related activities and processes that require a variety of resources, such as equipment, material, people, and information. Operations systems are characterized by complex interactions among resources, numerous sources of variability, and exhibit dynamic behavior, i.e. change over time.
<i>OptQuest</i>	A simulation optimization software product, developed and maintained by OptTek Systems, Inc., that provides algorithms and analysis techniques for determining the best input values to obtain the best outcomes.
Path Network	A set of connected nodes that defines a path on which Task Executors (TE) can travel. In <i>FlexSim</i> , the paths can restrict direction of travel, speed and, passing, as well as be straight-line or curved. In contrast to <i>A-Star</i> , which also controls Task

	Executer movement, a Path Network explicitly restricts or limits where a TE can travel; whereas, A* identifies where a TE needs to avoid or cannot travel.
Performance Measure	An output of a simulation model that is of interest for analyzing and understanding the estimated behavior of the system being considered.
Port	A way for <i>flowitems</i> to move in to and out of objects (input ports and output ports). Also, provide a means for objects to communicate (center ports). Whenever two items are connected, a port is created on each object. Theoretically, there is no limit on the number of ports an object may have.
Process Flow	Drag-and-drop, flowchart-like logic builder within <i>FlexSim</i> that is used to specify inter-object or intra-object logic.
Pull logic	A means of flow between objects where the decision as to which object an item is obtained from is determined by the receiving object, not the sending object. See also <i>Push logic</i> .
Push logic	A means of flow between objects where the decision as to which object receives the item is determined by the sending object, not the receiving object. See also <i>Pull logic</i> .
Queueing System	Composed of three main elements - customers, servers, and queues (also referred to as buffers). Customers require service from one or more servers and if a server is not available when the customer demands the service, the customer waits in queue until a server is available. The interaction among the three elements result in the behavior of the system.
Reliability	The probability that an object performs its desired functions under certain conditions for a period of time. Another way to state this is that reliability is the probability that an object does not fail over a period of time. See <i>MTBF/MTTR</i> for the means to specify reliability in <i>FlexSim</i> .
Replication	The repeating of a simulation experiment. Replications are necessary because simulation involve random variables or stochastic parameters.
Routing	The logic that is used to move flowitems between objects. Commonly, items are “pushed” from one object to another, but may also be “pulled” from one object to another. Routing is typically defined on an object’s Flow tab. <ul style="list-style-type: none">• Push logic involves deciding which output port will be used and includes the first downstream object that is available or the one with the shortest queue, conditioned on the value of the item’s attribute (label) or a state of the system, etc.• Pull logic options are similar to those used for pushing, but the decision involves which input port on the object should be used to obtain an item.
Run Time	The duration of a simulation run in simulated time, not real time. Time in <i>FlexSim</i> is unit-less; it is given context through the user’s specification of time units (e.g., seconds, minutes, days). If the user specifies the model units as seconds, then a Run Time of 10,000 is 10,000 seconds of simulated time (about 2.8 hours).

Run Speed	How fast a model runs relative to real time. For example, if a model's units are specified as seconds, then a Run Speed of 1000 means the model runs 1000 times faster than real time. Therefore, if a model's Run Time is 10,000 seconds (about 2.8 hours) and Run Speed is set to 1000, then the 2.8 hour simulation will take 10 seconds to run.
Scenario	A combination of model variable settings or characteristics. The values of the variables are changed during experimentation in order to determine their affect on estimated system performance.
Simulation	A process that involves the modeling and analysis of an operations system in order to improve organizational performance.
Simulation Analysis	Simulation models are built for analysis. As such, this is the ways and means of using a simulation model to experiment with and test ideas and alternatives before deciding actions and committing resources. Analyses may be performed within the software or exported to other software. In addition to output analyses, input analyses are needed to define system properties and represent them in a model. Oftentimes, input analyses involve selecting the most appropriate probability distributions to represent aspects of the system, e.g. process times, inter-arrival times, and quality.
Simulation Modeling	The ways and means for representing a system physically (size, distance, speed, etc.) and logically (what, who, when, and where things are done, as well as how much and how long) in order to understand its behavior over space and time and to assess possible consequences of actions, virtually.
Slot (Rack)	A storage space in a Rack object that is a subdivision of a <i>Bay</i> in the horizontal direction.
State	A condition of a system or the value of a system variable, such as whether a resource is busy or idle or how many customers are currently in the system.
Staytime	How long an items stays in an object.
Study Model	A small model that is developed to test and validate a method or concept in isolation before incorporating it into the main model. This good modeling practice is commonly used to develop logic for a single object or set of objects.
Table	A systematic means to store information. In <i>FlexSim</i> , tables can be imported from, or exported to, Microsoft Excel. See either <i>Global Table</i> or <i>Timetable</i> .
Task Executer	A mobile resource that moves about in a model, typically to move flowitems between objects. They include such objects as operator, fork truck, AGV, crane, etc. As the name indicates, the object executes a sequence of tasks that are sent from another object. Tasks include travel between objects, load an item, process an item, etc.
Task Sequence	A set of tasks that a task executer (operator, fork truck, crane, etc.) carries out. Typically, task sequences are requested and sent by fixed resources and involve tasks such as travel between objects, load an item, process an item, etc.

Timetable	A means to define and specify a resource's deterministic availability, such as shift schedule, operator break times, periodic inspections, etc.
Token	The basic component of <i>FlexSim's</i> Process Flow logic. Tokens are analogous to flowitems in 3D in that they flow through activities as a simulation runs. However, tokens are typically more abstract than flowitems since they usually represent logic flow, rather than physical flow. Each token has a set of labels or characteristics.
Toolbox	A type of library that contains modeling tools or aides, such as data tables, time tables, storage systems, etc.
Transfer	Objects that are automatically created in a model whenever a conveyor segment is connected to another conveyor segment or to another type of fixed resource.
Tree	The hierarchical, object-oriented data structure that stores the model data. In <i>FlexSim</i> , all of this data is readily accessible.
Trigger	A place in a model, typically on an object, where optional actions can be defined to occur when certain conditions are met as a model runs, such as: when a flowitem enters/exits an object, a process is finished, a message is received, the model is reset. There are many types of actions that can occur when a trigger is "fired," such as: change an item's appearance (shape, color, etc.), read/write a table or label value, open/close ports, send a message, etc.
Units	Units of measure - time, length, and volume (for fluid-based models only) - that must be specified at the beginning of a simulation.
Validation	The process of determining if a model is accurate enough for its intended use.
Verification	The process of determining if a model has been implemented correctly.
Warm up	A period of time at the beginning of a simulation where statistics are not collected. This allows the simulation to get to a point where the conditions are more representative of those that would occur in the real system. For example, a simulation may start with all resources empty and idle, which may not be typical in the real system, and thus the statistics would be biased low if this early time is included in an analysis.

ABOUT THE AUTHOR

Allen G. Greenwood, Ph.D., P.E.



Currently, Allen is both a Simulation Education Specialist at FlexSim Software Products, Inc. and Professor Emeritus of Industrial and Systems Engineering at Mississippi State University. One of his long-term, foundational professional goals, both in practice and in academe, has been to enhance and increase the application of simulation to support the problem solving and decision making processes.

In addition to his faculty positions at Mississippi State, he was Professor of Engineering Management at Poznan University of Technology in Poland, Professor and Chair of the Department of Engineering Management at Prince Sultan University in the Kingdom of Saudi Arabia, Professor of Engineering at the American University of Armenia, and Assistant Professor of Management Sciences at Northeastern University and Virginia Tech.

At all of these institutions, he developed and taught courses in systems simulation at the undergraduate, graduate, and professional levels. In addition, he taught courses in operations research / management science, logistics systems design, enterprise systems engineering, project management, statistics, decision analysis, information systems, etc.

Allen's research interests/expertise include the design and analysis of production and project systems; simulation modeling, analysis, and optimization; and the design and application of decision-support systems.

Prior to academe, he held engineering and supervisory positions at American Enka Company and General Dynamics Corporation. During his academic career, he has led or was a principal contributor to numerous projects in industry, mostly in the area of systems simulation. As a result, his professional experience spans a wide variety of domains -- engineering design and development (military aircraft and aerospace), manufacturing and production systems (military aircraft, shipbuilding, automotive, textile fibers, healthcare, electrical systems, material handling systems, consumer products, etc.), and project management. His work has been funded by such organizations as the US Air Force Research Laboratory, Office of Naval Research, Naval Sea Systems Command, NASA, Northrop Grumman Ship Systems, Nissan North America, General Electric Aviation, the Center for Advanced Vehicular Systems (MSU), and Poznan University of Technology.

He has authored or co-authored over 150 creative works, including journal and conference papers, technical reports, software programs, etc. In addition, he is co-author of *Applied Simulation: Modeling and Analysis Using FlexSim*, currently in its fifth edition, and recently authored two primers, one focused on simulation in general and one focused on using *FlexSim* software.

Allen received his B.S.I.E, M.S.I.E, and Ph.D. (Management Science) degrees from North Carolina State University, University of Tennessee, and Virginia Tech, respectively. He is a registered Professional Engineer (retired) in Texas.

Allen may be contacted at allen.greenwood@flexsim.com.