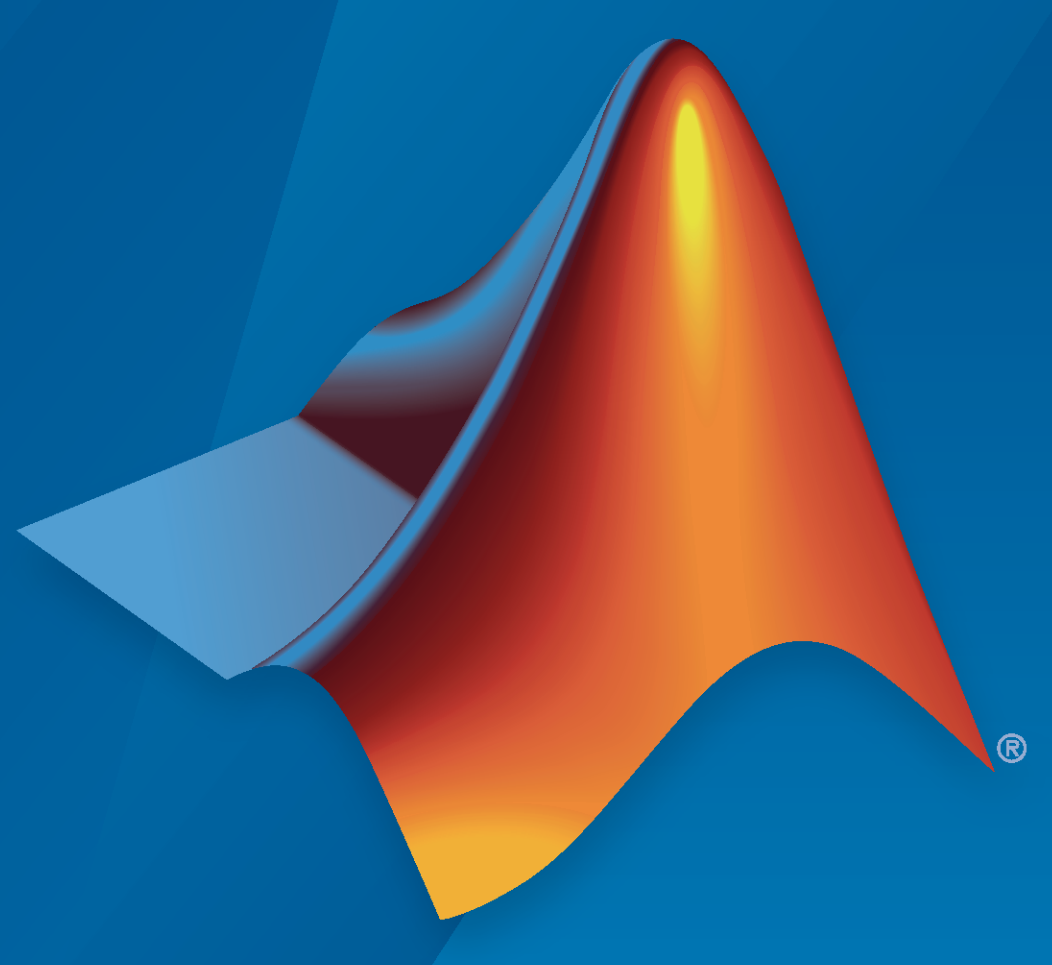
# SimEvents®

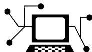
User's Guide



MATLAB® & SIMULINK®

R2021a

## How to Contact MathWorks



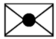
Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

Technical support: www.mathworks.com/support/contact_us

Phone: 508-647-7000



The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

*SimEvents® User's Guide*

© COPYRIGHT 2005–2021 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

**Trademarks**

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

**Patents**

MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

**Revision History**

| | | |
|---|---|---|
| November 2005 | Online only | New for Version 1.0 (Release 14SP3+) |
| March 2006 | Online only | Revised for Version 1.1 (Release 2006a) |
| September 2006 | Online only | Revised for Version 1.2 (Release 2006b) |
| March 2007 | Online only | Revised for Version 2.0 (Release 2007a) |
| September 2007 | Online only | Revised for Version 2.1 (Release 2007b) |
| March 2008 | Online only | Revised for Version 2.2 (Release 2008a) |
| October 2008 | Online only | Revised for Version 2.3 (Release 2008b) |
| March 2009 | Online only | Revised for Version 2.4 (Release 2009a) |
| September 2009 | Online only | Revised for Version 3.0 (Release 2009b) |
| March 2010 | Online only | Revised for Version 3.1 (Release 2010a) |
| September 2010 | Online only | Revised for Version 3.1.1 (Release 2010b) |
| April 2011 | Online only | Revised for Version 3.1.2 (Release 2011a) |
| September 2011 | Online only | Revised for Version 4.0 (Release 2011b) |
| March 2012 | Online only | Revised for Version 4.1 (Release 2012a) |
| September 2012 | Online only | Revised for Version 4.2 (Release 2012b) |
| March 2013 | Online only | Revised for Version 4.3 (Release 2013a) |
| September 2013 | Online only | Revised for Version 4.3.1 (Release 2013b) |
| March 2014 | Online only | Revised for Version 4.3.2 (Release 2014a) |
| October 2014 | Online only | Revised for Version 4.3.3 (Release 2014b) |
| March 2015 | Online only | Revised for Version 4.4 (Release 2015a) |
| September 2015 | Online only | Revised for Version 4.4.1 (Release 2015b) |
| March 2016 | Online only | Revised for Version 5.0 (Release 2016a) |
| September 2016 | Online only | Revised for Version 5.1 (Release 2016b) |
| March 2017 | Online only | Revised for Version 5.2 (Release 2017a) |
| September 2017 | Online only | Revised for Version 5.3 (Release 2017b) |
| March 2018 | Online only | Revised for Version 5.4 (Release 2018a) |
| September 2018 | Online only | Revised for Version 5.5 (Release 2018b) |
| March 2019 | Online only | Revised for Version 5.6 (Release 2019a) |
| September 2019 | Online only | Revised for Version 5.7 (Release 2019b) |
| March 2020 | Online only | Revised for Version 5.8 (Release 2020a) |
| September 2020 | Online only | Revised for Version 5.9 (Release 2020b) |
| March 2021 | Online only | Revised for Version 5.10 (Release 2021a) |

# Contents

## 1 Working with Entities

**Events and Event Actions** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **1-2**
Overview of Events . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **1-2**
Write Custom Code for Event Actions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **1-2**
SimEvents Blocks that Include Event Actions . . . . . . . . . . . . . . . . . . . . . . . **1-3**
Using the Event Actions Assistant . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **1-4**
Track Events with Event Calendar . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **1-6**
Visualize Event Actions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **1-6**
Preventing Livelock for Large Finite Numbers of Simultaneous Events . . . **1-7**

**Event Action Languages and Random Number Generation** . . . . . . . . . . . . **1-8**
Guidelines for Using MATLAB as the Event Action Language . . . . . . . . . . **1-8**
Generate Random Numbers with Event Actions . . . . . . . . . . . . . . . . . . . . **1-8**
Parameters in Event Actions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **1-11**

**Generate Entities When Events Occur** . . . . . . . . . . . . . . . . . . . . . . . . . . . **1-13**
Generate Entity When First Entity is Destroyed . . . . . . . . . . . . . . . . . . . **1-13**
Generate Event-Based Entities Using Data Sets . . . . . . . . . . . . . . . . . . . **1-14**

**Specify Intergeneration Times for Entities** . . . . . . . . . . . . . . . . . . . . . . . **1-16**
Determine Intergeneration Time . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **1-16**

**Generate Multiple Entities at Time Zero** . . . . . . . . . . . . . . . . . . . . . . . . . **1-21**
Build the model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **1-21**

**Adjust Entity Generation Times Through Feedback** . . . . . . . . . . . . . . . . . **1-24**

**Count Simultaneous Departures from a Server** . . . . . . . . . . . . . . . . . . . . **1-27**

**Noncumulative Counting of Entities** . . . . . . . . . . . . . . . . . . . . . . . . . . . . **1-29**

**Working with Entity Attributes and Entity Priorities** . . . . . . . . . . . . . . . **1-32**
Attach Attributes to Entities . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **1-32**
Set Attributes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **1-33**
Use Attributes to Route Entities . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **1-35**
Entity Priorities . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **1-36**

**Inspect Structures of Entities** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **1-37**
Display Entity Types . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **1-37**
Inspect Entities at Run Time . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **1-38**

**Generate Entities Carrying Nested Data Structures** . . . . . . . . . . . . . . . . **1-39**

**Model Resource Allocation Using Composite Entity Creator block** . . . . **1-44**

**Replicate Entities on Multiple Paths** . . . 1-45
Modeling Notes . . . 1-45

**Measure Point-to-Point Delays** . . . 1-46
Basic Example Using Timer Blocks . . . 1-46

## 2 Modeling Queues and Servers

**Model Basic Queuing Systems** . . . 2-2
Sort Entities Using the Entity Queue Block . . . 2-2
Queue Entity Overwriting Policies . . . 2-6
Customize Entity Service Time . . . 2-8
Build a Simple Queuing System to Change Entity Attributes . . . 2-10
Analyze Queue Length Using Statistics and Logical Queues . . . 2-12

**Broadcast Entities using Entity Multicasting** . . . 2-16

**Use Queue Event Actions to Model a Storage Tank** . . . 2-20

**Task Preemption in a Multitasking Processor** . . . 2-24
Example Model for Task Preemption . . . 2-24
Model Behavior and Results . . . 2-24

**Model Server Failure** . . . 2-27
Server States . . . 2-27
Use a Gate to Implement a Failure State . . . 2-27

**Serve High-Priority Customers by Sorting Entities Based on Priority** . . . 2-29

## 3 Routing Techniques

**Route Vehicles Using an Entity Output Switch Block** . . . 3-2

**Control Output Switch with Event Actions and Simulink Function** . . . 3-5
Control Output Switch with a Simulink Function Block . . . 3-5
Specify an Initial Port Selection . . . 3-6

**Match Entities Based on Attributes** . . . 3-7

**Role of Gates in SimEvents Models** . . . 3-9
Overview of Gate Behavior . . . 3-9
Gate Behavior . . . 3-9

**Enable a Gate for a Time Interval** . . . 3-11
Behavior of Entity Gate Block in Enabled Mode . . . 3-11
Sense an Entity Passing from A to B and Open a Gate . . . 3-11
Control Joint Availability of Two Servers . . . 3-13

Modeling Message Communication Patterns with SimEvents . . . . . . . . . 3-15

Build a Shared Communication Channel with Multiple Senders and Receivers . . . . . . . . . 3-17

Model an Ethernet Communication Network with CSMA/CD Protocol . . 3-22

## 4 Work with Resources

**Model Using Resources** . . . . . . . . . 4-2
Resource Blocks . . . . . . . . . 4-2
Resource Creation Workflow . . . . . . . . . 4-2

**Set Resource Amount with Attributes** . . . . . . . . . 4-4

**Group Entities Using Batching** . . . . . . . . . 4-6

**Find and Extract Entities in SimEvents Models** . . . . . . . . . 4-10
Finding and Examining Entities . . . . . . . . . 4-10
Extracting Found Entities . . . . . . . . . 4-13
Changing Found Entity Attributes . . . . . . . . . 4-16
Triggering Entity Find Block with Event Actions . . . . . . . . . 4-16
Building a Firewall and an Email Server . . . . . . . . . 4-18

## 5 Visualization, Statistics, and Animation

**Interpret SimEvents Models Using Statistical Analysis** . . . . . . . . . 5-2
Output Statistics for Data Analysis . . . . . . . . . 5-2
Output Statistics for Run-Time Control . . . . . . . . . 5-2
Average Queue Length and Average Store Size . . . . . . . . . 5-4
Average Wait . . . . . . . . . 5-6
Number of Entities Arrived . . . . . . . . . 5-8
Number of Entities Departed . . . . . . . . . 5-8
Number of Entities Extracted . . . . . . . . . 5-8
Number of Entities in Block . . . . . . . . . 5-8
Number of Pending Entities . . . . . . . . . 5-8
Pending Entity Present in Block . . . . . . . . . 5-9
Utilization . . . . . . . . . 5-9

**Visualization and Animation for Debugging** . . . . . . . . . 5-10
Which Debugging Tool to Use . . . . . . . . . 5-10
Observe Entities with Animation . . . . . . . . . 5-11
Explore the System Using the Simulink Simulation Stepper . . . . . . . . . 5-11
Information About Race Conditions and Random Times . . . . . . . . . 5-11

**Model Traffic Intersections as a Queuing Network** . . . . . . . . . 5-12

**Optimize SimEvents Models by Running Multiple Simulations** . . . . . . . **5-20**
Grocery Store Model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **5-20**
Build the Model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **5-20**
Run Multiple Simulations to Optimize Resources . . . . . . . . . . . . . . . . . **5-22**

**Use the Sequence Viewer to Visualize Messages, Events, and Entities** . **5-24**
Components of the Sequence Viewer Window . . . . . . . . . . . . . . . . . . . . **5-25**
Navigate the Lifeline Hierarchy . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **5-27**
View State Activity and Transitions . . . . . . . . . . . . . . . . . . . . . . . . . . . . **5-29**
View Function Calls . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **5-30**
Simulation Time in the Sequence Viewer Window . . . . . . . . . . . . . . . . . **5-31**
Redisplay of Information in the Sequence Viewer Window . . . . . . . . . . . **5-32**

## 6 Learning More About SimEvents Software

**Event Calendar** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **6-3**

**Save SimEvents Simulation Operating Point** . . . . . . . . . . . . . . . . . . . . . . **6-4**

**Example Model to Count Simultaneous Departures from a Server** . . . . . . **6-9**

**Example Model for Noncumulative Entity Count** . . . . . . . . . . . . . . . . . . **6-10**

**Adjust Entity Generation Times Through Feedback** . . . . . . . . . . . . . . . . **6-11**

**A Simple Example of Generating Multiple Entities** . . . . . . . . . . . . . . . . **6-14**

**A Simple Example of Event-Based Entity Generation** . . . . . . . . . . . . . . . **6-15**

**Serve Preferred Customers First** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **6-16**

**Find and Examine Entities** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **6-17**

**Extract Found Entities** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **6-20**

**Trigger Entity Find Block with Event Actions** . . . . . . . . . . . . . . . . . . . . **6-21**

**Build a Firewall and an Email Server** . . . . . . . . . . . . . . . . . . . . . . . . . . . **6-22**

**Implement the Custom Entity Storage Block** . . . . . . . . . . . . . . . . . . . . . **6-23**

**Implement the Custom Entity Storage Block with Iteration Event** . . . . . **6-24**

**Implement the Custom Entity Storage Block with Two Timer Events** . . **6-25**

**Implement the Custom Entity Generator Block** . . . . . . . . . . . . . . . . . . . **6-26**

**Implement the Custom Entity Storage Block with Two Storages** . . . . . . **6-27**

**Generating and Initializing Entities** . . . . . . . . . . . . . . . . . . . . . . . . . . . . **6-28**

M/M/1 Queuing System . . . 6-37

M/D/1 Queuing System . . . 6-41

G/G/1 Queuing System and Little's Law . . . 6-44

Comparing Queuing Strategies . . . 6-47

Modeling Hybrid Systems - Tank Filling . . . 6-51

Resource Allocation from Multiple Pools . . . 6-55

Using Entity Priority to Sequence Departures . . . 6-60

Using Custom Visualization for Entities . . . 6-62

Selection Server - Select Specific Entities from Server . . . 6-65

Flush Entities from a Queue-Server . . . 6-67

Server with Pause/Continue . . . 6-70

Simulation of a Medical Device . . . 6-72

Dining Philosophers Problem . . . 6-77

Simulate Scheduler of a Multicore Control System . . . 6-81

Develop Custom Scheduler of a Multicore Control System . . . 6-86

Distributing Multi-Class Jobs to Service Stations . . . 6-94

Effects of Communication Delays on an ABS Control System . . . 6-96

Aircraft Boarding Process Flow . . . 6-100

Optimization of Shared Resources in a Batch Production Process . . . 6-102

Modeling a Kanban Production System . . . 6-112

Job Scheduling and Resource Estimation for a Manufacturing Plant . 6-122

Modeling Load Within a Dynamic Voltage Scaling Application . . . 6-137

Modeling Machine Failure . . . 6-140

Inventory Management . . . 6-145

Modeling Cyber-Physical Systems . . . 6-148

802.11 MAC and Application Throughput Measurement . . . 6-153

802.11ax System-Level Simulation with Physical Layer Abstraction . . 6-169

## 7 Use SimEvents with Simulink

**Working with SimEvents and Simulink** . . . . . . . . . . . . **7-2**
Exchange Data Between SimEvents and Simulink . . . . . . . . . **7-2**
Time-Based Signals and SimEvents Block Transitions . . . . . . . **7-2**
SimEvents Support for Simulink Subsystems . . . . . . . . . . **7-2**
Save Simulation Data . . . . . . . . . . . . . . . . . . **7-3**

**Solvers for Discrete-Event Systems** . . . . . . . . . . . . **7-5**
Variable-Step Solvers for Discrete-Event Systems . . . . . . . . **7-5**
Fixed-Step Solvers for Discrete-Event Systems . . . . . . . . . **7-5**

**Model Simple Order Fulfilment Using Autonomous Robots** . . . . . **7-7**
Order Fulfilment Model . . . . . . . . . . . . . . . . . **7-7**
Warehouse Component . . . . . . . . . . . . . . . . . . **7-8**
Order Queue Component . . . . . . . . . . . . . . . . . **7-12**
Results . . . . . . . . . . . . . . . . . . . . . . . **7-12**

## 8 Build Discrete-Event Systems Using Charts

**Create Custom Queuing Systems Using Discrete-Event Stateflow Charts** . . . . . . . . . . . . . . . . . . . . . . . **8-2**
Properties of Discrete-Event Chart . . . . . . . . . . . . . **8-2**
Define Local Messages . . . . . . . . . . . . . . . . . . **8-3**
Specify Message Properties . . . . . . . . . . . . . . . . **8-4**
Event Triggering . . . . . . . . . . . . . . . . . . . . **8-4**
Message Triggering . . . . . . . . . . . . . . . . . . . **8-4**
Temporal Triggering . . . . . . . . . . . . . . . . . . **8-5**

**Discrete-Event Chart Precise Timing** . . . . . . . . . . . . **8-6**

**Trigger a Discrete-Event Chart Block on Message Arrival** . . . . . **8-9**

**Dynamic Scheduling of Discrete-Event Chart Block** . . . . . . . **8-18**

## 9 Build Discrete-Event Systems Using System Objects

**Create Custom Blocks Using MATLAB Discrete-Event System Block** . . . . **9-2**
Entity Types, Ports, and Storage in a Discrete-Event System Framework . . . . . . . . . . . . . . . . . . . . . . . **9-2**
Events . . . . . . . . . . . . . . . . . . . . . . . . **9-5**
Implement a Discrete-Event System Object with MATLAB Discrete-Event System Block . . . . . . . . . . . . . . . . . . . . **9-6**

**Delay Entities with a Custom Entity Storage Block** . . . . . . . . . . . . . . . . . **9-9**
Create the Discrete-Event System Object . . . . . . . . . . . . . . . . . . . . . . . . **9-9**
Implementing the Custom Entity Storage Block . . . . . . . . . . . . . . . . . . **9-11**

**Create a Custom Entity Storage Block with Iteration Event** . . . . . . . . . . **9-14**
Create the Discrete-Event System Object . . . . . . . . . . . . . . . . . . . . . . . . **9-14**
Define Custom Block Behavior . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **9-15**
Implement Custom Block . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **9-16**

**Custom Entity Storage Block with Multiple Timer Events** . . . . . . . . . . . . **9-19**
Create the Discrete-Event System Object with Multiple Timer Events . . . **9-19**
Custom Block Behavior . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **9-20**
Implement Custom Block . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **9-21**

**Custom Entity Generator Block with Signal Input and Signal Output** . . **9-24**
Create the Discrete-Event System Object . . . . . . . . . . . . . . . . . . . . . . . . **9-24**
Custom Block Behavior . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **9-26**
Implement Custom Block . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **9-27**

**Build a Custom Block with Multiple Storages** . . . . . . . . . . . . . . . . . . . . **9-31**
Create the Discrete-Event System Object . . . . . . . . . . . . . . . . . . . . . . . . **9-31**
Custom Block Behavior . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **9-33**
Implement the Custom Block . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **9-34**

**Create a Custom Resource Acquirer Block** . . . . . . . . . . . . . . . . . . . . . . . **9-38**
Create the Discrete-Event System Object . . . . . . . . . . . . . . . . . . . . . . . . **9-38**
Custom Block Behavior . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **9-39**
Implement the Custom Block . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **9-40**

**Create a Discrete-Event System Object** . . . . . . . . . . . . . . . . . . . . . . . . . **9-44**
Methods . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **9-44**
Inherited Methods from matlab.System Class . . . . . . . . . . . . . . . . . . . . . **9-46**
Reference and Extract Entities . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **9-47**

**Generate Code for MATLAB Discrete-Event System Blocks** . . . . . . . . . . . **9-48**
Migrate Existing MATLAB Discrete-Event System System object . . . . . . **9-48**
Limitations of Code Generation with Discrete-Event System Block . . . . . **9-50**

**Customize Discrete-Event System Behavior Using Events and Event Actions** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **9-51**
Event Types and Event Actions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **9-51**
Event Identifiers . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **9-53**

**Call Simulink Function from a MATLAB Discrete-Event System Block** . **9-55**
Modify Entity Attributes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **9-56**
Build the Model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **9-56**

**Resource Scheduling Using MATLAB Discrete-Event System and Data Store Memory Blocks** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **9-58**

## 10 Custom Visualization


**Use SimulationObserver Class to Monitor a SimEvents Model** . . . . . . . . **10-2**
SimulationObserver Class . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **10-2**
Custom Visualization Workflow . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **10-2**
Create an Application . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **10-2**
Use the Observer to Monitor the Model . . . . . . . . . . . . . . . . . . . . . . . . **10-4**
Stop Simulation and Disconnect the Model . . . . . . . . . . . . . . . . . . . . . **10-4**

**Observe Entities Using simevents.SimulationObserver Class** . . . . . . . . . **10-5**


## 11 Migrating SimEvents Models


**Migration Considerations** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **11-2**
When You Should Not Migrate . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **11-3**

**Migration Workflow** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **11-4**

**Identify and Redefine Entity Types** . . . . . . . . . . . . . . . . . . . . . . . . . . . **11-6**

**Replace Old Blocks** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **11-8**

**Connect Signal Ports** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **11-11**
If Connected to Gateway Blocks . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **11-11**
If Using Get Attribute Blocks to Observe Output . . . . . . . . . . . . . . . . . **11-11**
If Connected to Computation Blocks . . . . . . . . . . . . . . . . . . . . . . . . . . **11-12**
If Connected to Reactive Ports . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **11-13**

**Write Event Actions for Legacy Models** . . . . . . . . . . . . . . . . . . . . . . . . **11-15**
Replace Set Attribute Blocks with Event Actions . . . . . . . . . . . . . . . . . **11-15**
Get Attribute Values . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **11-16**
Replace Random Number Distributions in Event Actions . . . . . . . . . . . **11-16**
Replace Event-Based Sequence Block with Event Actions . . . . . . . . . . . **11-18**
Replace Attribute Function Blocks with Event Actions . . . . . . . . . . . . . **11-18**
If Using Simulink Signals in an Event-Based Computation . . . . . . . . . . **11-20**

**Observe Output** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **11-22**

**Reactive Ports** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **11-23**


## 12 Troubleshoot SimEvents Models


**Debug SimEvents Models** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **12-2**
Start the Debugger . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **12-3**
Step Through Model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **12-3**

# Working with Entities


- “Events and Event Actions” on page 1-2
- “Event Action Languages and Random Number Generation” on page 1-8
- “Generate Entities When Events Occur” on page 1-13
- “Specify Intergeneration Times for Entities” on page 1-16
- “Generate Multiple Entities at Time Zero” on page 1-21
- “Adjust Entity Generation Times Through Feedback” on page 1-24
- “Count Simultaneous Departures from a Server” on page 1-27
- “Noncumulative Counting of Entities” on page 1-29
- “Working with Entity Attributes and Entity Priorities” on page 1-32
- “Inspect Structures of Entities” on page 1-37
- “Generate Entities Carrying Nested Data Structures” on page 1-39
- “Model Resource Allocation Using Composite Entity Creator block” on page 1-44
- “Replicate Entities on Multiple Paths” on page 1-45
- “Measure Point-to-Point Delays” on page 1-46

# Events and Event Actions

**In this section...**


"Overview of Events" on page 1-2
"Write Custom Code for Event Actions" on page 1-2
"SimEvents Blocks that Include Event Actions" on page 1-3
"Using the Event Actions Assistant" on page 1-4
"Track Events with Event Calendar" on page 1-6
"Visualize Event Actions" on page 1-6
"Preventing Livelock for Large Finite Numbers of Simultaneous Events" on page 1-7


In a discrete-event simulation, an event is an instantaneous incident that may change a state variable, output, or the occurrence of other events. By using SimEvents, you can create custom actions that happen when an event occurs for an entity such as when an entity enters or exits a block.

## Overview of Events

In SimEvents, you can specify event actions based on entity status. A typical event sequence in a SimEvents model is:

1. An entity is generated.
2. The entity advances from an Entity Generator block to an Entity Server block.
3. The Entity Server block completes the service of an entity.
4. The entity exits Entity Server block and enters an Entity Terminator block.
5. The entity is destroyed.

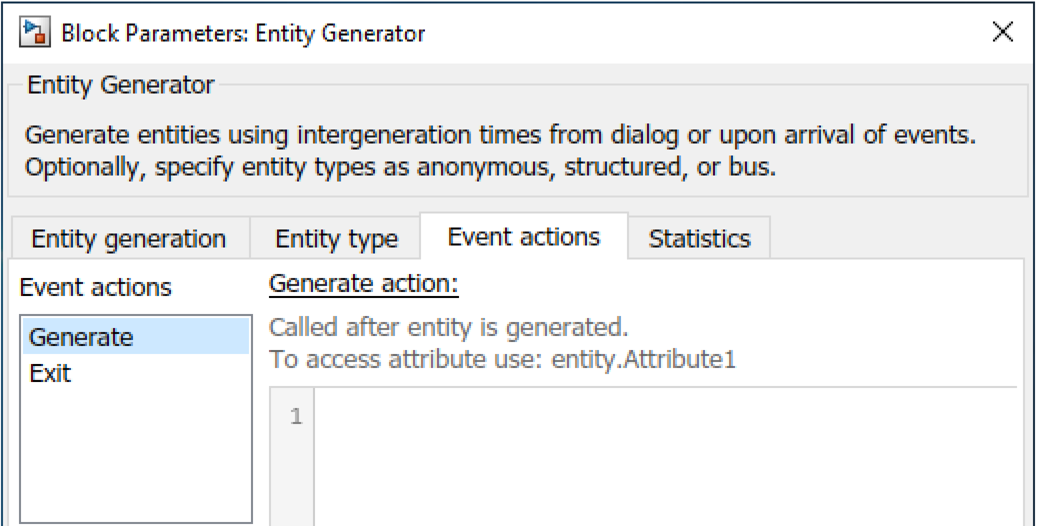
When an entity is created, enters or exits a block, or is serviced or destroyed, the entity changes status. You can use certain SimEvents library blocks to create event actions that trigger when these status changes occur. You can write event actions by using:

- MATLAB® code that performs calculations.
- Simulink® function calls that call a function that performs computations.

For more information about event action languages, see "Event Action Languages and Random Number Generation" on page 1-8.

## Write Custom Code for Event Actions

To create event action code and language, in a SimEvents block, select the **Event actions** tab and choose the event that invokes the action. For example, in the Entity Generator block, there are two events provided to invoke event actions, **Generate** and **Exit**. The event actions are triggered when an entity is generated or exits the block.




If you click the **Generate** event, you can write your code in the **Generate action** field.

When you use event actions:

- The entities are available as MATLAB structures and include structure fields that represent values of the entity attributes.
- Reserved fields, such as entity ID and entity priority, are also available in a separate MATLAB structure called `entitySys`.

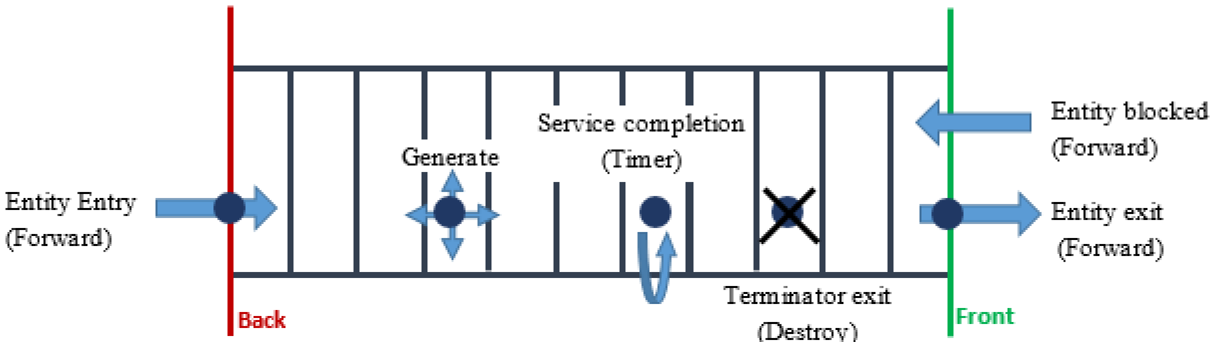
For an example of using event actions, see "Manage Entities Using Event Actions".

## SimEvents Blocks that Include Event Actions

You can see what event actions are available on the **Event actions** tab of a block. These are the possible events for which you can create actions.

| Entity Generator Block | Entity Queue Block | Entity Server Block | Entity Terminator Block | Resource Acquirer Block | Entity Batch Creator Block |
|---|---|---|---|---|---|
| Entity generation | Entity entry to queue block | Entity entry to server block | Entity entry to terminator block | Entity entry to acquirer block | Entity entry to batch block |
| Entity exit from block | Entity exit from block | Service completion of entity | N/A | Entity exit from acquirer block | Entity batch generation |
| N/A | Entity is blocked | Entity exit from block | N/A | Entity is blocked | Entity exit from block |
| N/A | N/A | Entity is blocked | N/A | N/A | Entity is blocked |
| N/A | N/A | Entity is preempted | N/A | N/A | N/A |

This illustration shows the flow of actions as entities move through a discrete-event system simulation.




Notes:

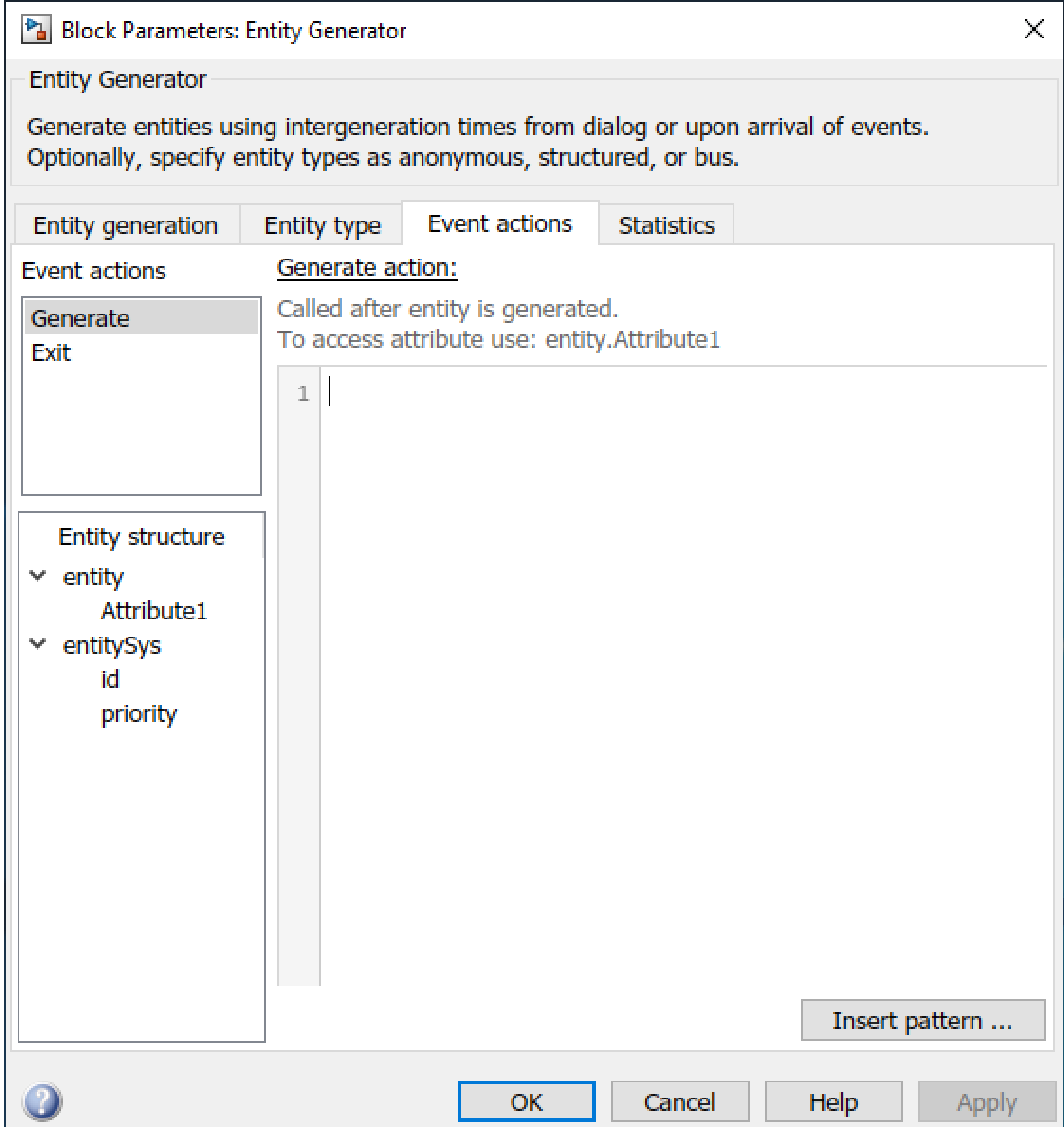
- Entity entry, exit, and blocking actions are performed as part of an entity forward event.
- Service completion action is performed following a timer event.
- Entity termination event performs a destruction action.

You can also modify the entity attributes (*entityName.attributeName*), entity priorities (`sys.entity.priority`), and entity IDs (`sys.entity.id`). However, you cannot change the entity attributes or system properties (`entitySys`) for exit actions. Attempting to change these values causes an error at simulation.

## Using the Event Actions Assistant

The Event Actions Assistant helps you create code for repeated sequence of event actions or random event actions according to a statistical distribution. For example, to access the assistant in an Entity Generator block:

1. Open the block and select the **Event actions** tab and select the **Generate** event action.
2. In the **Generate action** field, click the **Insert pattern** button.

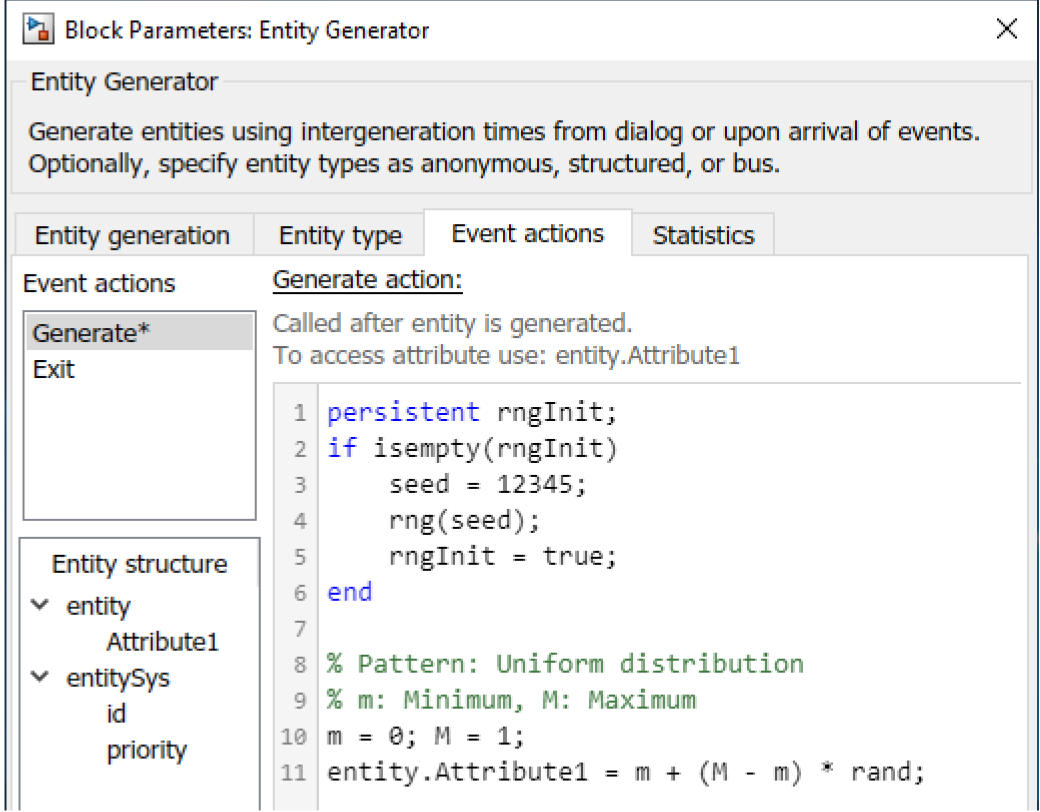



Suppose that you want to generate entities and assign random attribute values to them. The values are generated from a uniform distribution between `0` and `1`.

To achieve this behavior:

1. Select **Random number**.
2. To select a uniform distribution, set the **Distribution** parameter to `Uniform`.
3. By default, the **Minimum** and the **Maximum** parameters are specified as `0` and `1`, respectively.
4. To attach the values to the entity attribute `Attribute1`, set the **Assign output to** parameter to `entity.Attribute1`.

The assistant creates the code.




The code creates a persistent variable for the seed. Then a random value is attached to `entity.Attribute1`. After you define an action, an asterisk (*) appears in the Event actions tab to indicate that a code is called for that event. In this case, an asterisk is displayed after the **Generate** event action.

For more information on the event actions assistant, see “Event Action Languages and Random Number Generation” on page 1-8.

## Track Events with Event Calendar

SimEvents does not represent events graphically. Instead, the SimEvents software maintains an event calendar that schedules events. You can use the Event Calendar to observe events when you debug a SimEvents model. For more information, see “Debug SimEvents Models” on page 12-2.

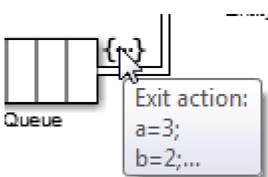
You can also interact with the event calendar by using `simevents.SimulationObserver` methods. You can create a custom event observer using this class and its methods. For more information, see “Use SimulationObserver Class to Monitor a SimEvents Model” on page 10-2.

## Visualize Event Actions

When you create an event action, `{...}` badge appears on the block to indicate that the action is created. The badges that appear depending on which event actions have associated code.

For instance, this illustration shows an Entity Queue block with event actions that are invoked by the entity entry and exit from the block.

When you hover over the badge, you can see the event action. For example, this illustration depicts an entity exit action.




Double-clicking the badge directly opens the **Event actions** tab of the block.

## Preventing Livelock for Large Finite Numbers of Simultaneous Events

Simultaneous events are events that occur at the same simulation clock time. Events scheduled on the event calendar at times *T* and *T*+Δt are considered simultaneous if 0 ≤ Δt ≤ `128*eps*`*T*, where `eps` is the floating-point relative accuracy in MATLAB software and *T* is the simulation time. If your simulation creates a large number of simultaneous events, this number might be an indication of an unwanted livelock situation. During a livelock situation, a block returns to the same state infinitely often at the same time instant. SimEvents software prevents livelock with these limits:

- SimEvents limits the maximum number of simultaneous events per block to 5,000.
- SimEvents limits the maximum number of simultaneous events per model to 100,000.

### See Also

Composite Entity Creator | Composite Entity Splitter | Discrete Event Chart | Entity Generator | Entity Queue | Entity Server | Entity Terminator

### Related Examples

- “Generate Entities When Events Occur” on page 1-13

### More About

- “Entities in a SimEvents Model”
- “Event Action Languages and Random Number Generation” on page 1-8
- “Event Calendar” on page 6-3

# Event Action Languages and Random Number Generation

| In this section... |
| --- |
| "Guidelines for Using MATLAB as the Event Action Language" on page 1-8 |
| "Generate Random Numbers with Event Actions" on page 1-8 |
| "Parameters in Event Actions" on page 1-11 |

You can write SimEvents actions using:

- MATLAB code — Use MATLAB. For information on guidelines for using MATLAB code as the event action language, see "Guidelines for Using MATLAB as the Event Action Language" on page 1-8
- Simulink functions — Use the Simulink Function block. The Simulink Function block does not accept entities as input.

## Guidelines for Using MATLAB as the Event Action Language

In general, using MATLAB as the SimEvents event action language follows the same rules as the use of MATLAB in the MATLAB Function block.

- Include a type prefix for identifiers of enumerated values — The identifier `TrafficColors.Red` is valid, but `Red` is not.
- Use the MATLAB format for comments — Use `%` to specify comments for consistency with MATLAB. For example, the following comment is valid:

  ```
  % This is a valid comment in the style of MATLAB
  ```
- Use one-based indexing for vectors and matrices — One-based indexing is consistent with MATLAB syntax.
- Use parentheses instead of brackets to index into vectors and matrices — This statement is valid:

  ```
  a(2,5) = 0;
  ```

  This statement is not valid:

  ```
  a[2][5] = 0;
  ```
- Persistent variable guidelines:
  - Manage states that are not part of the entity structure using MATLAB persistent variables.
  - Persistent variables defined in any event action of a block are scoped to only that action.
  - Block can share persistent variables across all of its event action by managing it in a MATLAB function on path (that is invoked from its event actions).
  - Two different blocks cannot share the same persistent variable.
- Assign an initial value to local and output data — When using MATLAB as the action language, data read without an initial value causes an error.
- Do not use parameters that are of data type cell array.

## Generate Random Numbers with Event Actions

You can generate random numbers using various distributions. There are two modeling approaches to use seeds during random number generation.

- You can use persistent variables for initializing unique seeds for each block in your model.
- You can use `coder.extrinsic()` function to generate seeds without persistent variables.

To generate these random distributions, use code in the **Usage** column of this table in SimEvents blocks that support event actions or intergeneration time actions.

| Distribution | Parameters | Usage | Requires Statistics and Machine Learning Toolbox™ Product |
|---|---|---|---|
| Exponential | Mean (m) | `-m * log(1-rand)` | No |
| Uniform | Minimum (m)<br><br>Maximum (M) | `m + (M-m) * rand` | No |
| Bernoulli | Probability for output to be 1 (P) | `binornd(1,P)` | Yes |
| Binomial | Probability of success in a single trial (P)<br><br>Number of trials (N) | `binornd(N,P)` | Yes |
| Triangular | Minimum (m)<br><br>Maximum (M)<br><br>Mode (mode) | `persistent pd`<br>`if isempty(pd)`<br>`    pd = makedist('Triangular',...`<br>`    'a',m,'b',mode,'c',M)`<br>`end`<br>`random(pd)` | Yes |
| Gamma | Threshold (T)<br><br>Scale (a)<br><br>Shape (b) | `gamrnd(b,a)` | Yes |
| Gaussian (normal) | Mean (m)<br><br>Standard deviation (d) | `m + d*randn` | No |
| Geometric | Probability of success in a single trial (P) | `geornd(P)` | Yes |
| Poisson | Mean (m) | `poissrnd(m)` | Yes |
| Lognormal | Threshold (T)<br><br>Mu (mu)<br><br>Sigma (S) | `T + lognrnd(mu,S)` | Yes |
| Log-logistic | Threshold (T)<br><br>Scale (a) | `persistent pd`<br>`if isempty(pd)`<br>`    pd = makedist('Loglogistic',...`<br>`    'mu',m,'sigma',S);`<br>`end`<br>`random(pd)` | Yes |

| Distribution | Parameters | Usage | Requires Statistics and Machine Learning Toolbox™ Product |
|---|---|---|---|
| Beta | Minimum (m)<br><br>Maximum (M)<br><br>Shape parameter a (a)<br><br>Shape parameter b (b) | `betarnd(a,b)` | Yes |
| Discrete uniform | Minimum (m)<br><br>Maximum (M)<br><br>Number of values (N) | `persistent V P`<br>`if isempty(V)`<br>`    step = (M-m)/N;`<br>`    V = m : step : M;`<br>`    P = 0 : 1/N : N;`<br>`end`<br>`r = rand;`<br>`idx = find(r < P, 1);`<br>`V(idx)` | No |
| Weibull | Threshold (T)<br><br>Scale (a)<br><br>Shape (b) | `T + wblrnd(a,b)` | Yes |
| Arbitrary continuous | Value vector (V)<br><br>Cumulative probability function vector (P) | `r = rand;`<br>`if r == 0`<br>`    val = V(1);`<br>`else`<br>`    idx = find(r < P,1);`<br>`    val = V(idx-1) + ...`<br>`    (V(idx)-V(idx-1))*(r-P(idx-1));`<br>`end` | No |
| Arbitrary discrete | Value vector (V)<br><br>Probability vector (P) | `r = rand;`<br>`idx = find(r < cumsum(P),1);`<br>`V(idx)` | No |

For an example, see “Model Traffic Intersections as a Queuing Network” on page 5-12.

If you need additional random number distributions, see “Statistics and Machine Learning Toolbox”.

**Random Number Distribution with Persistent Variables**

To generate random numbers, initialize a unique seed for each block in your model. If you use a statistical pattern, you can manually change the initial seed to a unique value for each block to generate independent samples from the distributions.

To reset the initial seed value each time a simulation starts, use MATLAB code to initialize a persistent variable in event actions, for example:

```
persistent init
if isempty(init)
```

```
    rng(12234);
    init=true;
end
```

Here is an example code. The value vector is assigned to `FinalStop`:

```
% Set the initial seed.
persistent init
if isempty(init)
    rng(12234);
    init=true;
end
% Create random variable, x.
x=rand();
%
% Assign values within the appropriate range
% using the cumulative probability vector.
if x < 0.3
    entity.FinalStop = 2;
elseif x >= 0.3 && x< 0.6
    entity.FinalStop = 3;
elseif x >= 0.6 && x< 0.7
    entity.FinalStop = 4;
elseif x >= 0.7 && x< 0.9
    entity.FinalStop = 5;
else
    entity.FinalStop = 6;
end
```

#### Random Number Generation with Callbacks

In some scenarios, you generate random numbers without using the persistent variables. In this case, use `coder.extrinsic()` function to make sure that SimEvents is using the function in MATLAB and a seed is defined in the base workspace of MATLAB. This may cause performance decrease in simulation.

Consider this code as an example.

```
% Random number generation
coder.extrinsic('rand');
value = 1;
value = rand();
% Pattern: Exponential distribution
mu = 0.5;
dt = -1/mu * log(1 - value);
```

The output of the extrinsic function is an `mxArray`. To convert it to a known type, a variable `val = 1` is declared to set its type to double and `rand` is assigned to that variable `val=rand`. For information about extrinsic functions, see “Working with mxArrays”.

For an example, see “Model Traffic Intersections as a Queuing Network” on page 5-12.

## Parameters in Event Actions

From within an event action, you can refer to these parameters:

- Mask-specific parameters you define using the Mask Editor **Parameters** pane.
- Any variable you define in a workspace (such as base workspace or model workspace).
- Parameters you define using the `Simulink.Parameter` object.

---

**Note** With SimEvents actions, you cannot:

- Modify parameters from within an event action.
- Tune parameters during simulation.
- Event actions are not supported with string entity data type.

---

## See Also

Entity Generator | Entity Queue | Entity Replicator | Entity Server | Entity Terminator | MATLAB Function | Multicast Receive Queue | Resource Acquirer | Simulink Function | `Simulink.Parameter`

## Related Examples

- “Generate Entities When Events Occur” on page 1-13

## More About

- “Events and Event Actions” on page 1-2
- “Mask Editor Overview”

# Generate Entities When Events Occur



| In this section... |
| --- |
| "Generate Entity When First Entity is Destroyed" on page 1-13 |
| "Generate Event-Based Entities Using Data Sets" on page 1-14 |



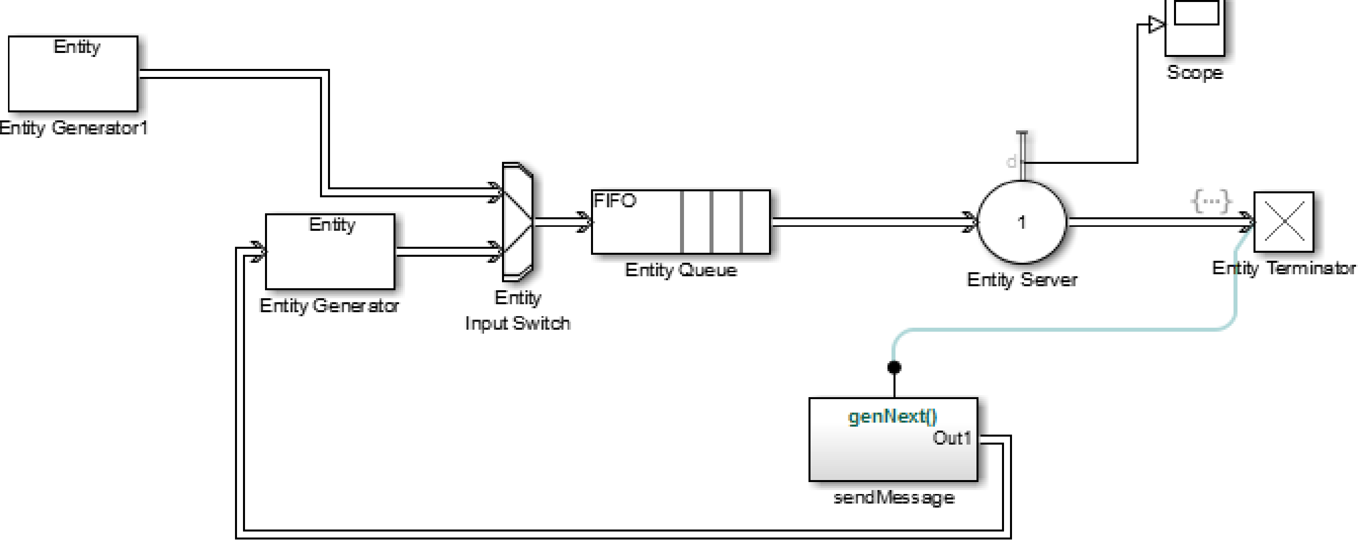
In addition to time-based entity generation, the Entity Generator block enables you to generate entities in response to events that occur during the simulation. In event-based generation, a new entity is generated whenever a message arrives at the input port of the Entity Generator block.

Event times and the time intervals between pairs of successive entities are not necessarily predictable in advance.

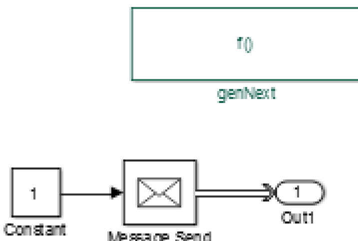
Generating entities when events occur is appropriate if you want the dynamics of your model to determine when to generate entities.

## Generate Entity When First Entity is Destroyed

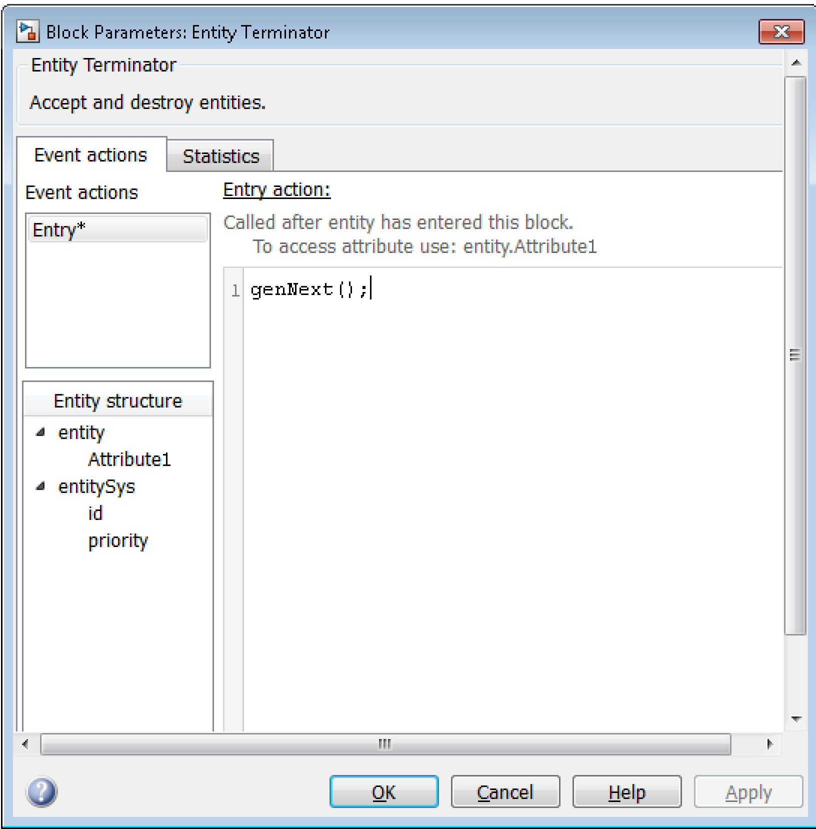
To generate an entity when the first entity is destroyed, use two Entity Generator blocks and a Simulink Function block. The Entity Terminator block calls the Simulink Function after destroying the first entity. For more information, see "Generating and Initializing Entities" on page 6-28.




In this example, Entity Generator1 generates the first entity. SendMessage contains the `genNext` function, which sends a message.




The Entity Terminator block calls the `genNext` function.




## Generate Event-Based Entities Using Data Sets

For an example that uses an Excel® spreadsheet, see “Generating and Initializing Entities” on page 6-28.

## See Also

Composite Entity Splitter | Discrete Event Chart | Entity Gate | Entity Generator | Entity Input Switch | Entity Multicast | Entity Output Switch | Entity Queue | Entity Replicator | Entity Server | Entity Terminator | MATLAB Discrete Event System | Multicast Receive Queue

## Related Examples


- “Specify Intergeneration Times for Entities” on page 1-16
- “Working with Entity Attributes and Entity Priorities” on page 1-32
- “Inspect Structures of Entities” on page 1-37
- “Generate Multiple Entities at Time Zero” on page 1-21
- “Count Simultaneous Departures from a Server” on page 1-27
- “Replicate Entities on Multiple Paths” on page 1-45


## More About


- “Entities in a SimEvents Model”
- “Events and Event Actions” on page 1-2

# Specify Intergeneration Times for Entities

The intergeneration time is the time interval between successive entities that the block generates. You can have a generation process that is:

- Periodic
- Sampled from a random distribution or time-based signal
- From custom code

For example, if the block generates entities at T = 50, T = 53, T = 60, and T = 60.1, the corresponding intergeneration times are 3, 7, and 0.1. After each new entity departs, the block determines the intergeneration time that represents the interval until the block generates the next entity.

## Determine Intergeneration Time

You configure the Entity Generator block by indicating criteria that it uses to determine intergeneration times for the entities it creates. You can generate entities:

- From random distribution
- Periodically
- At arbitrary times

Use the dropdown list in the **Time source** parameter of the Entity Generation block to determine intergeneration times:

- `Dialog`

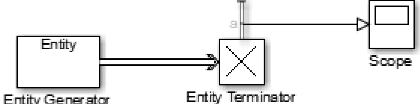
  Uses the **Period** parameter to periodically vary the intergeneration times.
- `Signal port`

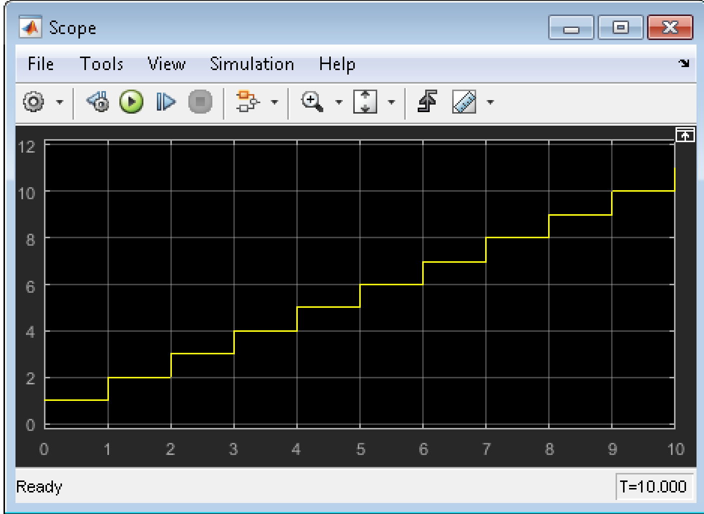
  Uses a signal from an external block, such as the Sine wave block, to vary the intergeneration times.
- `MATLAB action`

  Enables an **Intergeneration time action** field, in which you enter MATLAB code to customize the intergeneration times.

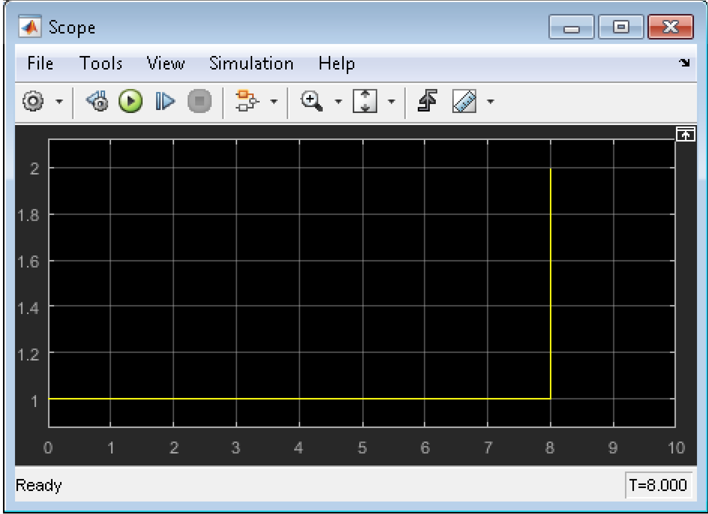
### Periodically Vary the Intergeneration Times

1. In a new model, from the SimEvents library, drag the Entity Generator, Entity Terminator, and Scope blocks.
2. In the **Entity Generation** tab of the Entity Generator, set the **Time source** parameter to `Dialog`.
3. In the **Statistics** tab of the Entity Terminator block, select the **Number of entities arrived** check box.
4. Connect these blocks and simulate the model. The period is 1.




5 Vary the period to 8 and simulate the model again. Observe the change in the scope.




### Use a Signal to Vary the Intergeneration Times

1 In a new model, from the SimEvents library, drag the Entity Generator and Entity Terminator blocks. From the Simulink library add the Sine Wave, and Scope blocks.

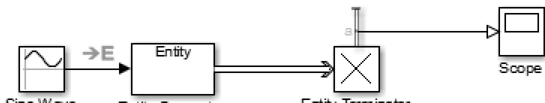
2 In the **Entity Generation** tab of the Entity Generator, set the **Time source** parameter to `Signal port`.

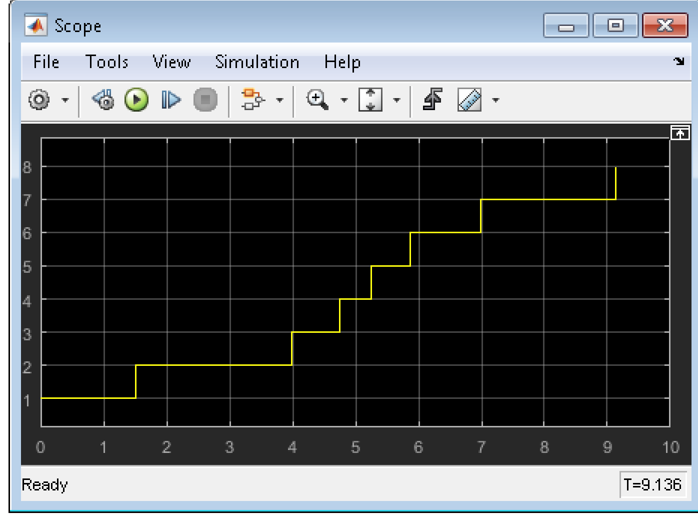
   A new signal port appears on the Entity Generator block.

3 In the **Statistics** tab of the Entity Terminator block, select the **Number of entities arrived** check box.

4 Double-click the Sine Wave block. By default, the first value of the Sine Wave block is `0`. To add a constant value to the sine to produce the output of this block, change the **Bias** parameter to another value, for example, `1.5`.

5 Connect these blocks and simulate the model.




Upon generating each entity, the Entity Generator block reads the value of the input signal and uses that value as the time interval until the next entity generation.

Notice the capital **E** on the signal line from the Sine Wave block to the **Entity Generator** block. This icon indicates the transition from a time-based system to a discrete-event system.

### Customize the Variation of the Intergeneration Times

1. In a new model, from the SimEvents library, drag the Entity Generator, Entity Terminator, and Scope blocks.
2. In the **Entity Generation** tab of the Entity Generator, set the **Time source** parameter to `MATLAB action`.

   A new **Intergeneration time action** field appears on the Entity Generator block.
3. To customize the intergeneration times for your model, in the **Intergeneration time action** field, enter MATLAB code, for example:

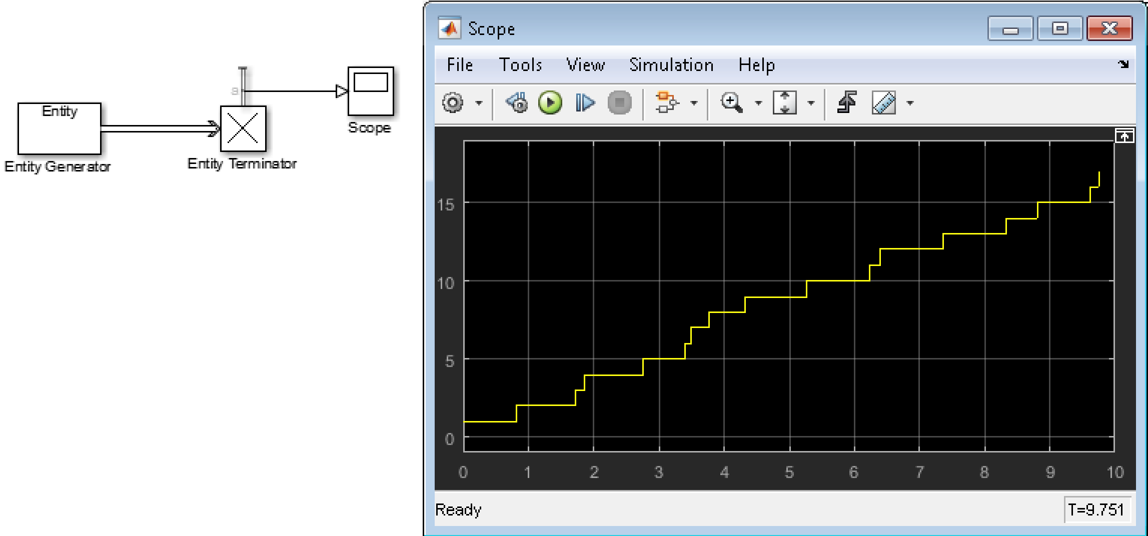
```
dt = rand();
```

---

**Note** For intergeneration times, you must set the fixed name, *dt*. You cannot set any other variable name for this value.

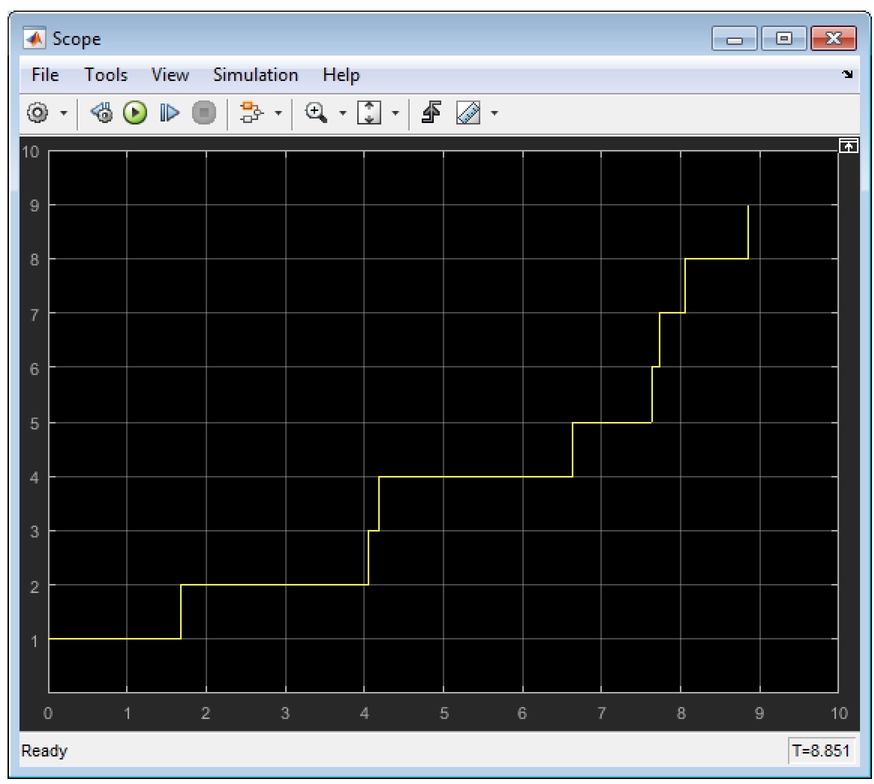
---

4. In the Statistics tab of the Entity Terminator block, select the **Number of entities arrived** check box.
5. Connect these blocks and simulate the model.




To generate entities with exponential random arrival times, in the **Intergeneration time action** field, enter MATLAB code that uses the `mean` function, for example:

```
mean = 1;
dt = -mean*log(1-rand());
```




## See Also

Discrete Event Chart | Entity Server | Entity Generator | Entity Queue | Entity Replicator | Entity Terminator | MATLAB Discrete Event System

## Related Examples


- “Generate Entities When Events Occur” on page 1-13
- “Working with Entity Attributes and Entity Priorities” on page 1-32
- “Inspect Structures of Entities” on page 1-37
- “Generate Multiple Entities at Time Zero” on page 1-21
- “Count Simultaneous Departures from a Server” on page 1-27
- “Model Resource Allocation Using Composite Entity Creator block” on page 1-44
- “Replicate Entities on Multiple Paths” on page 1-45


## More About


- “Entities in a SimEvents Model”
- “Role of Entity Ports and Paths”

# Generate Multiple Entities at Time Zero

In a discrete-event simulation, an event is an observation of an instantaneous incident that may change a state variable, an output, and/or the occurrence of other events.

Suppose that you want to:

- Preload a queue or server with entities at the start of the simulation, before you analyze queueing or processing delays.
- Initialize the capacity of a shared resource before you analyze resource allocation behavior.

These scenarios requires multiple entity generation at the simulation start.

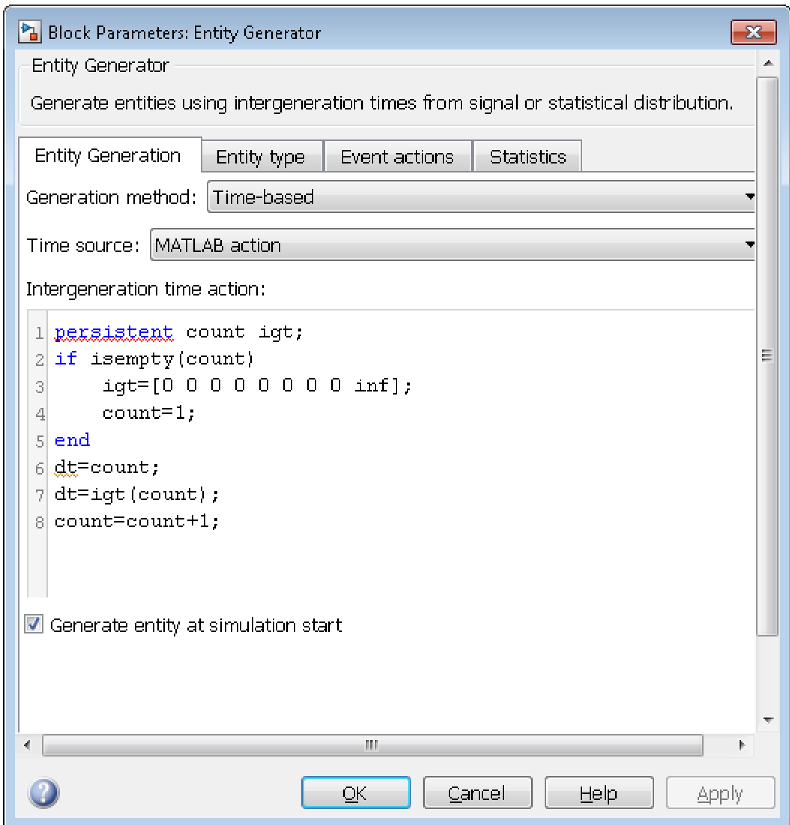
In these scenarios, you can simultaneously generate multiple entities at the start of the simulation. You can then observe the behavior of only those entities for the remainder of the simulation.

## Build the model

To generate multiple entities at time `0`, use MATLAB code in the Entity Generator block.

To open the example model without performing the configuration steps, see `A Simple Example of Generating Multiple Entities`.

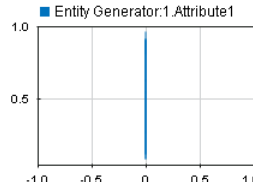
1. In a new model, from the SimEvents library, drag the Entity Generator, Entity Terminator, and Dashboard Scope blocks.
2. Double-click the Entity Generator block.
3. From the **Time source** drop-down list, select `MATLAB action`.
4. In the **Intergeneration time action** field, use MATLAB code to enter the number of entities that you want to generate. For example, you could use 8. In that case, at simulation time `0`, the Entity Generator block generates 8 simultaneous events.




5 In the **Events action** tab, randomize the entity attribute. Select the **Generate** event action and, in the **Generate action** field, enter the MATLAB code:

```
entity.Attribute1=rand();
```

The output of the Dashboard Scope block shows that the software generates multiple entities at time 0.




## See Also

Entity Generator | Entity Queue | Entity Server | Entity Terminator

## Related Examples

- “Generate Entities When Events Occur” on page 1-13
- “Specify Intergeneration Times for Entities” on page 1-16
- “Working with Entity Attributes and Entity Priorities” on page 1-32

- “Inspect Structures of Entities” on page 1-37
- “Count Simultaneous Departures from a Server” on page 1-27

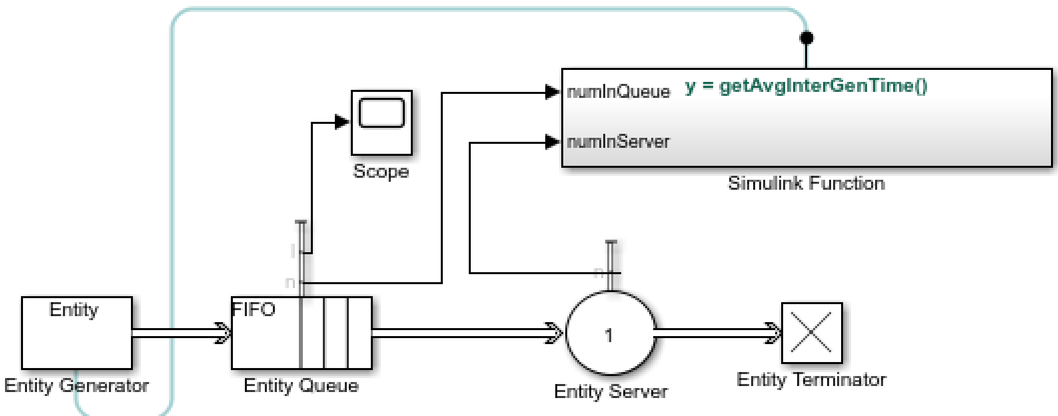

## More About


- “Entities in a SimEvents Model”

# Adjust Entity Generation Times Through Feedback

This example shows a queuing system in which feedback influences the arrival rate. The goal of the feedback loop is to stabilize the entity queue by slowing the entity generation rate of the Entity Generator block as more entities accumulate in the Entity Queue block and the Entity Server block.

The diagram shows a simple queuing system with an Entity Generator, an Entity Queue, an Entity Server, and an Entity Terminator block. For more information about building this simple queuing system, see “Create a Discrete-Event Model”.




The capacity of the Entity Server block is `1`. This causes an increase in the queue length without feedback. The goal is to regulate entity intergeneration time based on the size of the queue and the number of entities waiting to be served.

- In the Entity Generator block, select `MATLAB action` as the **Time source**. Add this code to the **Intergeneration time action** field.

```
persistent rngInit;

if isempty(rngInit)
    seed = 12345;
    rng(seed);
    rngInit = true;
end

% Pattern: Exponential distribution
mu = getAvgInterGenTime();
dt = -mu*log(1-rand());
```

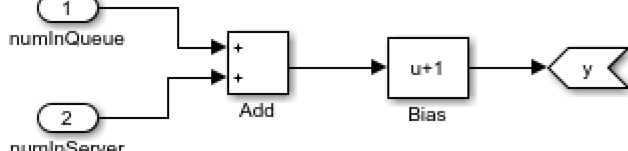
The entity intergeneration time `dt` is generated from an exponential distribution with mean `mu`, which is determined by the function `getAvgInterGenTime()`.

- In the Entity Queue block, in the **Statistics tab**, select the **Number of entities in block, n** and **Average queue length, l** as output statistics.

- In the Entity Server block, select `MATLAB action` as the **Service time source**. Add this code to the **Service time action** field.

```
persistent rngInit;
if isempty(rngInit)
    seed = 67868;
    rng(seed);
    rngInit = true;
end

% Pattern: Exponential distribution
mu = 3;
dt = -mu*log(1-rand());

The service time |dt| is drawn from an exponential distribution with
mean |3|.
```

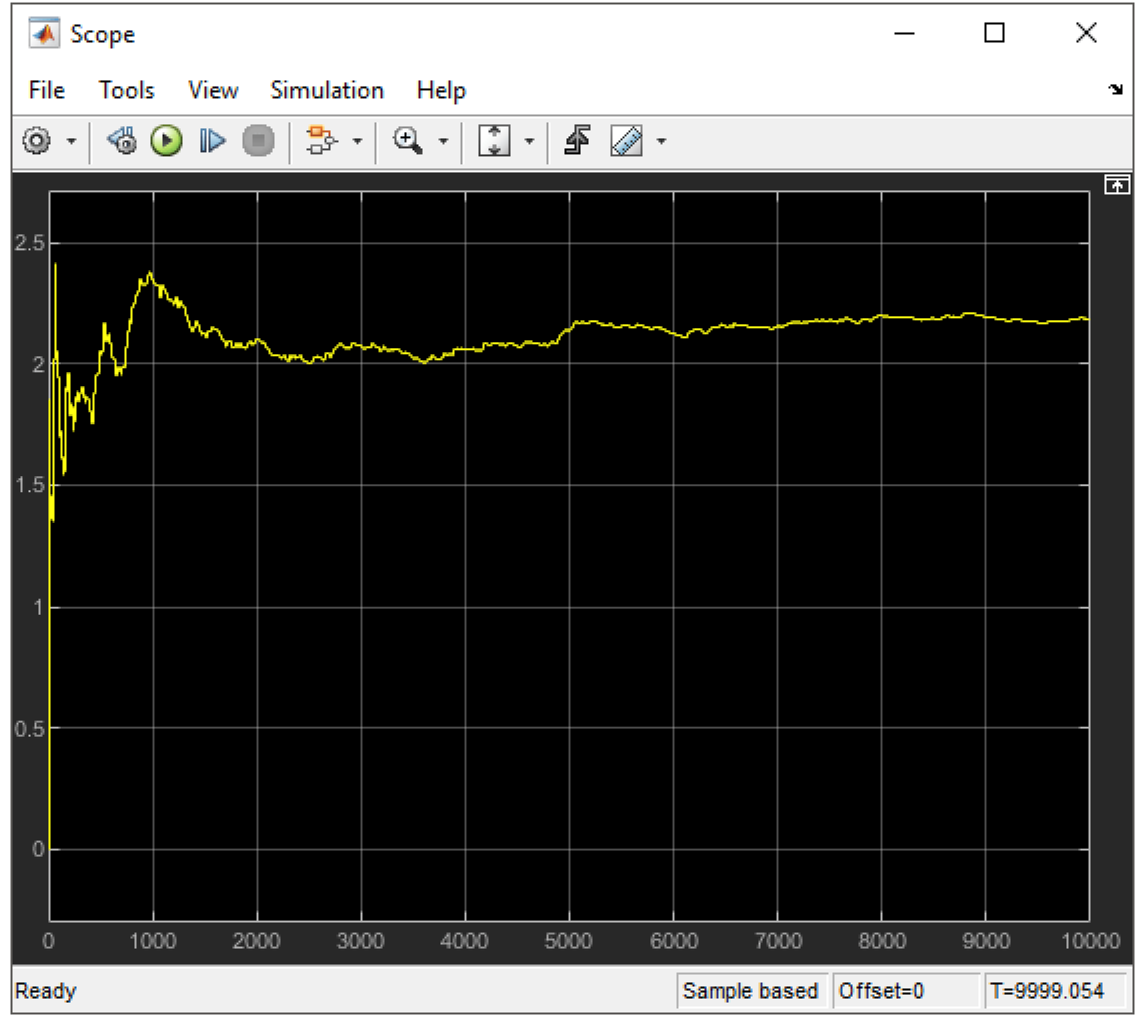
- In the Entity Server block, in the **Statistics tab**, select the **Number of entities in block, n** as output statistics.
- Add a Simulink Function block. On the Simulink Function block, double-click the function signature and enter `y = getAvgInterGenTime()`.
- In the Simulink Function block:




1. Add two In1 blocks and rename them as numInQueue and numInServer.
2. numInQueue represents the current number of entities accumulated in the queue and numInServer represents the current number of entities accumulated in the server.
3. Use Add block to add these two inputs.
4. Use a Bias block and set the `Bias` parameter as `1`. The constant bias `1` is to guarantee a nonzero intergeneration time.

Optionally, select **Function Connections** from the **Information Overlays** under the **Debug** tab to display the feedback loop from the Simulink Function block to the Entity Generation block.

- In the parent model, connect the **Number of entities in block, n** statistics from the Entity Queue and Entity Server blocks to the Simulink Function block.
- Connect a Scope block to the **Average queue length, l** statistic from the Entity Queue block. The goal is to investigate the average queue length.
- Increase the simulation time to `10000` and simulate the model.
- Observe that the **Average queue length, l** in the scope is nonincreasing due to the effect of feedback for the discouraged entity generation rate.




## See Also

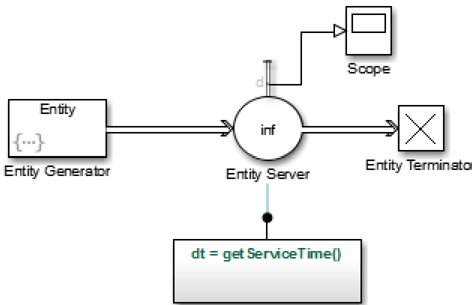
Entity Generator | Entity Queue | Entity Server | Entity Terminator

## Related Examples

- "Generate Entities When Events Occur" on page 1-13
- "Generate Multiple Entities at Time Zero" on page 1-21
- "Count Simultaneous Departures from a Server" on page 1-27
- "Replicate Entities on Multiple Paths" on page 1-45

# Count Simultaneous Departures from a Server

This example shows how to count the simultaneous departures of entities from a server. Use the **d** output from the Entity Server block to learn how many entities have departed (or arrived at) the block. The output signal also indicates when departures occurred. This method of counting is cumulative throughout the simulation.

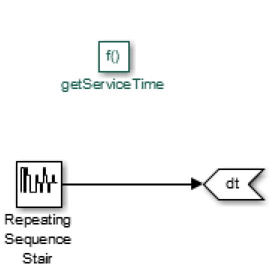


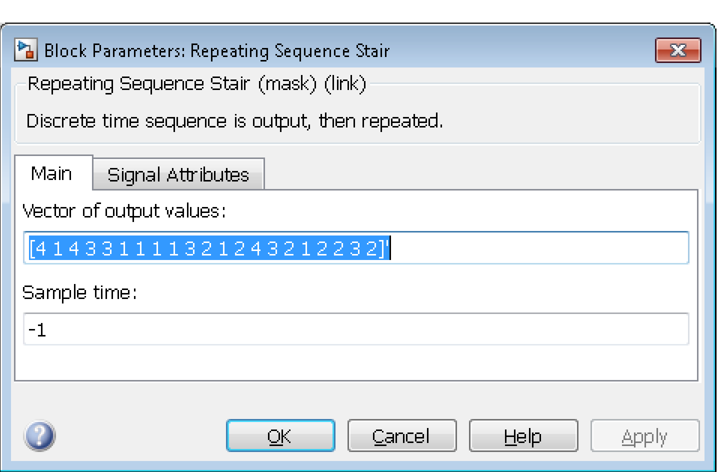

To open the example, see `Count Simultaneous Departures`.

1. In a new model, from the SimEvents library, drag the Entity Generator, Entity Server, Entity Terminator, and Simulink Function blocks. Add a Simulink Scope block.
2. Double-click the Entity Generator block.
   - In the **Event actions** tab, to generate random attribute values, enter:

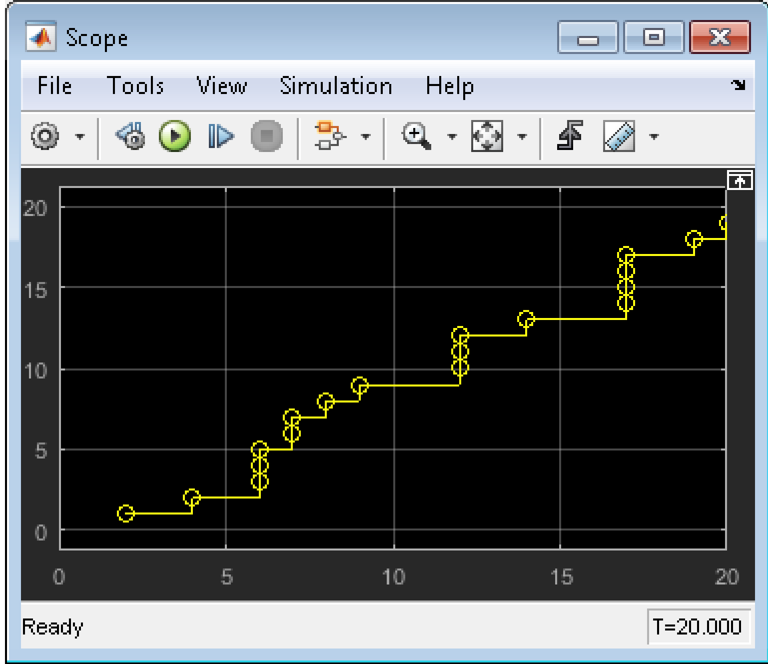
     ```
     entity.Attribute1=rand();
     ```
3. Double-click the Entity Server block. In the **Main** tab:
   - In the **Capacity** parameter, enter `inf`.
   - For the **Service time** parameter, select `MATLAB action`.
   - In the **Service time action** parameter, enter:

     ```
     dt = getServiceTime();
     ```
   - In the **Statistics** tab, select **Number of entities departed, d**.
4. In the Simulink Function block, add a Repeating Sequence Stair and define the `getServiceTime` function.



5 Connect the blocks as shown and simulate the model. Observe that the scope displays simultaneous entity departures for the corresponding time.




## See Also

Composite Entity Creator | Entity Gate | Entity Generator | Entity Multicast | Entity Queue | Entity Server | Entity Terminator | Resource Acquirer

## Related Examples

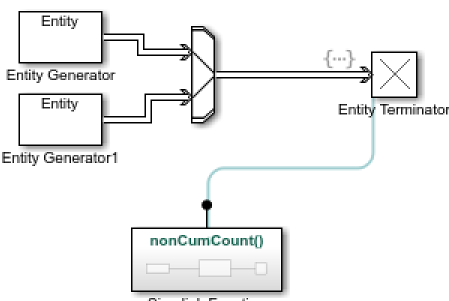
- "Generate Entities When Events Occur" on page 1-13
- "Specify Intergeneration Times for Entities" on page 1-16
- "Working with Entity Attributes and Entity Priorities" on page 1-32
- "Generate Multiple Entities at Time Zero" on page 1-21
- "Replicate Entities on Multiple Paths" on page 1-45

## More About

- "Entities in a SimEvents Model"

# Noncumulative Counting of Entities

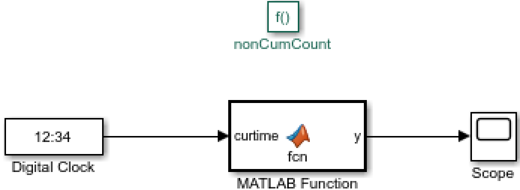
This example shows how to count entities, which arrive to an Entity Terminator block, in a noncumulative way by resetting the counter at each time instant.




To open the example, see `Example Model for Noncumulative Entity Count`.

1. Add two Entity Generator blocks, an Entity Input Switch block, an Entity Terminator block, and a Simulink Function block from the SimEvents library to a new model. For more information, see Simulink Function.
2. Connect the blocks as shown in the diagram.
3. Double-click the Entity Generator1 block. In the **Entity generation** tab, set the **Period** to 2.

   In the model, 2 entities arrive to Entity Terminator block at time `0`, `2`, `4`, `6`, `8`, `10` and `1` entity arrives at time `1`, `3`, `5`, `7`, `9`.
4. Double-click the function signature on the Simulink Function block and enter `nonCumCount()`.




5. Double-click the Simulink Function block. Add a Digital Clock block from the **Simulink** > **Sources** library. Set the **Sample time** parameter to `-1` for inherited sample time.
6. Add a MATLAB Function block. Double-click it and enter this code.

```
function y = fcn(curtime)
% Define count for counting and prevtime for previous time stamp
persistent count prevtime;
% Check if prevtime is empty and initiate the count
if isempty(prevtime)
    prevtime = curtime;
    count = 0;
end
% Increase count by 1 for equal time stamps.
if isequal(curtime, prevtime)
    count = count + 1;
% Reset count to 1 if two consequitive time stamps are not identical
else
```

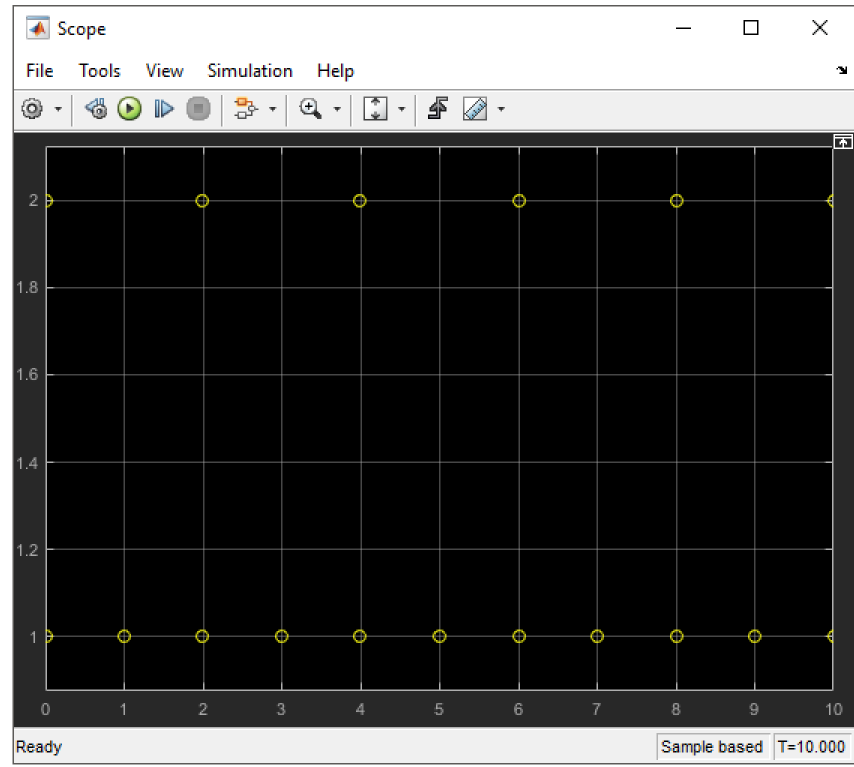
```
    prevtime = curtime;
    count = 1;
end
% Output count for visualization
y = count;
end
```

Save the file (optional).

7. Connect the output of the MATLAB Function block to a Simulink Scope block.
8. In the parent model, double-click the Entity Terminator block. In the **Entry action** field of the **Event actions** tab, enter this code.

```
nonCumCount();
```

9. Simulate the model and open the Scope block in the Simulink Function block.
10. Change the plotting settings of the Scope block by right-clicking the plot and selecting **Style**. Select `no line` for the **Line** and `circle` for the **Marker** parameters.
11. Observe that the block illustrates the noncumulative entity count for the entities arriving the Entity Terminator block. The block also illustrates the instantaneous entity arrivals at each time.




To count the number of events that occur instantaneously, use `nonCumCount()` in any **Event actions**.

## See Also

Entity Gate | Entity Generator | Entity Input Switch | Entity Terminator

## Related Examples


- "Count Simultaneous Departures from a Server" on page 1-27
- "Generate Entities When Events Occur" on page 1-13
- "Specify Intergeneration Times for Entities" on page 1-16
- "Generate Multiple Entities at Time Zero" on page 1-21


## More About


- "Entities in a SimEvents Model"
- "Role of Entity Ports and Paths"

# Working with Entity Attributes and Entity Priorities



| In this section... |
| --- |
| "Attach Attributes to Entities" on page 1-32 |
| "Set Attributes" on page 1-33 |
| "Use Attributes to Route Entities" on page 1-35 |
| "Entity Priorities" on page 1-36 |



You can attach data to an entity using one or more entity attributes. Each attribute has a name and a numeric value. You can read or change the values of attributes during the simulation.

For example, suppose your entities represent a message that you are transmitting across a communication network. You can attach the length of each particular message to the message itself using an attribute named `length`.

You can use attributes to describe any measurable property of an entity. For example, you could use attribute values to specify:

- Service time to be used by a downstream server block
- Switching criterion to be used by a downstream switch block

You can also set entity priorities which is used to prioritize events

## Attach Attributes to Entities

To attach attributes to an entity, use the Entity Generator block. You can attach attributes such as:

- Constant values
- Random numbers
- Elements of either a vector in the MATLAB workspace or a vector that you can type in a block dialog box
- Values of an output argument of a MATLAB function
- Values of a signal
- Outputs of a function defined in Simulink or Stateflow® environment.

These lists summarize the characteristics of attribute values for structured entity types.

Attribute values must be:

- Real or complex
- Arrays of any dimension, where the dimensions remain fixed throughout the simulation
- All built-in data types (`double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, and `uint32`)
- Enumerations

For a given attribute, the characteristics of the value must be consistent throughout the discrete-event system in the model. Attribute values can not be:

#### Not Permitted as Attribute Values

- Structures
- Buses
- Variable-size signals or variable-size arrays
- Frames

## Set Attributes

To build and manage the list of attributes to attach to each departing entity, use the controls under the **Define attributes** section of the Entity Generator block. Each attribute appears as a row in a table.

Using these controls, you can:

- Manually add an attribute.
- Modify an attribute that you previously created.

The buttons under **Set Attribute** perform these actions.

| Button | Action | Notes |
| --- | --- | --- |
|  | Add an attribute to the table. | Rename the attribute and specify its properties. |
|  | Remove the selected attribute from the attribute table. | When you delete an attribute this way, no confirmation appears and you cannot undo the operation. |

You can also organize the attributes by clicking  and .

The table displays the attributes you added manually. Use it to set these attribute properties.

| Property | Specify | Use |
| --- | --- | --- |
| **Attribute Name** | The name of the attribute. Each attribute must have a unique name. | Double-click the existing name, and then type the new name. |
| **Attribute Initial Value** | The value to assign to the attribute. | Double-click the value, and then type the value you want to assign. |

#### Write Functions to Manipulate Attributes

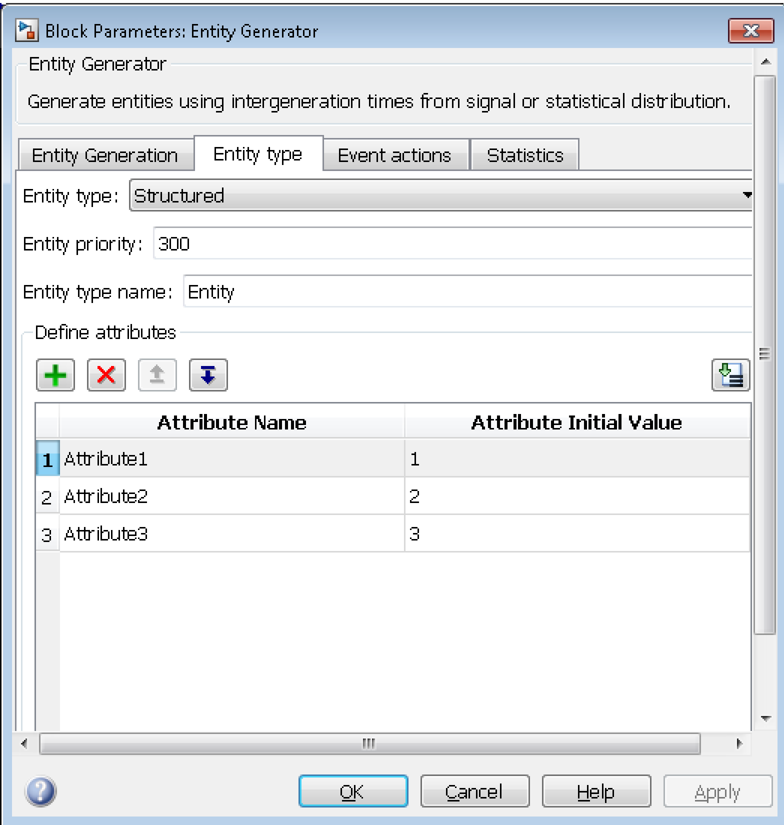
To manipulate attributes using MATLAB code, use the **Event actions** tab of a block. To access the attribute, use the notation *entityName.attributeName*. For example:

```
entity.Attribute1 = 5;
```

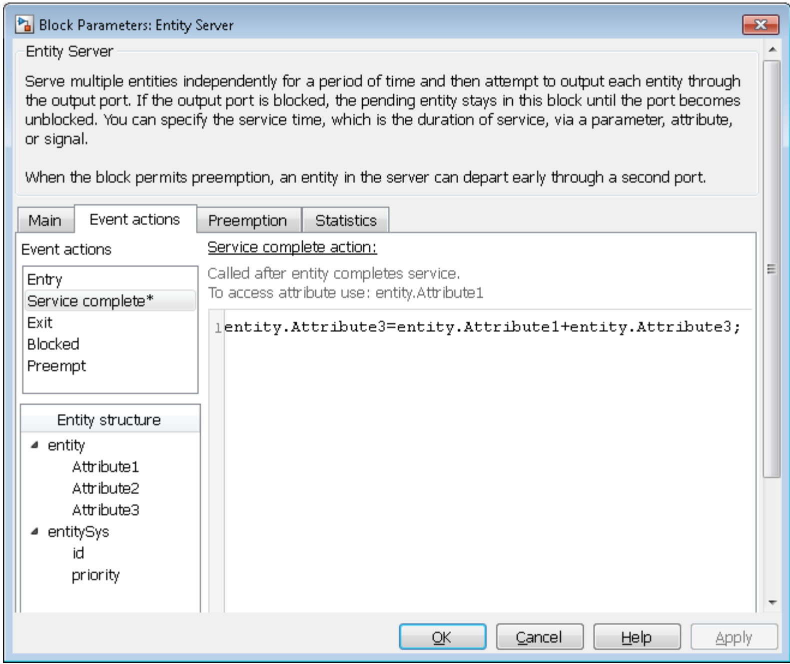
Suppose that you want to modify the attribute of an entity after it has been served.

1. In a new model, from the SimEvents library, drag the Entity Generator, Entity Server, and Entity Terminator blocks and connect them.

2 Double-click Entity Generator block and, in the **Entity type** tab, add three attributes to the attributes table.
3 Double-click on the second and third attributes in the **Attribute Name** column and rename them Attribute2 and Attribute3, respectively.

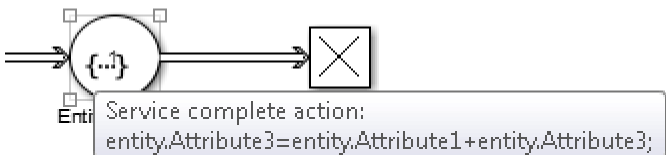



4 In the Entity Server block, click the **Event actions** tab.
5 Click **Service complete**, and enter MATLAB code to manipulate the entity attributes you added in the Entity Generator block:




Click **OK**. The Entity Server block displays the event action language.

6 To see the action, in the model, hover over the Entity Server block event action icon block.




## Use Attributes to Route Entities

Suppose entities represent manufactured items that undergo a quality control process and a packaging process. Items that pass the quality control test proceed to one of three packaging stations, while items that fail the quality control test proceed to one of two rework stations. You can model the decision-making process by using these switches:

- An Entity Output Switch block that routes items based on an attribute that stores the results of the quality control test
- An Entity Output Switch block that routes passing-quality items to the packaging stations
- An Entity Output Switch block that routes failing-quality items to the rework stations

You can use the block **Switching criterion** parameter `From attribute` option to use an attribute to select the output port. For an example, see “Model Traffic Intersections as a Queuing Network” on page 5-12.

## Entity Priorities

SimEvents uses entity priorities to prioritize events. The smaller the priority value, the higher the priority.

You specify entity priorities when you generate entities. In the Entity Generator block, in the **Entity Type** tab, the **Entity priority** specifies the priority value of the generated entity.

You can later change entity priorities using an event action. For example, in the Entity Generator block **Event actions** tab, you can define an event action to change the entity priority during simulation using code such as:

```
entitySys.priority=MATLAB code
```

The entity priorities have a role in prioritization of events in the Event Calendar which schedules events to be executed.

In SimEvents, the Event Calendar sorts events based on their times and associated entity priorities as follows:

1. The event that has the earliest time executes first.
2. If two entities have events occurring at the same time, the event with the entity of higher priority occurs first.
3. If both entities have the same priority, either event may be served first. To service the entities in a deterministic order, change one of the entity priorities.

For example, assume a forward event is associated with an entity that exits block *A* and enters block *B*. The priority of this event is the priority of the entity being forwarded. If there are two entities trying to depart a block at the same time, the entity with the higher priority departs first.

For more information about Event Calendar and debugging SimEvents models, see “Debug SimEvents Models” on page 12-2.

## See Also

Discrete Event Chart | Entity Generator | MATLAB Discrete Event System

## Related Examples


- “Working with Entity Attributes and Entity Priorities” on page 1-32
- “Serve High-Priority Customers by Sorting Entities Based on Priority” on page 2-29


## More About


- “Entities in a SimEvents Model”
- “Model Resource Allocation Using Composite Entity Creator block” on page 1-44

# Inspect Structures of Entities

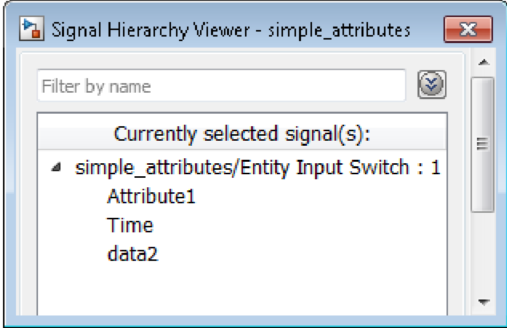
You can inspect entity structures using these methods:

- On a signal line, using the Signal Hierarchy Viewer (for more information, see "Display Entity Types" on page 1-37).
- In a block at run-time, using the Storage Inspector.

## Display Entity Types

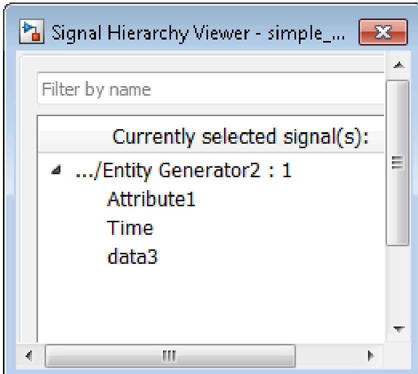
To show entity types in your model, in the model editor, right-click a line and select **Signal Hierarchy**. The Signal Hierarchy Viewer interactively displays about entities, signals, and bus objects. For more information on the Signal Hierarchy Viewer, see "Display Bus Hierarchy".

If you have configured any blocks to receive an entity structure that the preceding block does not provide, upon compilation, the software automatically displays entity types. This behavior helps you to troubleshoot the mismatch in entity structures before simulation. The software displays an approximate list of the entity types and attributes. Use this as a guideline and not as a definitive list.




If entities on two separate paths have the same structure throughout the model, you can use the same entity type for both entity paths.

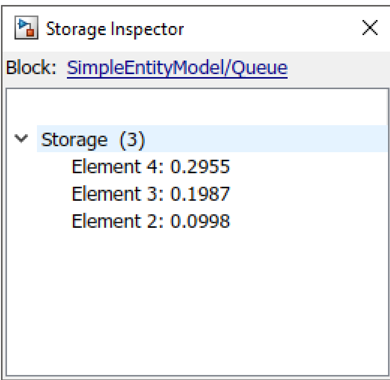
If you now modify the second Entity Generate block path to change `data2` to `data3`, the structure of entities on the second path becomes unique. You must specify a new entity type name for the second Entity Generator block.



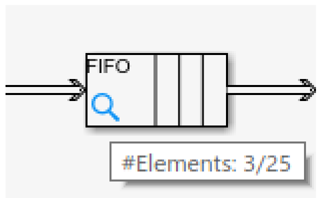
## Inspect Entities at Run Time

To inspect entities at run-time, use the Storage Inspector. Inspect entities, batched entities, and their attribute values in a block.

1. In a SimEvents model, use the Simulink Simulation Stepper to step through the model.
2. As you step through the model, each block with entities updates to contain a magnifying glass.
3. To display entity details, including attributes, click the magnifying glass.




4. To see the number of entities, hover over the magnifying glass.




Alternatively, use the SimEvents Debugger to inspect entities. For more information, see SimEvents Debugger.

## See Also

Entity Generator | SimEvents Debugger

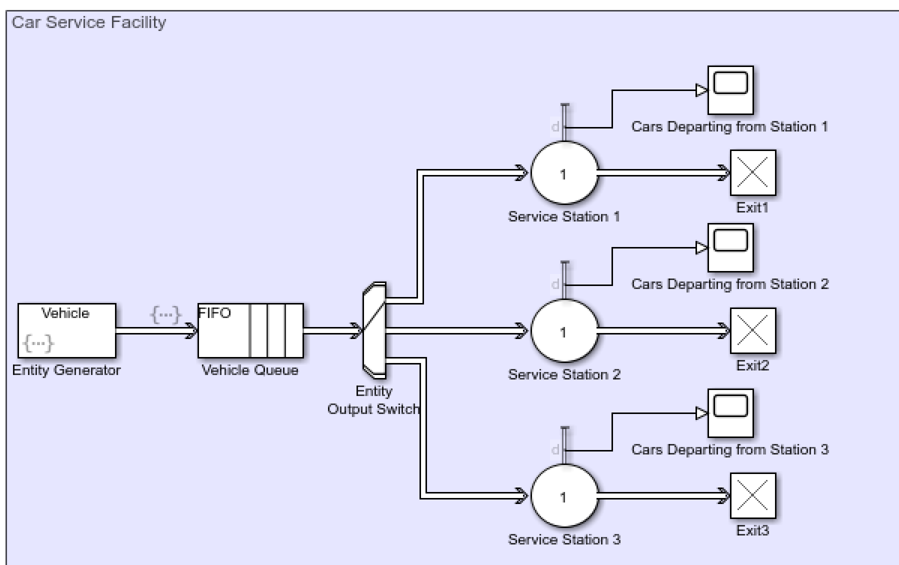
## More About

- "Working with Entity Attributes and Entity Priorities" on page 1-32
- "Entities in a SimEvents Model"
- "Role of Entity Ports and Paths"

# Generate Entities Carrying Nested Data Structures

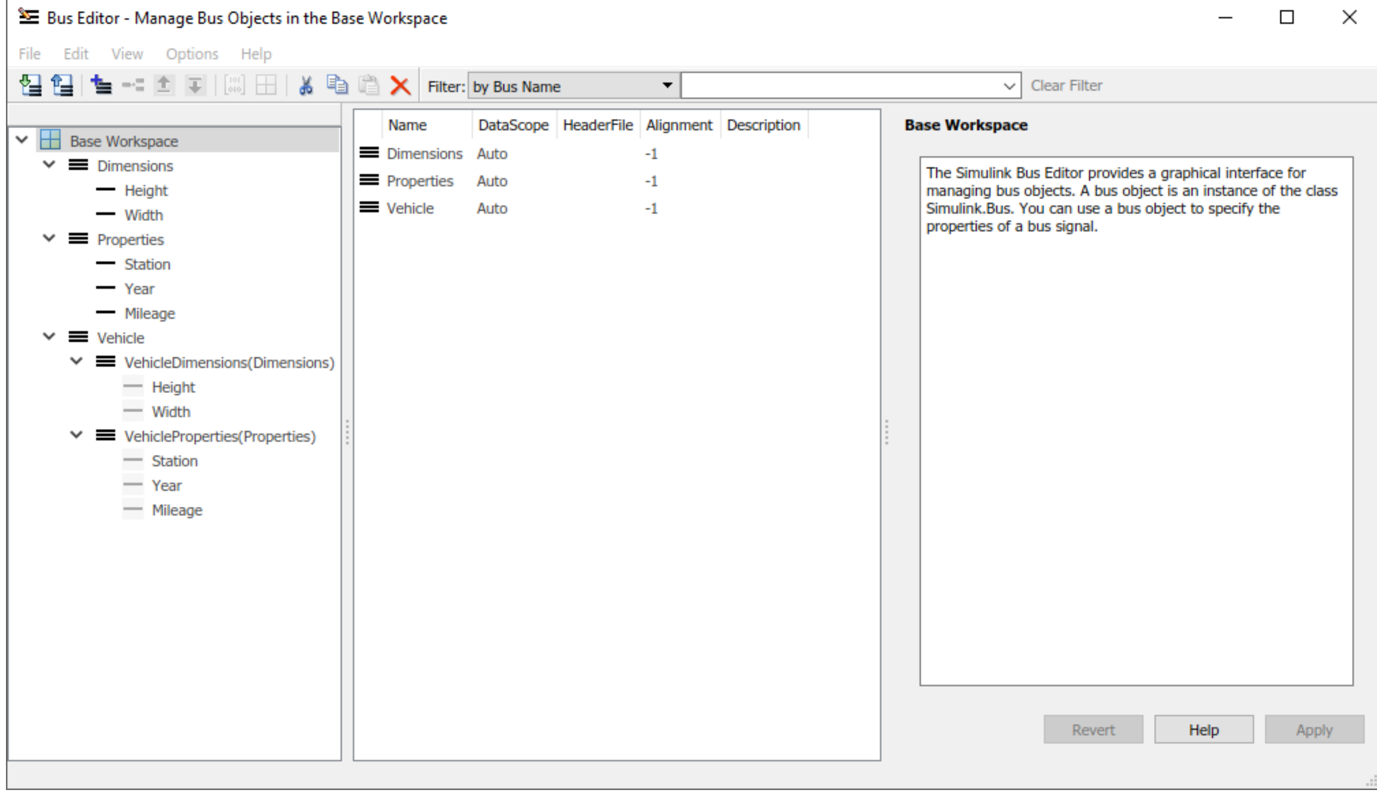
This example shows how to investigate the throughput of a vehicle service facility using Simulink Bus Editor to create nested data structures carried by entities.

The facility has three service stations represented by three Entity Server blocks. The vehicles arriving at the facility are queued and then directed to one of the three service stations based on their size and mileage. It is assumed that the older vehicles require more service time.




1. Create entities that represent vehicles arriving at a service facility. The entities carry data representing the vehicle dimensions and properties as nested bus objects. Vehicle dimensions include vehicle height and width in meters and vehicle properties include its age and current mileage. For more information about using the Bus Editor, see “Create and Specify Simulink.Bus Objects”.

   a. Under **Modeling** tab, in **Design** section, select and open the **Bus Editor**.
   b. In the Bus Editor, select **File** > **Add Bus**.
   c. Create a new bus object and set the **Name** property to Dimensions.
   d. Select **File** > **Add/Insert BusElement** to create two bus elements, Height and Width.
   e. Create another bus object and set the **Name** property to Properties. Add three bus elements Station, Year, and Mileage.
   f. Create another bus object and set the **Name** property to Vehicle.
   g. Add two bus elements and set their **Name** properties to VehicleDimensions and VehicleProperties. For their **Data type** properties, use the Bus: <object name> template, replacing <object name> with Dimensions and Properties.




2 Add an Entity Generator block. Double-click the Entity Generator block.

   a Select the **Entity type** tab. Set the **Entity type** to `Bus object` and **Entity type name** as `Vehicle`.

      `Vehicle` is the bus object created by the Bus Editor.

   b Select the **Event actions** tab. In the **Generate action** field, enter:

```
% Vehicle Dimensions
entity.VehicleDimensions.Height = 1+rand();
entity.VehicleDimensions.Width = 1+rand();
% Vehicle Properties
entity.VehicleProperties.Year = randi([1996 2018]);
entity.VehicleProperties.Mileage = randi([50000 150000]);
```

      The vehicles arrive at the facility with random dimensions and properties.

3 Add an Entity Queue block and rename it Vehicle Queue.

   a In the '**Main** tab, set the **Capacity** to `Inf`.

   b Select the **Event actions** tab. In the **Entry action** field, enter this code to specify service station selection for vehicles.

```
% If the height and width of the vehicle are greater than 1.5 m, select Station 1.
if entity.VehicleDimensions.Width > 1.5 && entity.VehicleDimensions.Height > 1.5
    entity.VehicleProperties.Station = 1;
% Else, if the vehicle's mileage is greater than 90000 km, select Station 2.
else if entity.VehicleProperties.Mileage > 90000
     entity.VehicleProperties.Station = 2;
% If the vehicle's mileage is less than 90000 km, select Station 3.
else
     entity.VehicleProperties.Station = 3;
```

```
end
end
```

The vehicles are queued to be directed to the correct service station and vehicle dimensions and properties are used to select the appropriate service station.

4. Add an Entity Output Switch block.

   a. Set the **Number of output ports** to 3.
   b. Set the **Switching criterion** to `From attribute`.
   c. Set the **Switch attribute name** to `VehicleProperties.Station`.

   The Entity Output Switch block directs the vehicles to the stations based on the specified `Station` attribute.

5. Add an Entity Server block that represents the service station. Rename the block Service Station 1.

   a. In the **Main** tab, set the **Service time source** to `MATLAB action`.
   b. In the **Service time action** field, enter:

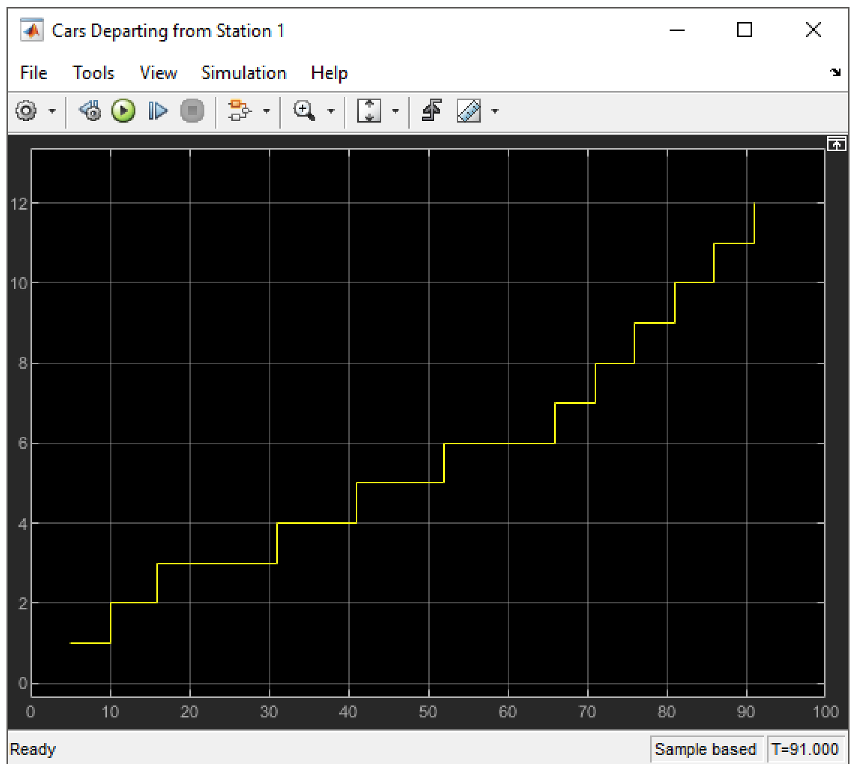
   ```
   if entity.VehicleProperties.Year > 2015
       dt = 1;
   else
       dt = 5;
   end
   ```

   It is assumed that the vehicle service time is longer for older vehicles.

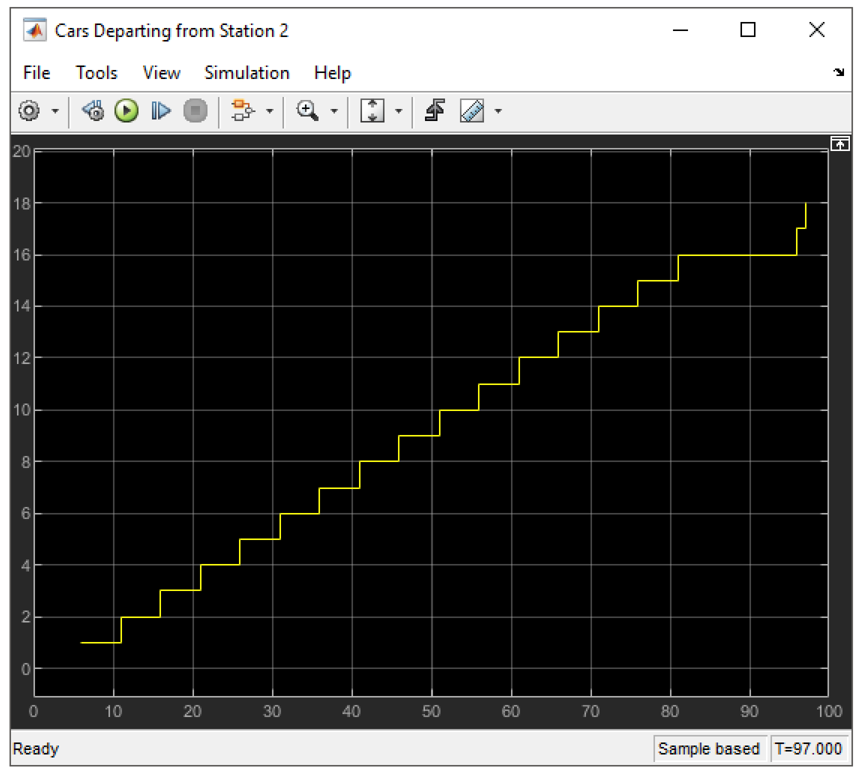
   c. In the **Statistics** tab, select **Number of entities departed, d** statistic and connect it to a scope.

6. Connect Service Station 1 to an Entity Terminator block.
7. Follow the same steps to create Service Station 2 and Service Station 3 and connect them as shown.
8. Increase the simulation time to `100` and run the simulation.

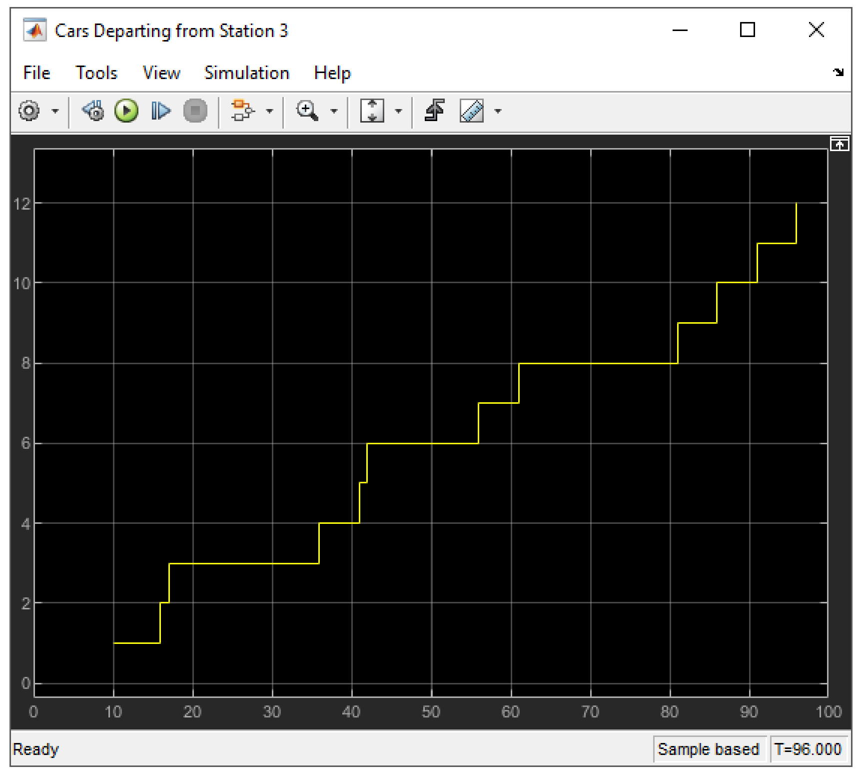
   Observe the number of vehicles served at Service Station 1.




Observe the number of vehicles served at Service Station 2.




Observe the number of vehicles served at Service Station 3.




## See Also

Entity Generator | Entity Output Switch | Entity Queue | Entity Server

## More About

- “Entities in a SimEvents Model”
- “Adjust Entity Generation Times Through Feedback” on page 1-24
- “Generate Entities When Events Occur” on page 1-13
- “Inspect Structures of Entities” on page 1-37

# Model Resource Allocation Using Composite Entity Creator block

The goal of this example is to show how to use Composite Entity Creator block for resource allocation. You can combine entities from different paths using the Composite Entity Creator block. The entities that you combine, called composite entities, might represent different parts within a larger item, such as the header, payload, and trailer that are parts of a data packet. Alternatively, you can model resource allocation by combining an entity that represents a resource with an entity that represents a part or other item.

The Composite Entity Creator block detects when all necessary component entities are present and when the composite entity that results from the combining operation will be able to advance to the next block. The Composite Entity Creator block provides options for managing information (attributes and timers) associated with the component entities. You can also configure the Composite Entity Creator block to make the combining operation reversible via the Composite Entity Splitter block.

## See Also

Composite Entity Creator | Composite Entity Splitter | Entity Generator

## More About

- "Entities in a SimEvents Model"

# Replicate Entities on Multiple Paths

The Entity Replicator block enables you to distribute copies of an entity on multiple entity paths. Replicating entities might be a requirement of the situation you are modeling. For example, copies of messages in a multicasting communication system can advance to multiple transmitters or multiple recipients.

Similarly, copies of computer jobs can advance to multiple computers in a cluster so that the jobs can be processed in parallel on different platforms.

In some cases, replicating entities is a convenient modeling construct.

## Modeling Notes

- Unlike the Entity Output Switch block, the Entity Replicator block has departures at all of its entity output ports that are not blocked, not just a single selected entity output port.
- If your model routes the replicates such that they use a common entity path, then be aware that blockages can occur during the replication process. For example, if you have this scenario:

  - An Entity Replicator block has the **Replicas depart from** parameter set to `Separate output ports`.
  - The block has these output ports connected to individual Entity Server blocks.

  A blockage can occur because the servers can accommodate at most one of the replicates at a time. The blockage causes fewer than the maximum number of replicates to depart from the block.
- Each time the Entity Replicator block replicates an entity, the copies depart in a sequence whose start is determined by the **Hold original entity until all replicas depart** parameter. Although all copies depart at the same time instant, the sequence might be significant in some modeling situations. For details, see the reference page for the Entity Replicator block.

## See Also

Entity Generator | Entity Replicator

## More About

- “Entities in a SimEvents Model”

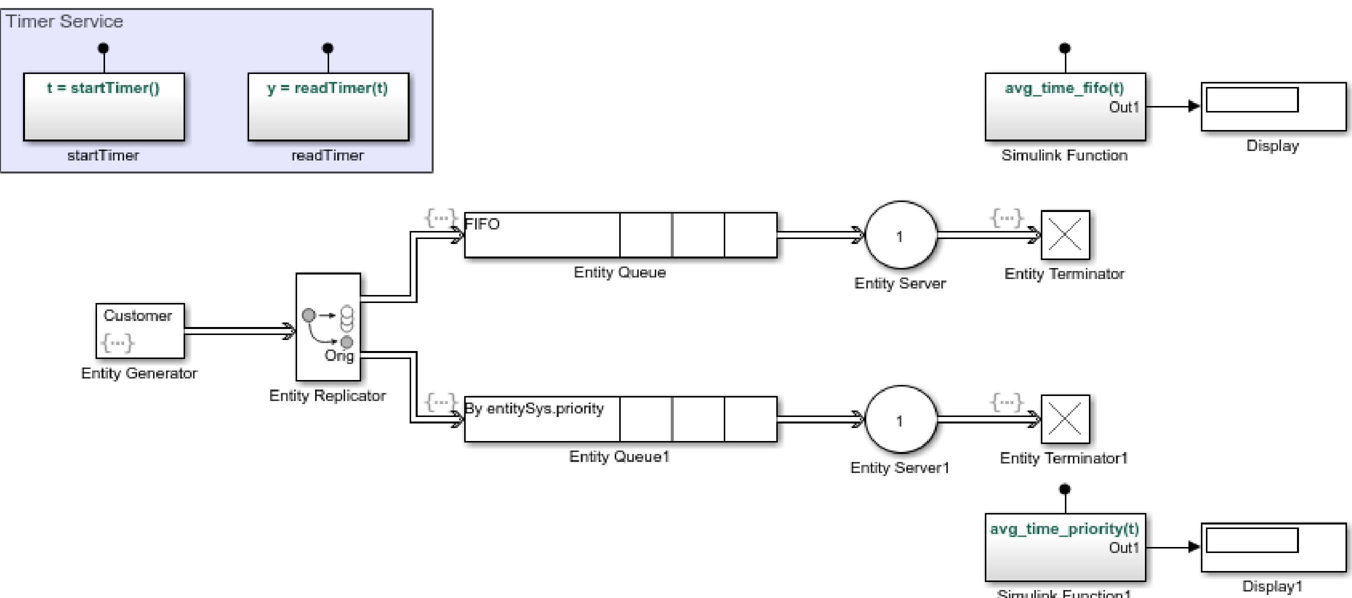
# Measure Point-to-Point Delays

Determine how long each entity takes to advance from one block to another, or how much time each entity spends in a particular region of your model. To compute these durations, you can measure time durations on each entity that reaches a particular spot in the model. A general workflow is:

1. Create an attribute on the entity that can hold the time value.
2. When the entity reaches a particular point in the model, set the current value of time on the attribute. Call a Simulink function that contains a Digital Clock block.
3. When the entity reaches the destination, compute the time interval by passing the attribute value to another Simulink function that compares it to the current simulation time.

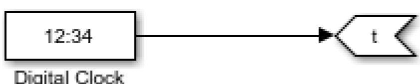
## Basic Example Using Timer Blocks

This example lets you see if a FIFO order or prioritized queue for customers results in a shorter wait time. The `startTimer` and `readTimer` Simulink functions jointly perform the timing computation. This example uses the Mean block from the DSP System Toolbox™ to calculate average times.

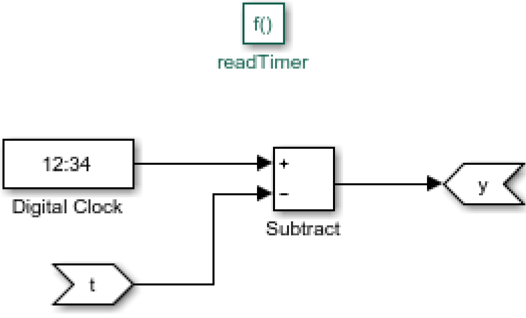



This example has four Simulink Function blocks. Two define timer functions, startTimer and readTimer. The other functions calculate average times.

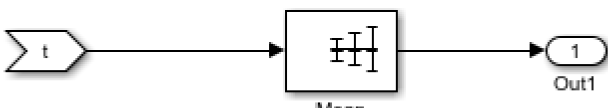
1. In a new model, drag the blocks shown in the example and relabel and connect them as shown.
2. For the startTimer block, define:




3 For the readTimer block, define:




4 For the `avg_time_fifo(t)` and `avg_time_priority`Simulink Function blocks, insert a Mean block, for example:




5 For the Entity Generator block:

   a In the **Entity type** tab, add two attributes, `ServiceTime` and `Timer`.

   b In the **Entity actions** tab, set the two attribute values:

```
entity.ServiceTime = exprnd(3);
entitySys.priority = randi(2);
```

6 In Entity Queue:

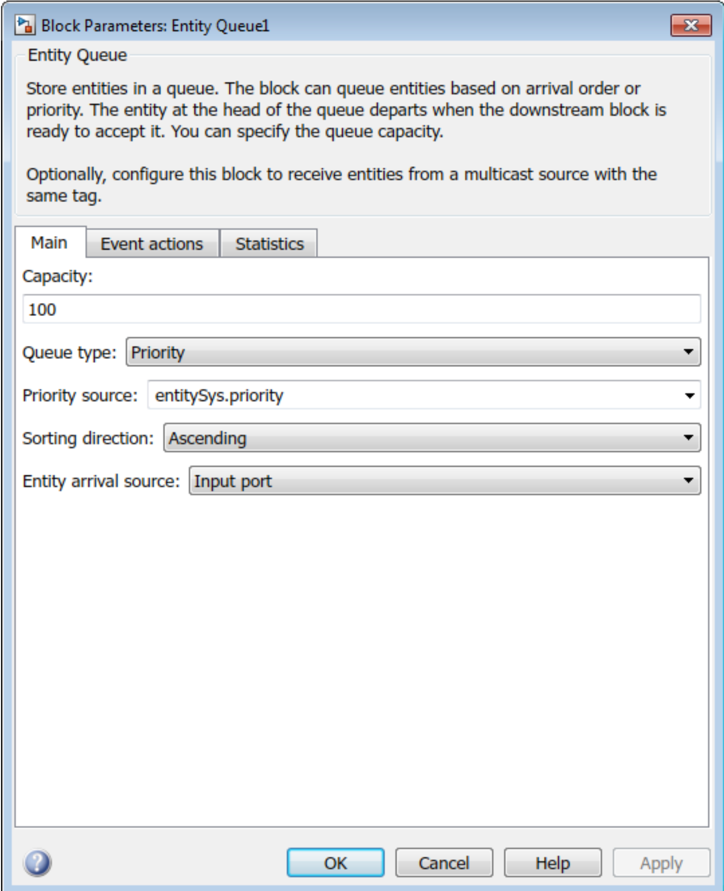
   a In the **Main** tab, set **Queue type** to FIFO.

   b In the **Event actions** tab, call the `startTimer` function for the Entry action:

```
entity.Timer = startTimer();
```

7 In Entity Queue1:

   a In the **Main** tab, configure the block to be a priority queue with a priority source of `entitySys.priority`:




**b** In the **Event actions** tab, call the `startTimer` function for the Entry action:

```
entity.Timer = startTimer();
```

**8** For both Entity Server blocks:

**a** Set **Service time source** to `Attribute`.

**b** Set **Service time attribute name** to `ServiceTime`.

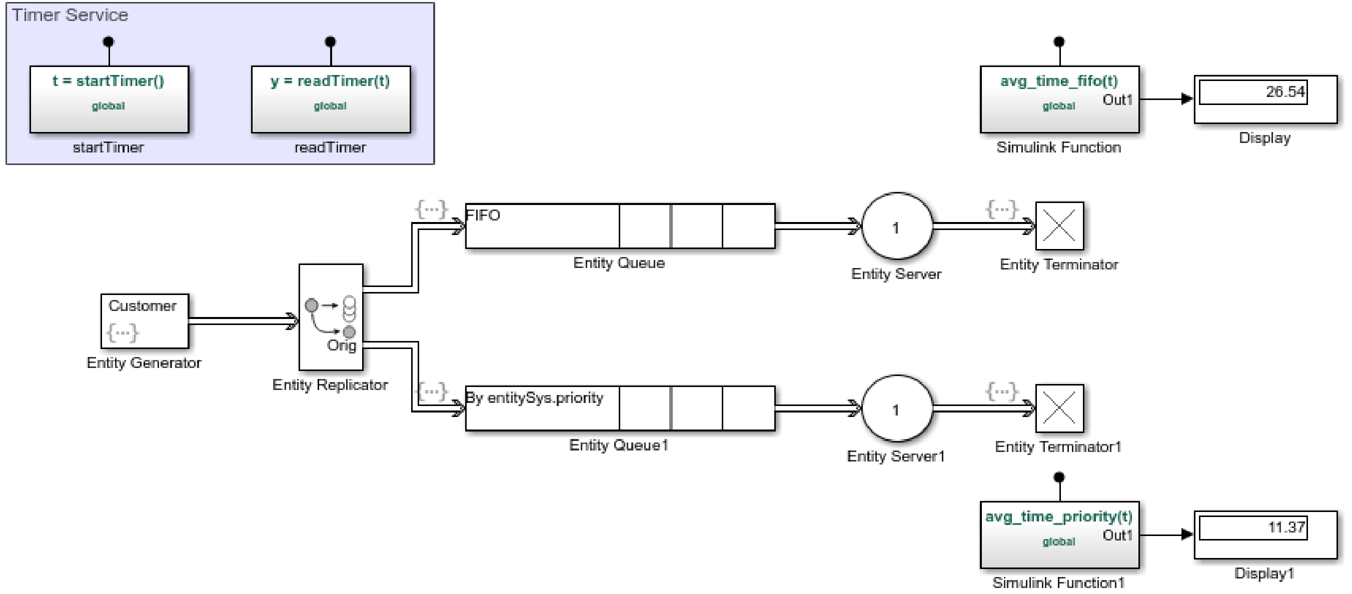
**9** For Entity Terminator, call the `readTimer` and `avg_time_fifo` functions for the Entry event:

```
% Read timer
elapsedTime = readTimer(entity.Timer);

% Compute average
avg_time_fifo(elapsedTime);
```

**10** For Entity Terminator1, call the `readTimer` and `avg_time_priority` functions for Entry event:

```
% Read timer
elapsedTime = readTimer(entity.Timer);

% Compute average
avg_time_priority(elapsedTime);
```

**11** Save and run the model.




## See Also

Entity Generator | Entity Replicator | Simulink Function

## More About

- "Entities in a SimEvents Model"

# Modeling Queues and Servers


- "Model Basic Queuing Systems" on page 2-2
- "Broadcast Entities using Entity Multicasting" on page 2-16
- "Use Queue Event Actions to Model a Storage Tank" on page 2-20
- "Task Preemption in a Multitasking Processor" on page 2-24
- "Model Server Failure" on page 2-27
- "Serve High-Priority Customers by Sorting Entities Based on Priority" on page 2-29

# Model Basic Queuing Systems

This example shows how to model basic queueing systems in a discrete-event simulation using the Entity Queue and Entity Server blocks.

The Entity Queue block stores entities for a length of time that cannot be determined in advance. An everyday example of a queue is people waiting in line for a store register. A shopper cannot determine in advance how long they must wait to complete their purchase. You can use Entity Queue in different applications such as Airplanes waiting to access a runway or messages waiting to be transmitted. The Entity Queue block has storage capacity, entity sorting policy, and entity overwriting policy. Based on these parameters, the block attempts to output entities depending on whether the downstream block accepts new entities.

The Entity Server block stores entities for a length of time, called the service time, then attempts to send the entity depending on whether the downstream block accepts new entities. During the service period, the block is serving the entity that it stores. An everyday example of a server is a person (such as a bank teller or a retail cashier) with whom you perform a transaction with a projected duration.

This example presents basic queuing models that show how to:

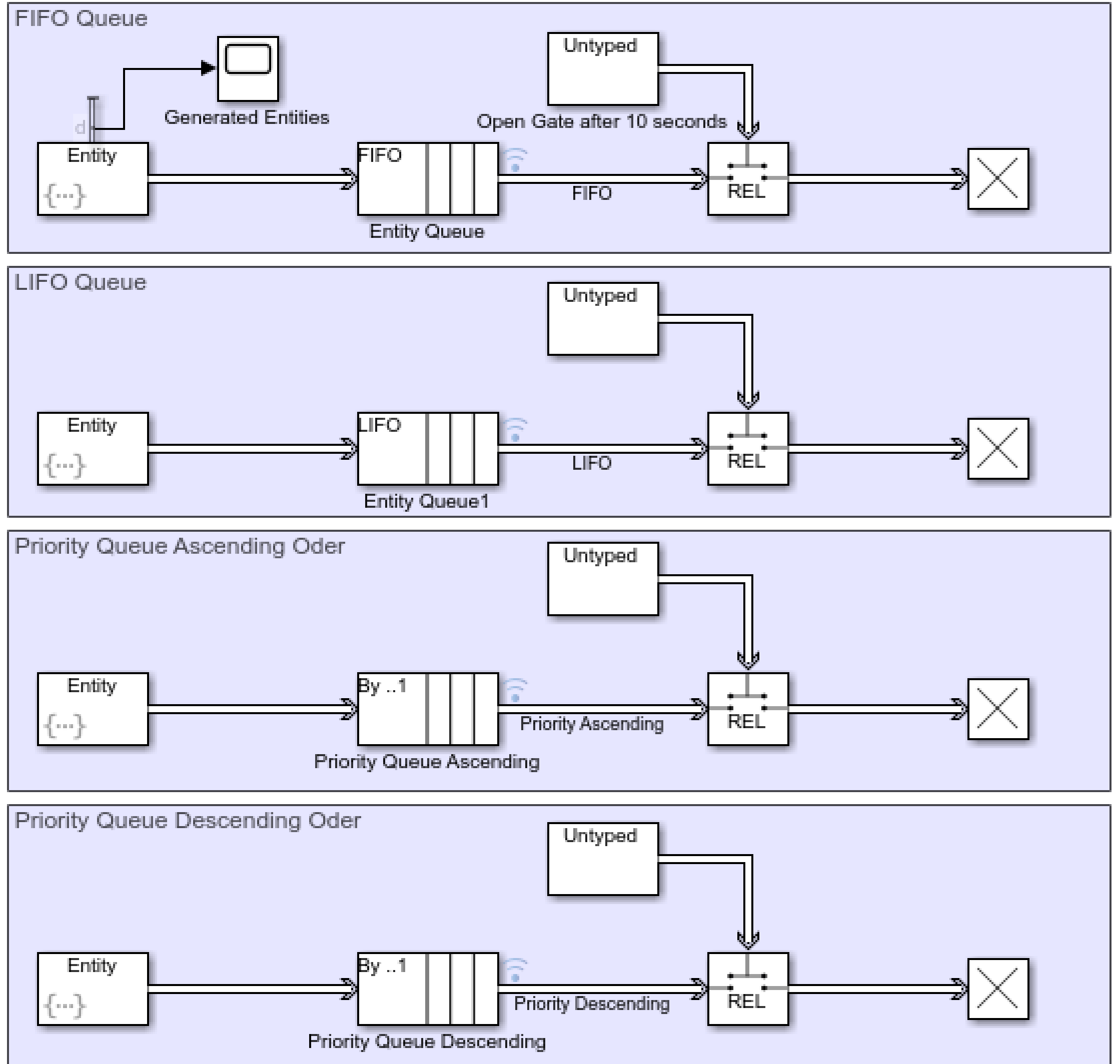
- Model FIFO queue, LIFO queue, and Priority Queue.
- Specify entity overwriting policies when the queue reaches capacity.
- Customize and vary entity service times.
- Assign and change entity attributes based on events.
- Understand queue length statistics during simulation.

## Sort Entities Using the Entity Queue Block

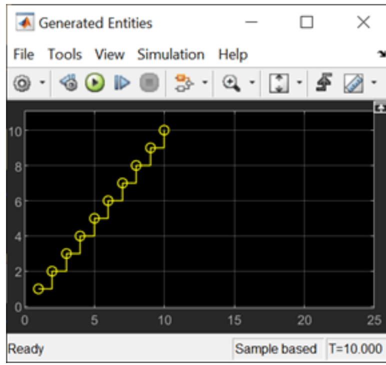
This model shows how to sort entities by changing the queue sorting policy. The Entity Queue block supports three message sorting policies:

- Last-in-first-out (LIFO) — The newest entity in the storage departs first.
- First-in-first-out (FIFO) — The oldest entity in the storage departs first.
- Priority — Entities are sorted based on their priority. You can only use the priority queue when the **Overwrite the oldest element if queue is full** check box is cleared.

The model below shows four different entity sorting behaviors: FIFO, LIFO, Priority in ascending order, and Priority in descending order.




Four identical Entity Generator blocks generate 10 entities each. Each block uses a repeating sequence pattern for entity intergeneration time `dt`.



After generating 10 entities, the intergeneration time `dt` is set to infinity to stop generating entities. In the Entity Generator block, in the **Intergeneration time action** field, this code is used.

```
persistent SEQ;
persistent idx;
if isempty(SEQ)
    % Generate 10 entites with 1 second intervals.
    SEQ = [1 1 1 1 1 1 1 1 1 1];
    idx = 1;
end
if idx > numel(SEQ)
    % Stop entity generation after generating 10 entities.
    dt = inf;
else
    dt = SEQ(idx);
end
```

The block generates an entity and it specifies the attribute `Attribute1` on each entity. You can use attributes to represent features or properties of entities. In this example, the first entity carries a value of `1`, and each generated entity's attribute value increases by `1`. To achieve this behavior, in the Entity Generator block, in the **Event actions** tab, in the **Generate action** field, this code is used.

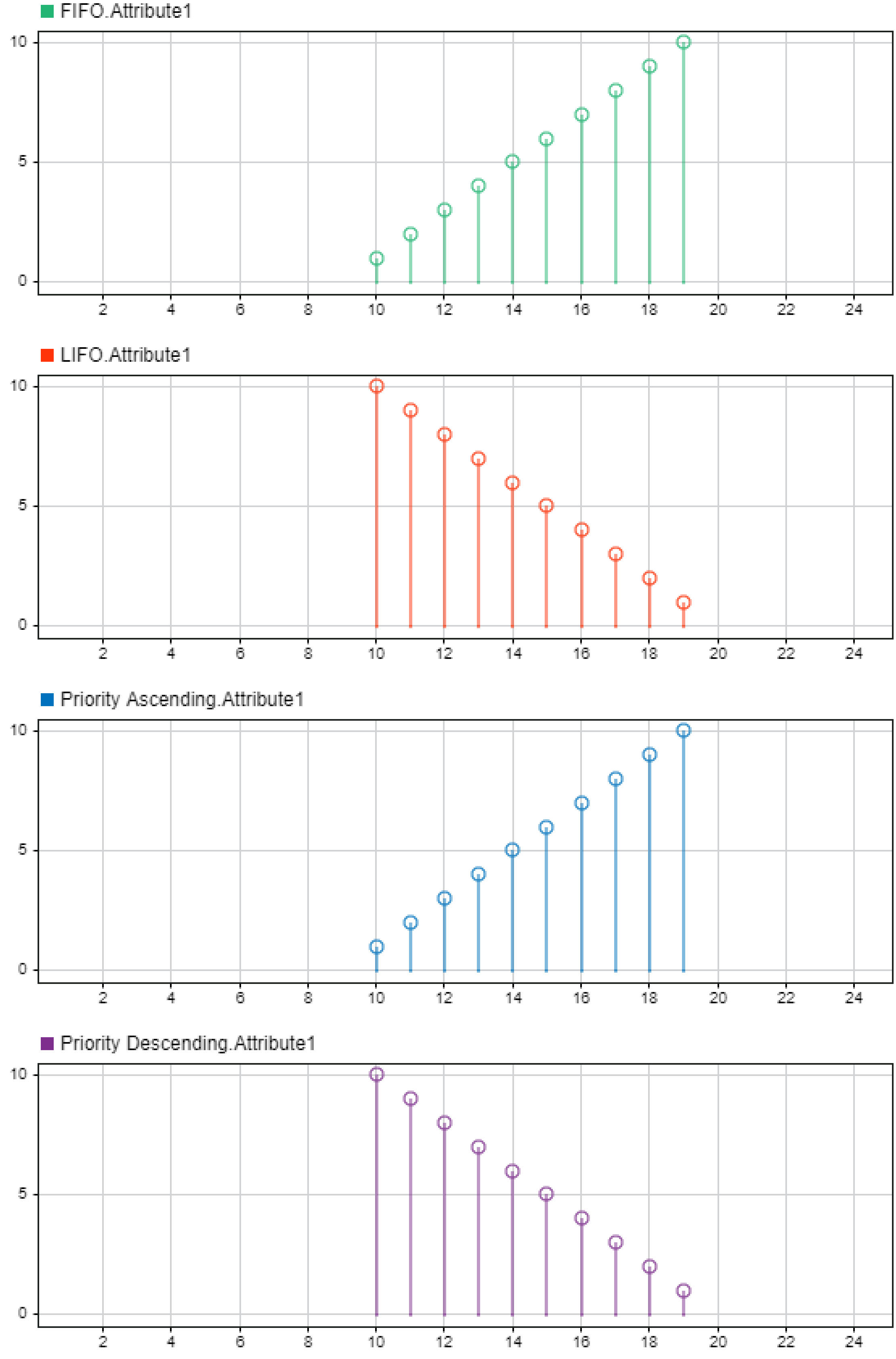
```
% Pattern: Repeating Sequence
persistent SEQ1;
persistent idx1;
if isempty(SEQ1)
    SEQ1 = [1 2 3 4 5 6 7 8 9 10];
    idx1 = 1;
end
if idx1 <= numel(SEQ1)
    entity.Attribute1 = SEQ1(idx1);
end
idx1 = idx1 + 1;
```

The generated entities are forwarded to the four Entity Queue blocks. In order to show the sorting behavior, the Entity Queue blocks are connected to four identical Entity Gate blocks configured as release gates. A release gate allows one entity to pass when it receives an entity carrying a positive (greater than `0`) value from its control port. The gates block entities for the first `10` seconds and store them in the queue. After the first 10 seconds, the gates allow one entity to pass per second based on the sorting policy.

Simulate the model. Open the Simulation Data Inspector and observe that the entities departing from each Entity Queue block are sorted based on the queue sorting policy.

- The first plot shows entities departing from the queue with a FIFO policy. The first entity with `Attribute` value `1`, departs from the queue when the gate opens at time `11`. Subsequent entities depart the queue in the same order of their generation, with increasing attribute value.
- The second plot shows entities departing from the queue with a LIFO policy. This policy reverses the entity departure sequence starting with the entity carrying the largest attribute value.
- The third plot shows entities departing from a priority queue that sorts entities based on their attributes in ascending order instead of their order of entry to the queue. The entity carrying the smallest attribute value departs first. Subsequent entities follow the same policy.

- The fourth plot shows the entities departing from a priority queue that sorts entities based on their attributes in descending order. The entity with the largest attribute value departs first and the rest of the entities follow the same policy.





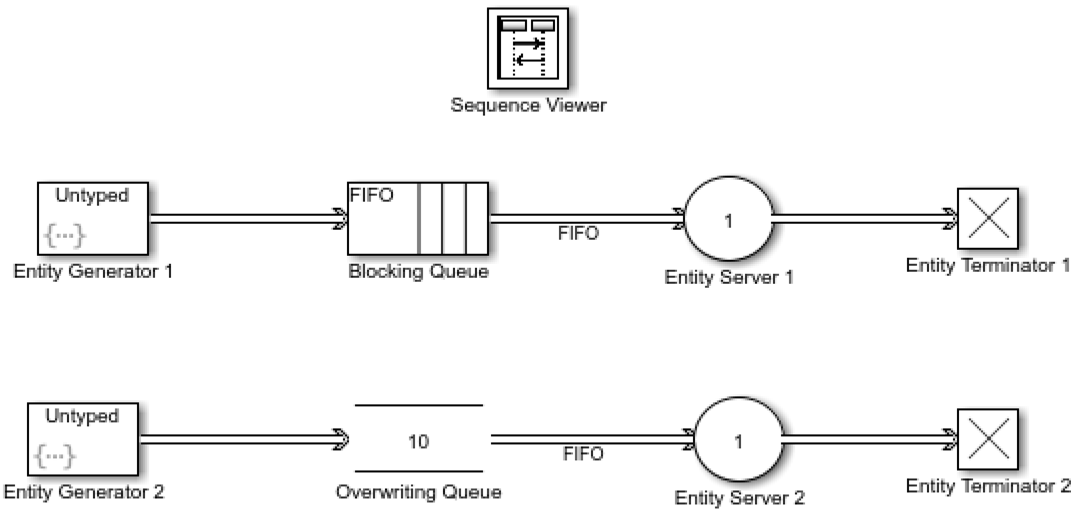

## Queue Entity Overwriting Policies

You can specify what Entity Queue block does when the block reaches its capacity by setting the

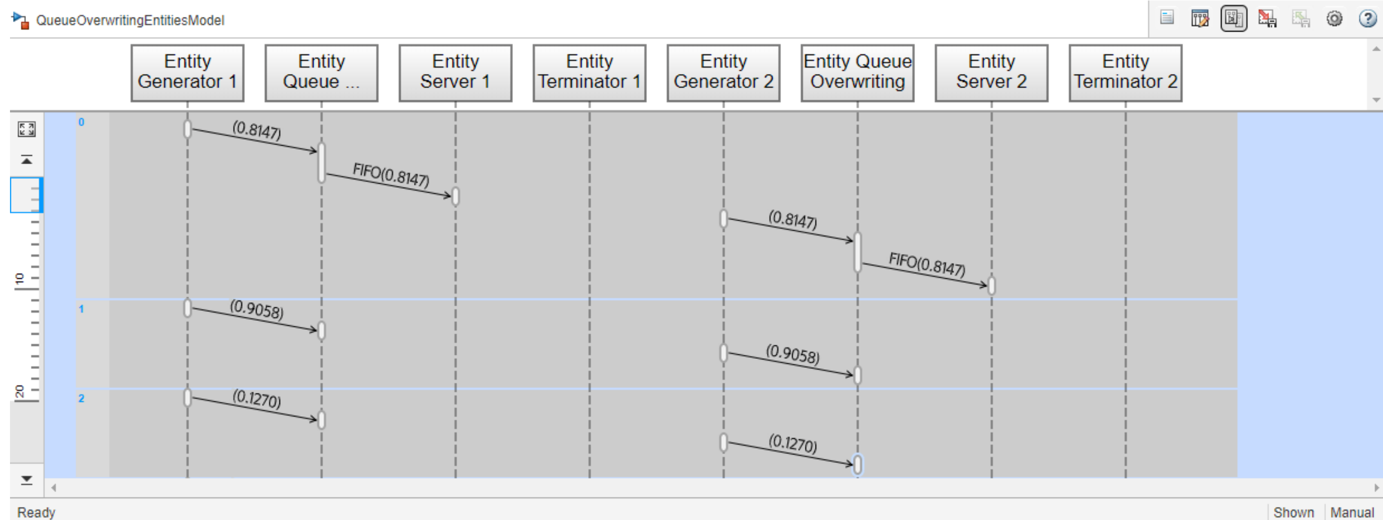
entity overwriting policy. Specify the policy by selecting or clearing the **Overwrite the oldest element if queue is full** check box.

- If the **Overwrite the oldest element if queue is full** check box is cleared, the block does not accept new entities when the queue is full. This is a blocking queue behavior.
- If the **Overwrite the oldest element if queue is full** check box is selected, the block is set to always accept an incoming entity by overwriting the oldest entity in the storage. The block overwrites the oldest entity, but the entity departing the block is determined by the queue sorting policy.

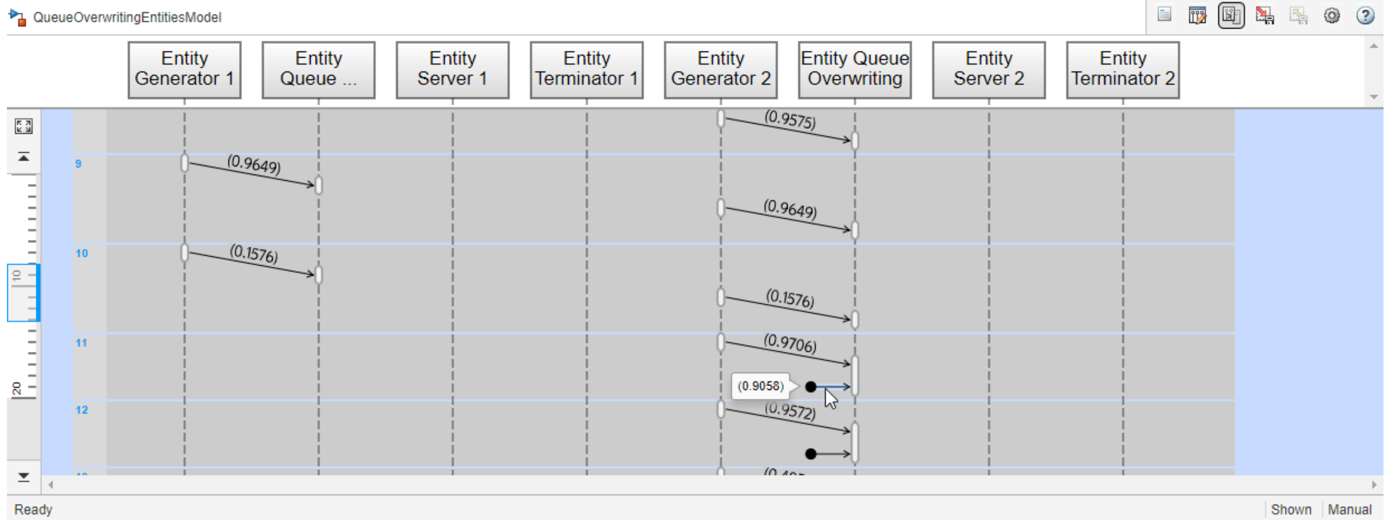
In this model, two identical Entity Generator blocks generate entities every `1` second. The entities are forwarded to two Entity Queue blocks each with a capacity of `10` and a FIFO entity sorting policy. However, the Blocking Queue is configured to not accept new entities when its queue is full, while the Overwriting Queue is configured to overwrite the oldest entity when its queue is full. Blocking Queue and Overwriting Queue are connected to two identical Entity Server blocks, each with a service time value of `25` seconds. The entity generation rate of the Entity Generator block is much higher than the service rate of Entity Server block. This difference causes entities to accumulate in the Entity Queue block.




Simulate the model and open the Sequence Viewer block. Observe that the Entity Generator 1 and Entity Generator 2 blocks initially generate entities with data values of `0.8147`, and the entities are forwarded to Entity Server 1 and Entity Server 2. Both Entity Generator blocks generate a second set of entities with data values of `0.9058`, which are stored in the Blocking Queue and Overwriting Queue because both Entity Server blocks are full. The rest of the generated entities are also stored in the Entity Queue blocks.




Observe that Entity Queue 1 block stops accepting new entities to its storage the time `11`. However, Entity Queue 2 allows the new entity with attribute `0.9706` to storage and overwrites the oldest existing entity, which has a data value of `0.9058`.



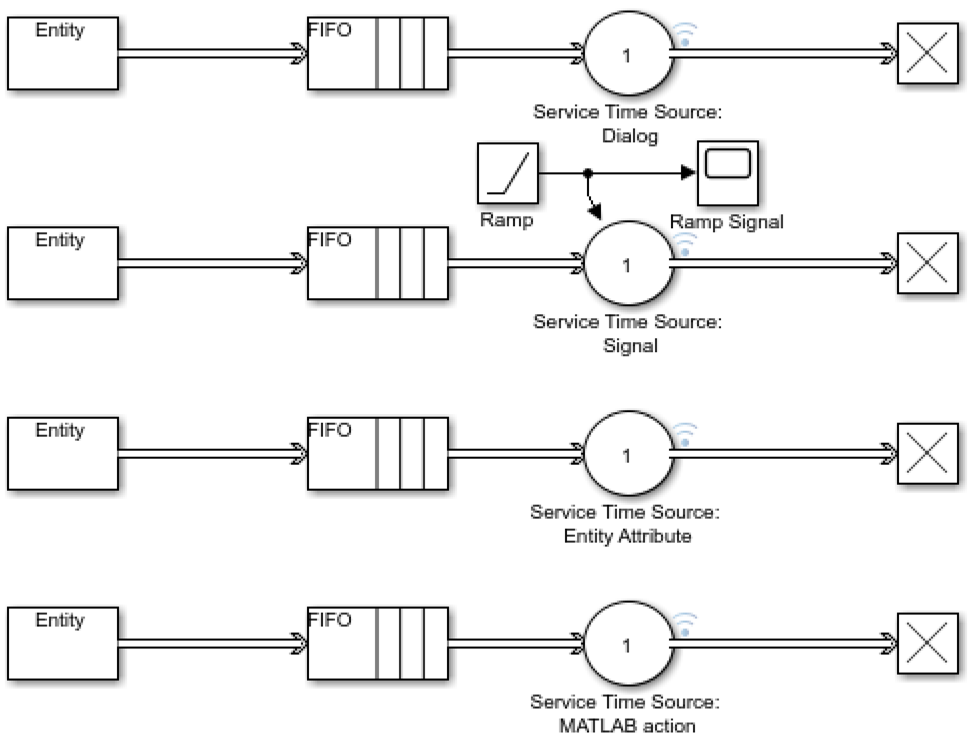

## Customize Entity Service Time

In a basic queuing system, you can use an Entity Server block to model delays based on the processes in your system. You can determine the source that specifies the delay by changing the **Service time source** parameter of the Entity Server block.

This example shows four different sources you can use based on your application:

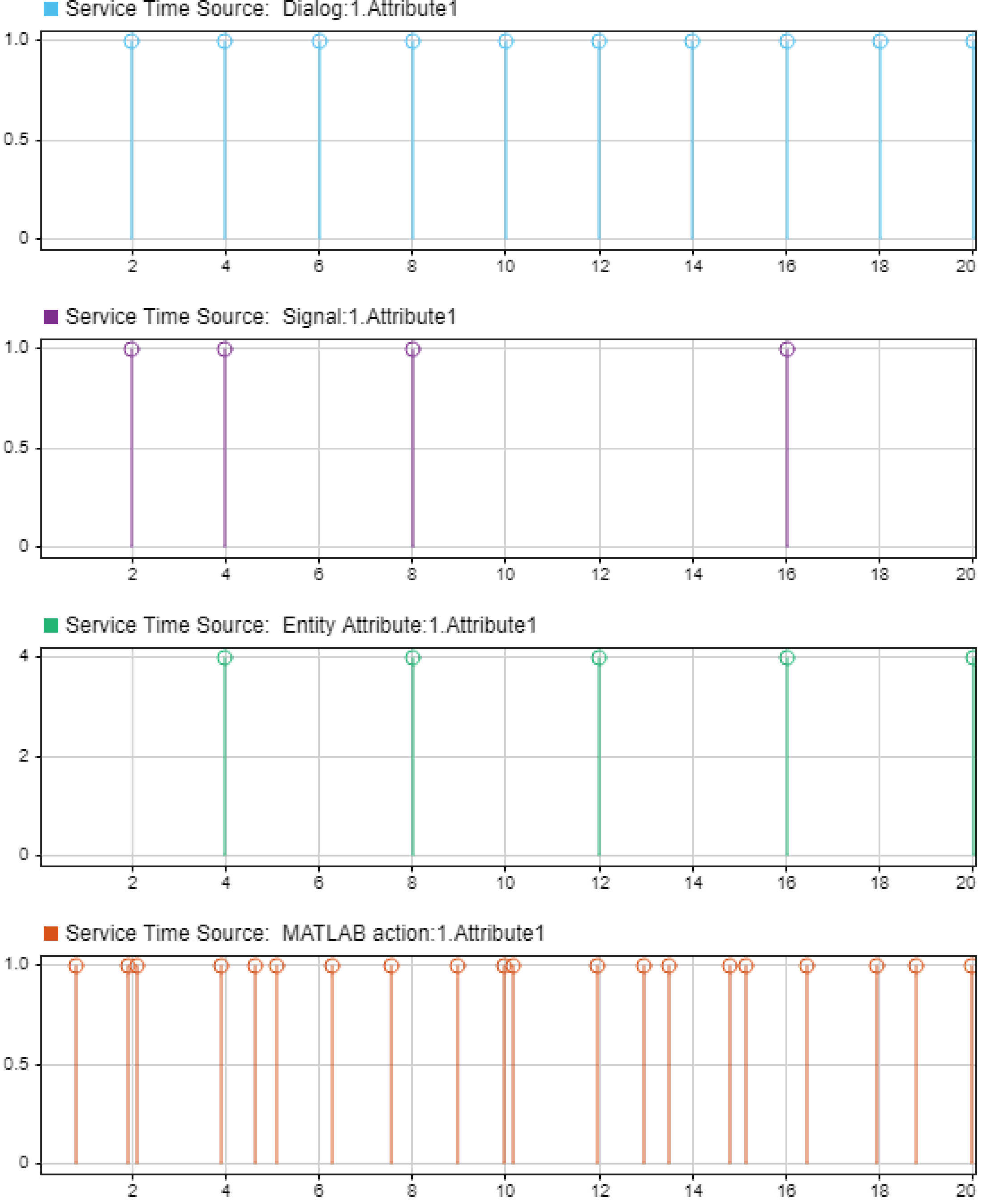
- `Dialog` — You can define a constant service time value. In the first modeling pattern, entities are delayed for 2 seconds. The block then attempts to forward entities to the next block.
- `Signal port` — An incoming Simulink® signal determines the service time. In the second modeling pattern, the block uses ramp signal values as the service time source.

- `Attribute` — A specified entity attribute value determines the service time. In the third modeling pattern, each entity carries `Attribute1` with value `4` which is the service time source.
- `MATLAB action` — You can enter MATLAB™ code in the **Service time action** field and assign the variable to `dt`, which is the parameter the model uses as service time. In the fourth modeling pattern, the random service time `dt = rand(1,1);` is used, and the code sets a random service time value that is uniformly distributed between `0` and `1`.




### Simulate the Model and Review Results

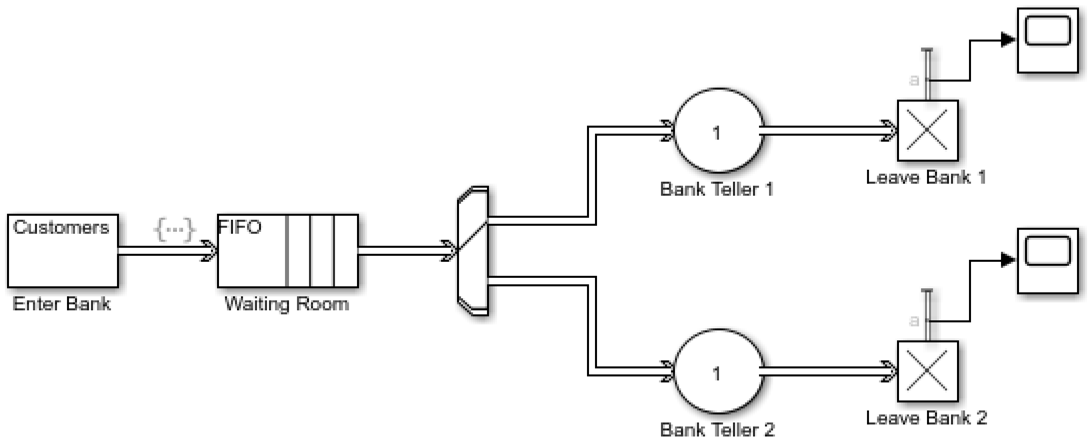
Simulate the model and observe the Simulation Data Inspector, which displays entities forwarded by the Entity Server block.




## Build a Simple Queuing System to Change Entity Attributes

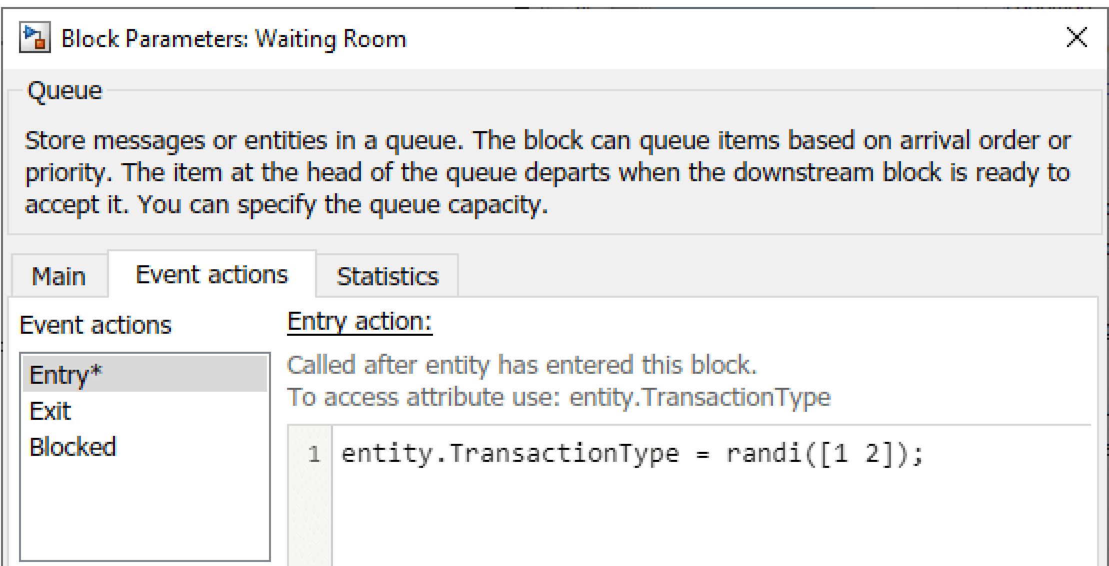
You can attach attributes to entities to represent their features. In a queueing system, you can use actions as responses to events and change entity attributes. For instance, you can change the value of an entity attribute when the entity enters and exits the Entity Queue block. In the Entity Queue block, in the **Event actions** tab, you can see the set of events for which you can create actions.

Suppose you want to model a customer service system in a bank branch. The branch has two bank tellers, each assigned a particular transaction type. Customers arrive at the branch. They pick a number for their transaction and they are directed to the correct bank teller. Customers leave the branch after the transaction is complete.




In this example, customer arrivals are modeled by an Entity Generator block. The customers are assumed to arrive with constant interarrival times, and the **Period** is `1`. Each generated entity is attached an attribute `TransactionType` to represent the customer requests. The `TransactionType|is initialized as |0` because the transaction is unknown before the customers enter the branch.

The waiting room is represented by an Entity Queue block. When a customer enters the waiting room, they are given a number for the corresponding bank teller. This action is represented by changing the entity attributes in the event actions of the Entity Queue block. Below is the action invoked by the entity entry event to the Entity Queue block.




## Analyze Queue Length Using Statistics and Logical Queues

You can use queue statistics to analyze and understand the behavior of a queue in your simulation. Specifically, you can measure:

- The number of entities departed from a queue to a downstream block.
- The number of entities at a specific simulation time.
- The average wait of the entities in the queue before departing the block.
- The average queue length or number of entities extracted from the queue by the Entity Find block.

Understanding these statistics can give insight about your model's behavior. For more information about queue statistics, see “Interpret SimEvents Models Using Statistical Analysis” on page 5-2.

This example presents two different methods for visualizing and understanding queue length.

To determine whether a queue is storing any entities, you can output the statistics that correspond to the number of entities stored in a block.

To output statistics follow these simple steps.

1. Enable the **n** output signal from the queue block. In the block dialog box, on the **Statistics** tab, select the **Number of entities in block, n** check box.
2. From the Sinks library in the Simulink library set, insert a Scope block into the model. Connect the **n** output port of the queue block to the input port of the Scope block.

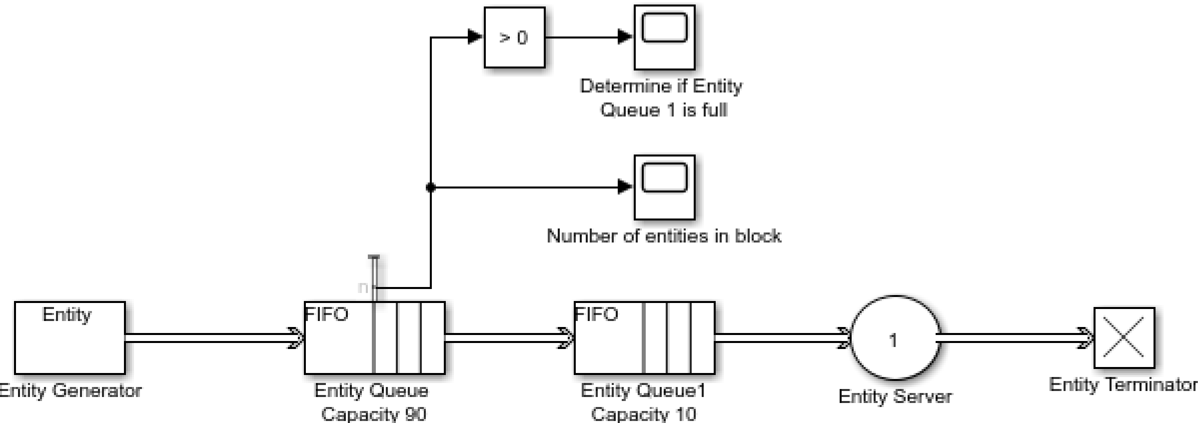
   The scope shows if the queue is empty.

For more information about visualizing queue statistics, see “Explore Statistics and Visualize Simulation Results”.

### Partition a Queue to Understand Queue Length

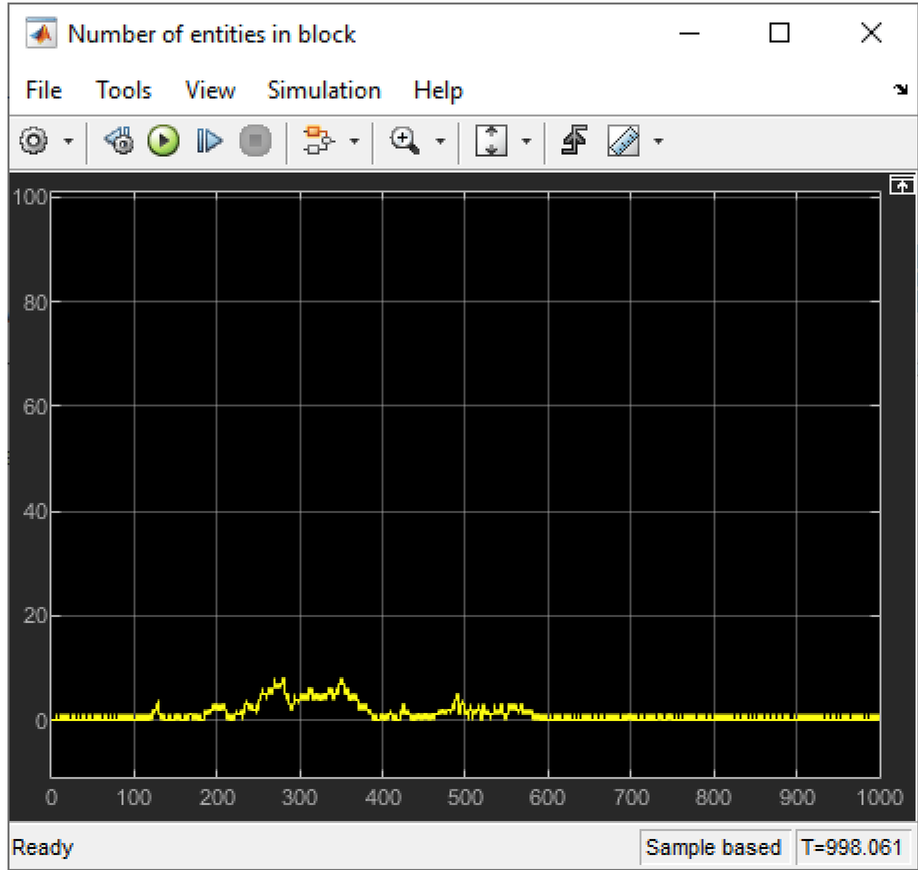
You can partition a queue to understand more details about the queue length and behavior during simulation.

Suppose that you want to determine what proportion of the time the queue length exceeds 10 for a queue with a capacity of 100. You can investigate this by using a pair of queues connected in series. The queues have lengths of 90 and 10. Together they represent a queue with a capacity of 100.

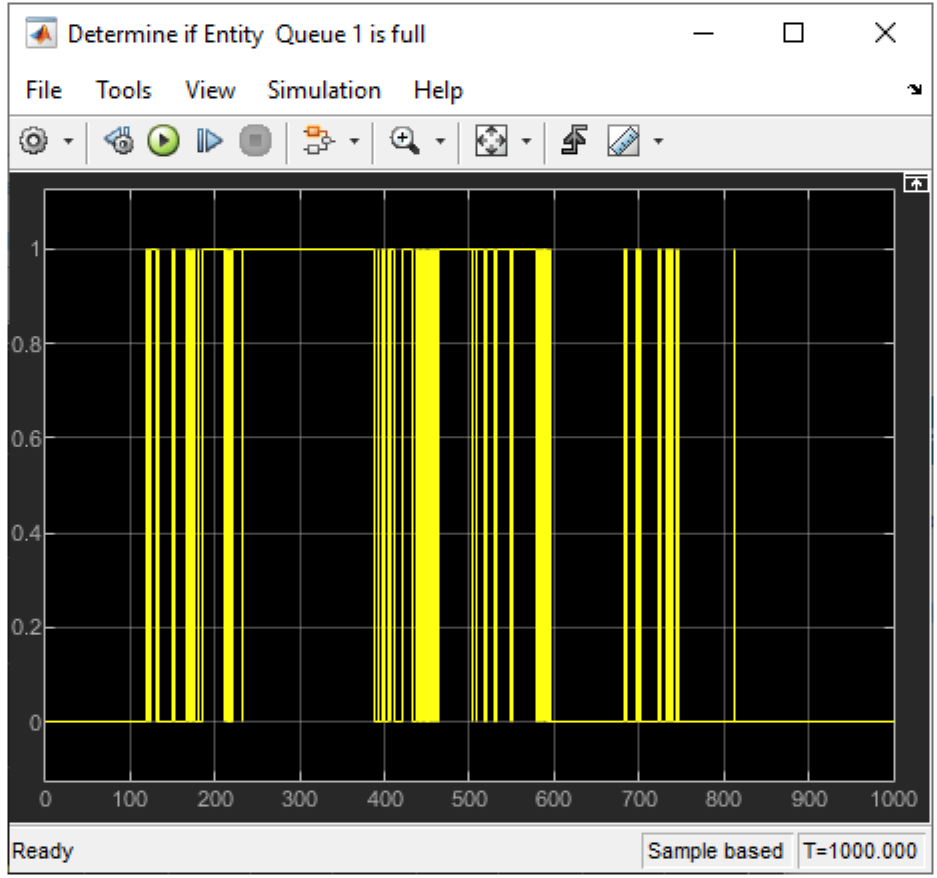



Partitioning the original queue into two smaller queues allows you to gather statistics related to one of the smaller queues. For example, you can view the queue length statistic for the Entity Queue block of with a capacity of 90. If there are entities accumulated in the queue with a capacity of 90, in means that the queue with a capacity of 10 is full. Thus, determining the proportion of the time that the queue with a capacity of 100 has minimum 10 entities is equivalent to determining the proportion of time the queue length of the queue with a capacity of 90 is greater than 0.

Simulate the model. Observe that the Entity Queue Capacity 90 block outputs the **Number of entities in block, n**. Observe that in certain time intervals entities are stored in the block, which indicates that the queue with a capacity of 10 is full.




To visualize the proportion of time that the queue with a capacity of `10` is full, the statistic signal is further processed and compared to zero.




## See Also

Entity Queue | Entity Server

## Related Examples

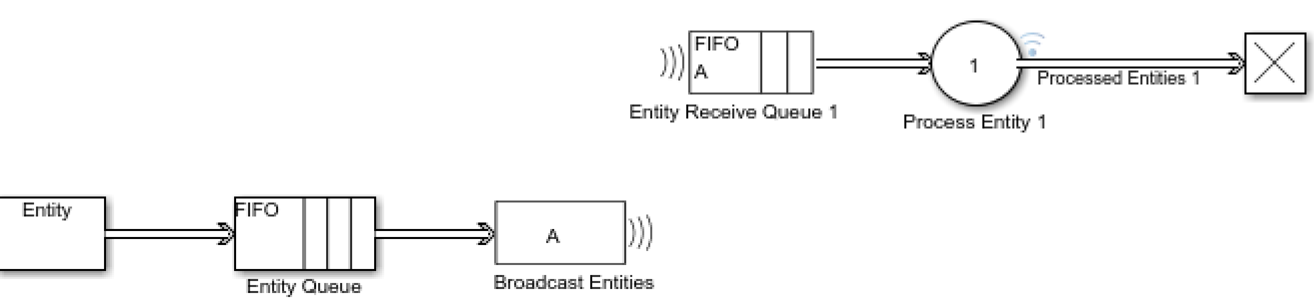
- "Overview of Queues and Servers in Discrete-Event Simulation"
- "Broadcast Entities using Entity Multicasting" on page 2-16
- "Use Queue Event Actions to Model a Storage Tank" on page 2-20
- "Serve High-Priority Customers by Sorting Entities Based on Priority" on page 2-29

# Broadcast Entities using Entity Multicasting

This example shows how to broadcast entities using Entity Multicast and Multicast Receive Queue blocks. Use entity multicasting when you want to copy and broadcast entities to a single receiver or multiple receivers in your model. One common application is creating communication networks in which messages are copied and transmitted between network nodes. For more information see "Model an Ethernet Communication Network with CSMA/CD Protocol" on page 3-22.

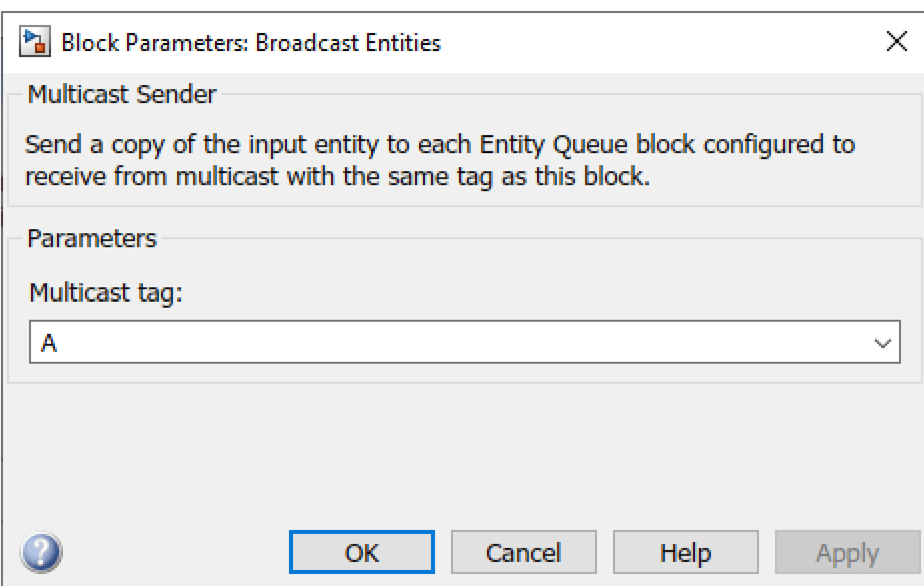
### Build a Simple Model to Broadcast Entities

In this model, an Entity Generator block generates entities. The entities are then queued in the Entity Queue block with FIFO sorting policy. The entities are sent wirelessly to the receiver and further processed by the Entity Server block.

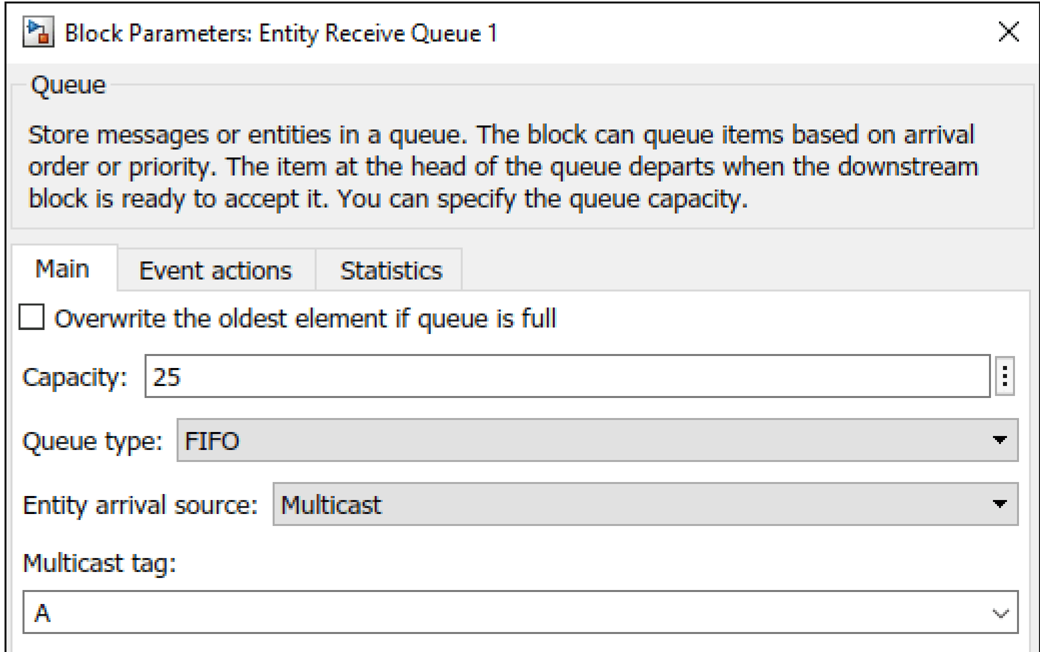



To broadcast entities:

- An Entity Multicast block is connected to the output of the Entity Queue block. The broadcasted entities are tagged such that only Multicast Receive Queues with a matching tag A can receive them.

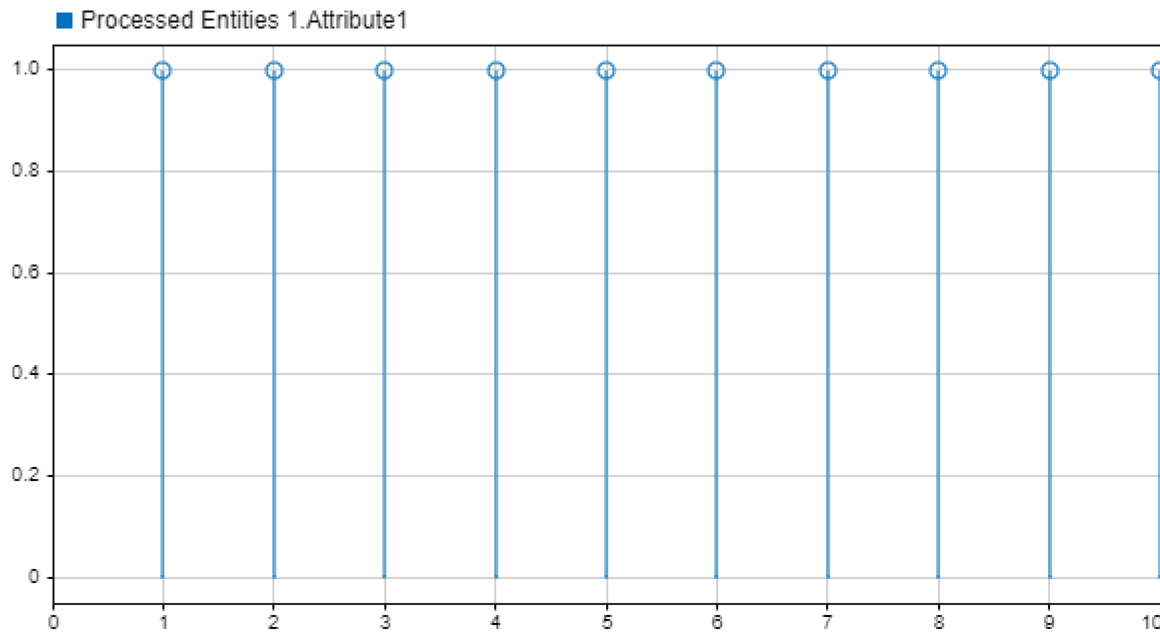


- The Entity Receive Queue block is configured to receive entities with tag A.



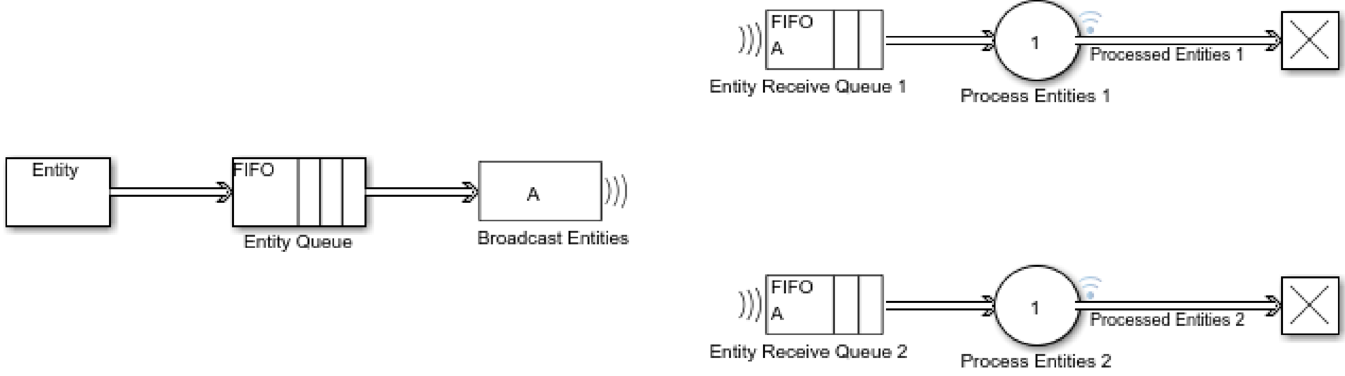

### Simulate the Model with Single Receiver and Review Results

Simulate the model. Open the Data Inspector that displays the received and processed entities that depart the Process Entity 1 block.



#### Use Multicast Tag to Send Entities to Multiple Receivers

You can further modify the model such that the multicast mode enables multiple queues to receive same set of entities from the Entity Multicast block. You can achieve this behavior by creating multiple Multicast Receive Queue blocks whose **Multicast tag** parameter is set to A.

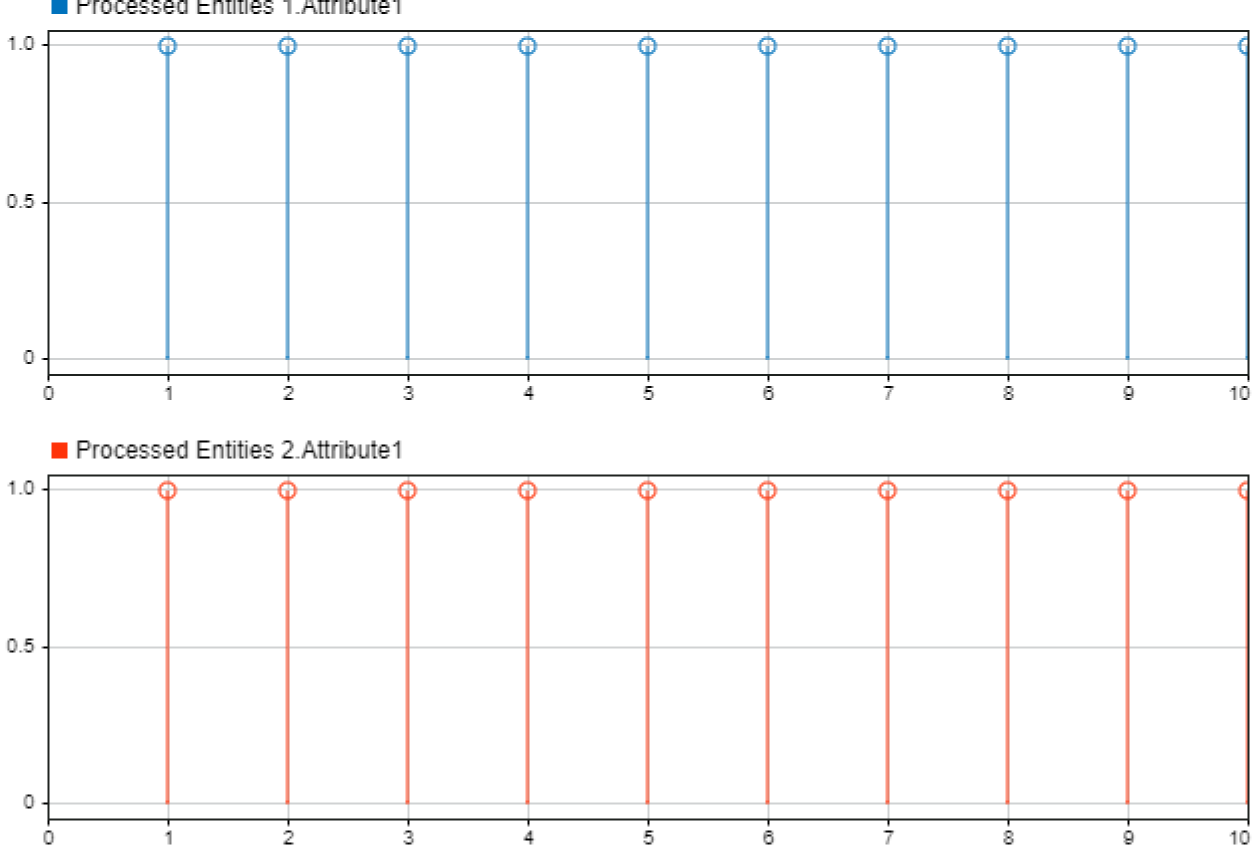



To open the model, use this code:

```
open_system('ParallelEntityQueueServerPairMulticastModel');
```

#### Simulate the Model with Multiple Receivers and Review Results

In this case, Broadcast Entities block copies the entity and sends them to the receivers. Simulate the model to observe its behavior. Open the Data Inspector that displays the same set of entities processed by Processed Entities 1 and Processed Entities 2 blocks.




## See Also

Entity Queue | Entity Server

## Related Examples

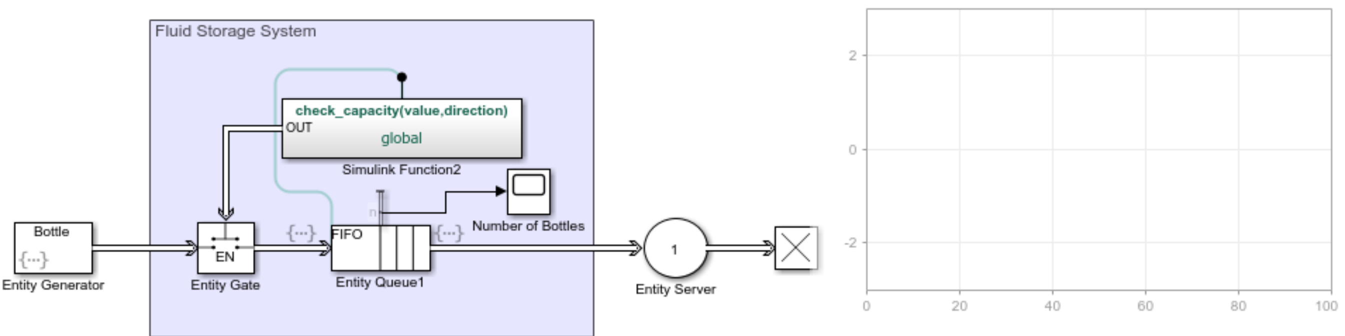
- “Overview of Queues and Servers in Discrete-Event Simulation”
- “Use Queue Event Actions to Model a Storage Tank” on page 2-20
- “Serve High-Priority Customers by Sorting Entities Based on Priority” on page 2-29

# Use Queue Event Actions to Model a Storage Tank

This example shows how to use Entity Queue block event actions, a Simulink Function block, and an Entity Gate block to model a bottle storage system with a limited capacity.

This example uses Entity Generator, Entity Queue, Entity Server, and Entity Terminator blocks to model an assembly line for a bottle storage system with a limited weight capacity. The Entity Generator block represents the arrival of the bottles. Each bottle has a `gallons` attribute, whose value ranges from `0` and `10`, to indicate the amount of liquid it carries. The Entity Queue block represents the storage with limited capacity. The Entity Server block represents the processing of the bottles before they leave the facility.

A fluid storage control system is deployed by using a Simulink Function block and an Entity Gate block to allow bottles to enter the storage system as long as total stored gallon amount is less than `50` gallons. This amount corresponds to the maximum allowed weight the storage can support. It is assumed that the bottle weight is negligible. The Entity Gate block has two states: open and closed. The Simulink Function block monitors the amount of liquid in the storage and controls the state of the Entity Gate block.




Copyright 2020 The MathWorks, Inc.

### Construct the Fluid Storage Control System using Queue Event Actions

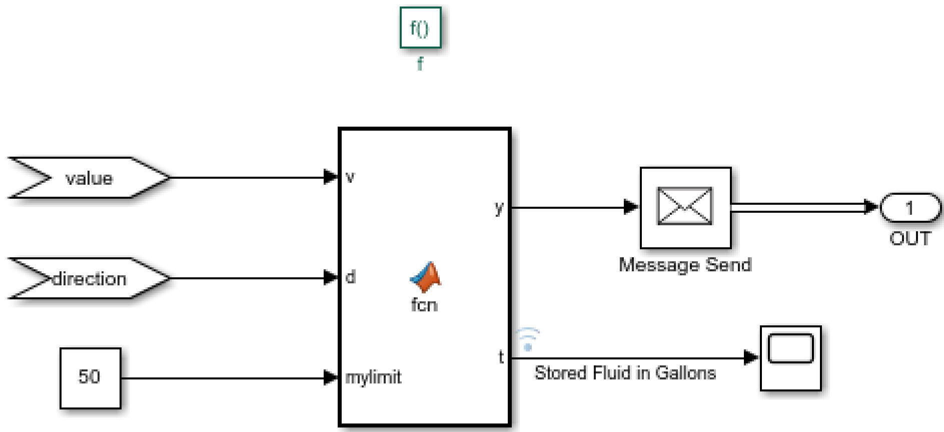
The goal of constructing the Fluid Storage Control System is to allow bottles to enter the queue's storage as long as the total storage gallon amount does not exceed `50` gallons.

To construct the Fluid Storage Control System:

- A Simulink Function block is used with the function signature `check_capacity(value,direction)` to control the Entity Gate block. When the block sends a message with value `1`, the Enable Gate opens to allow containers into the storage. Otherwise, when the block sends a message with value `0`, the gate remains closed.
- In the Entity Queue block, in the **Entry action** and **Exit action** fields, `check_capacity(entity.gallons,1)` and `check_capacity(entity.gallons,-1)` send the container gallon value to the Simulink Function block. The second input of `check_capacity(~,~)` function takes value `1` when an entity enters the queue and `-1` when an entity exits the queue.

- In the Simulink Function block, a MATLAB Function block creates the logic to control the gate. The block accepts three inputs.

1. `v` is the value of the `gallon` attribute for each entity that enters or exits the block.
2. `direction` takes value `1` or `-1` to indicate if an entity enters or exits the block.
3. `myLimit` is the capacity of the liquid container in gallons.




- The MATLAB Function block has two outputs.

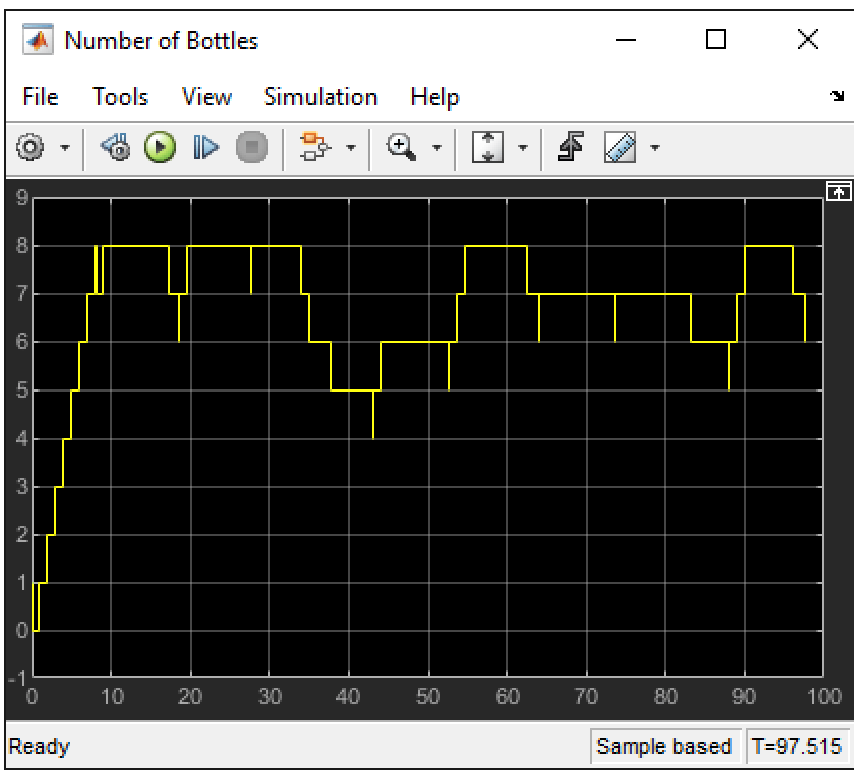
1. `y` takes the value `1` to open the gate, when there is enough storage, and `0` otherwise.
2. `t` is the number of gallons of liquid in the storage.

- The MATLAB Function block contains the following logic to open and close the gate.

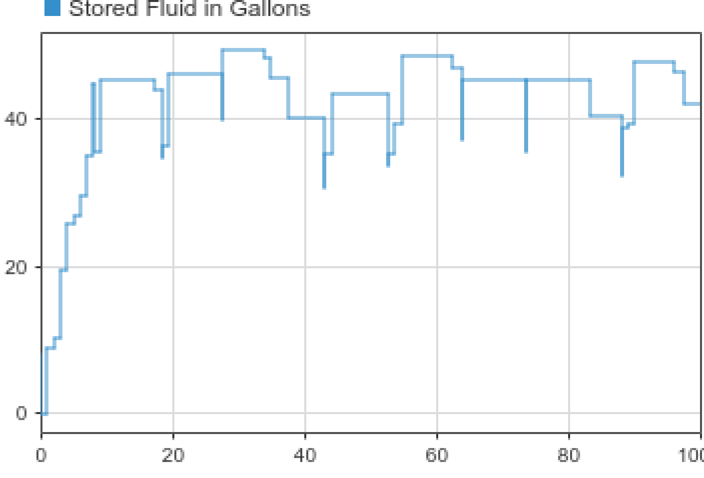
```
function [y,t] = fcn(v,d,myLimit)
% Initialize the persistent variable total that represents the total
% number of gallons of liquid in the storage.
persistent total
% Initialize the variable total.
if isempty(total)
  total = 0;
end
% Add or subtract the gallons carried by the entering or exiting
% entity. d is 1 if the entitiy enters the storage and -1 otherwise.
total = total + d*v;
% Update total number of gallons in the storage.
t = total;
% Compare the total value with myLimit. Containers can have maximum 10
% gallons of liquid.
if total > (myLimit-10)
  % Keep the gate closed.
  y = 0;
else
  % Open the gate to allow containers.
  y = 1;
end
end
```

### Simulate the Model and Review Results

Simulate the model. Observe the number of containers in the liquid storage.




Also, observe the amount of liquid in the storage throughout the simulation, which does not exceed `50` gallons.




## See Also

Entity Queue | Entity Server

## Related Examples

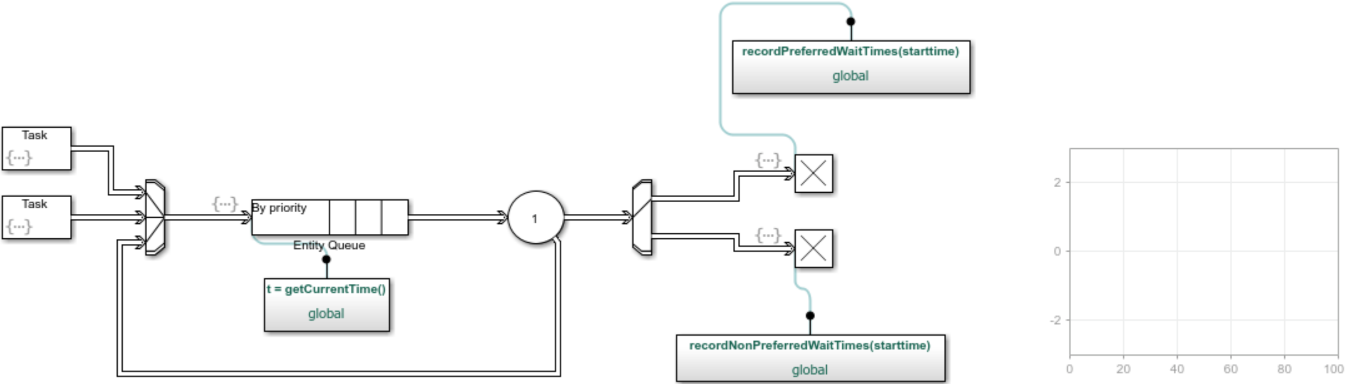
- "Overview of Queues and Servers in Discrete-Event Simulation"
- "Broadcast Entities using Entity Multicasting" on page 2-16
- "Serve High-Priority Customers by Sorting Entities Based on Priority" on page 2-29

# Task Preemption in a Multitasking Processor

This example shows how to force service completion in an Entity Server block using functionality available on the block **Preemption** tab.

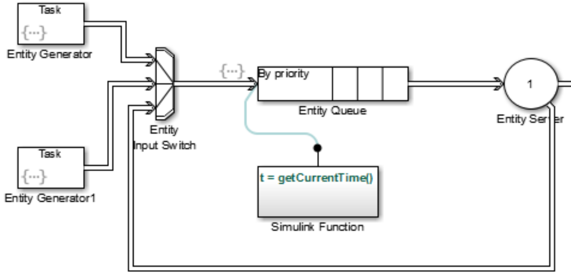
## Example Model for Task Preemption

The example shows preemption(replacement) of low priority tasks by a high priority task in a multitasking processor. In the model, the Entity Server block represents the task processor presented with a capacity to process multiple concurrent tasks.




## Model Behavior and Results

The following graphic shows how the model generates both low and high priority tasks.



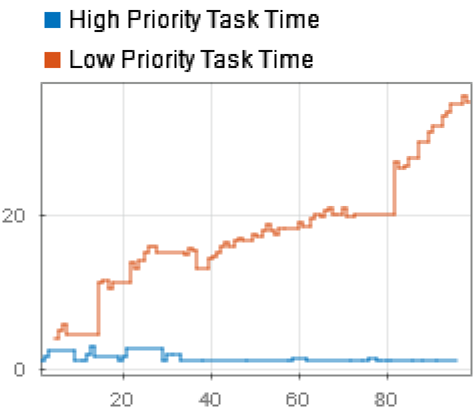

- The top and bottom Entity Generator randomly generate entities that represent high and low priority tasks, respectively. Both blocks use the `exprnd` function to generate random entities. The top block uses `exprnd(3)`, the bottom uses `exprnd(1)`, which requires the Statistics and Machine Learning Toolbox license.
- The Entity Input Switch block merges the paths of the new low priority tasks with previously preempted tasks that are returning from the task processor (server).
- The Simulink Function block runs the `getCurrentTime` function to start a timer on the low priority tasks. When preemption occurs, a downstream Simulink Function block determines the remaining service time of the preempted tasks.
- The Entity Output Switch block merges the paths of the high and low priority tasks. Tasks on the merged path proceed for processing.

An Entity Server block represents a multitasking processor with capacity for multiple tasks.

When preemption occurs, causing the Entity Server block to complete immediately service of all low priority tasks, one of the two Simulink Function blocks calculates the elapsed time of each departing task using the `recordPreferredWaitTimes` and `recordNonPreferredWaitTimes` functions. The two Entity Terminator blocks calls these Simulink Function to calculate the elapsed times.

If the elapsed time of a departing task is less than the service time of the Entity Server block, meaning that preemption forced the task to depart the server early, the Output Switch block feeds the task back to reenter the server. If the elapsed time in the Simulink Function `getCurrentTime` block is *equal* to the service time of the Entity Server block, the server has completed the full service time on the task. The entity terminates in the Entity Terminator block.

The Dashboard Scope block shows the simulation results.




The plot displays wait time for high an low priority tasks. It can be observed that wait time of high priority tasks is significantly decreased.

## See Also

Entity Queue | Entity Server

## Related Examples

- “Model Basic Queuing Systems” on page 2-2

- “Serve High-Priority Customers by Sorting Entities Based on Priority” on page 2-29
- “Model Server Failure” on page 2-27

## More About

- “Overview of Queues and Servers in Discrete-Event Simulation”

# Model Server Failure

| In this section... |
| --- |
| "Server States" on page 2-27 |
| "Use a Gate to Implement a Failure State" on page 2-27 |

## Server States

In some applications, it is useful to model situations in which a server fails. For example, a machine breaks down and later is repaired, or a network connection fails and later is restored. This section explores ways to model failure of a server, and server states.

Server blocks do not have built-in states, so you can design states in any way that is appropriate for your application. Some examples of possible server states are in this table.

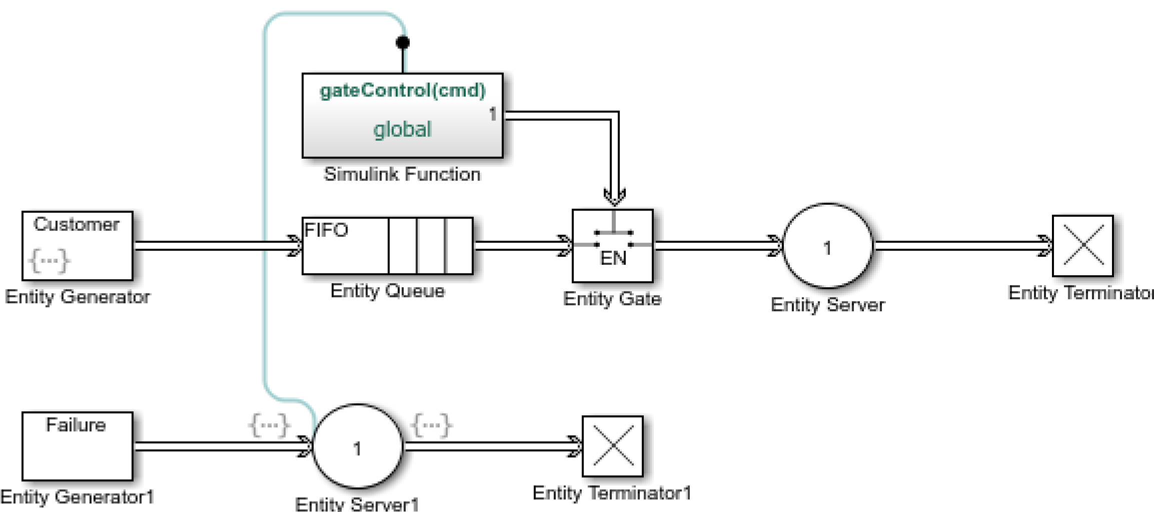
| Server as Communication Channel | Server as Machine | Server as Human Processor |
| --- | --- | --- |
| Transmitting message | Processing part | Working |
| Connected but idle | Waiting for new part to arrive | Waiting for work |
| Unconnected | Off | Off duty |
| Holding message (pending availability of destination) | Holding part (pending availability of next operator) | Waiting for resource |
| Establishing connection | Warming up | Preparing to begin work |

## Use a Gate to Implement a Failure State

For any state that represents a server inability or refusal to accept entity arrivals even though the server is not necessarily full, a common implementation involves an Entity Gate block preceding the server.

The gate prevents entity access to the server whenever the gate control message at the input port at the top of the block carries zero or negative values. The logic that creates the control message determines whether the server is in a failure state. You can implement such logic using the Simulink Function block, using a Message Send block, or using Stateflow® charts to transition among a finite number of server states.

This example shows an instance in which an Entity Gate block precedes a server. The example is not specifically about a failure state, but the idea of controlling access to a server is similar. It models a stochastically occurring failure that lasts for some amount of time.




**Note:** The gate prevents new entities from arriving at the server but does not prevent the current entity from completing its service. If you want to eject the current entity from the server upon a failure occurrence, then you can use the preemption feature of the server to replace the current entity with a high-priority 'placeholder' entity.

## See Also

Entity Queue | Entity Server

## Related Examples

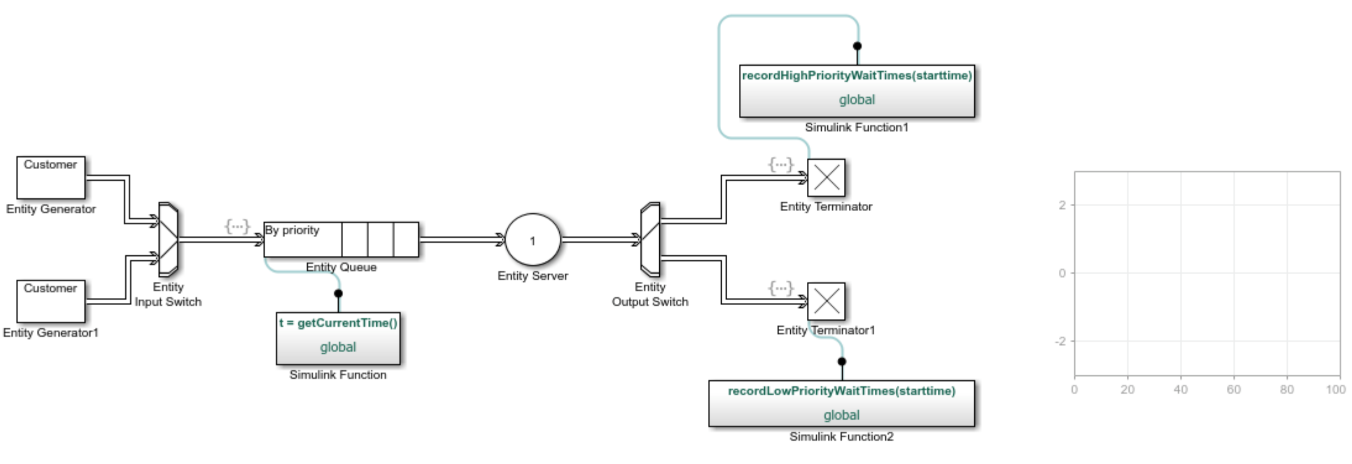
- “Model Basic Queuing Systems” on page 2-2
- “Serve High-Priority Customers by Sorting Entities Based on Priority” on page 2-29
- “Task Preemption in a Multitasking Processor” on page 2-24

## More About

- “Overview of Queues and Servers in Discrete-Event Simulation”

# Serve High-Priority Customers by Sorting Entities Based on Priority

This example shows how to minimize the time required to serve high-priority customers by using a priority queue and Entity Input Switch and Entity Output Switch blocks. Customers are served based on their service priorities. In this example, two types of customers enter a queuing system. One type represents high-priority customers with high urgency. The second type of customers are lower priority and are served with less urgency. The priority queue places high-priority customers ahead of low-priority customers.




### Build the Model

In the model, arriving customers are represented by Entity Generator and Entity Generator1.

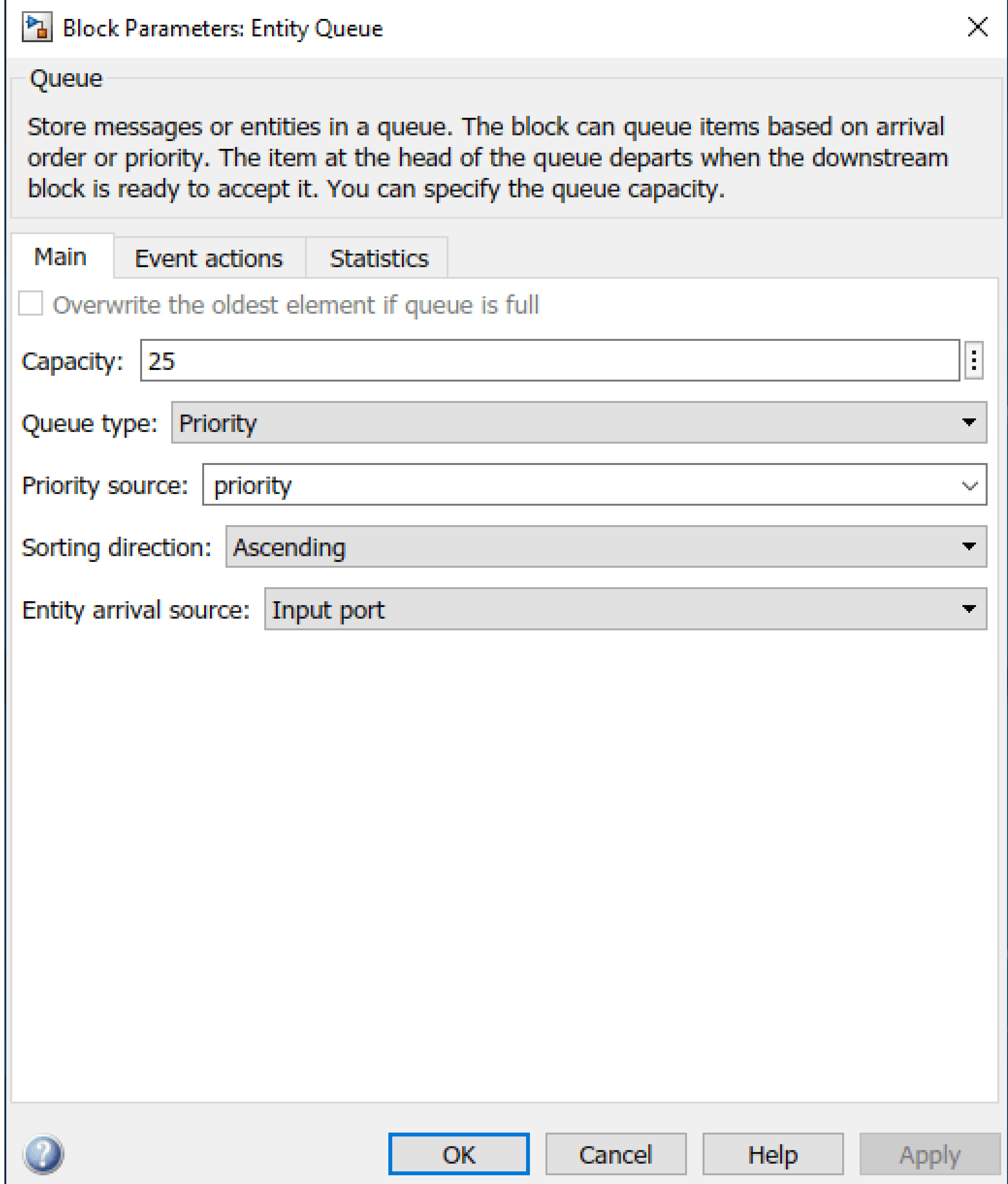
- In the Entity Generator block, customer inter arrival times are generated from an exponential distribution with a mean of 3.
- The Entity Generator block generates entities that have attributes, `priority` and `start time`. The `priority` attribute is set to `1`, which is the service urgency of the customer. The `start time` attribute is also set to `1`, which initializes the start time value used in the model.
- Similarly, Entity Generator1 generates entities whose inter arrival times are generated from an exponential distribution with a mean of `1`. The entities have the same attributes, `priority` and `start time`. The `priority` attribute is set to `2` which is the service urgency of the customer. The `start time` attribute is set to `1`.

The Entity Output Switch block accepts entities generated by the Entity Generator and the Entity Generator1 blocks and forwards them to the priority queue.

The Entity Queue block represents the queueing of the customers and prioritizes them based in their service urgency.

- The **Capacity** of the Entity Queue block is `25`.
- **Queue type** is set to `Priority` to sort the entities based on their priority values.

- **Priority source** is set to `priority`, which is the attribute used to sort the entities.
- **Sorting direction** is set to `Ascending`. Entities with lower values of `priority` are placed at the front of the queue. In this setup, The customers with `priority` value of `1` are prioritized over the customers with a value of `2`.

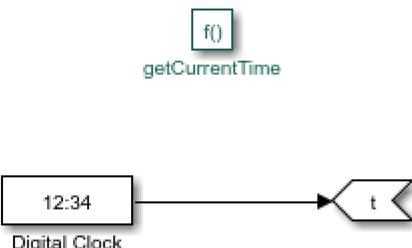



The Simulink Function block is used to timestamp the entities that enter the Entity Queue block.

- In the Entity Queue block, in the **Event actions** tab, in the **Entry** action, the following code is used so that every time an entity enters the block, the `getCurrentTime()` Simulink function is called.

```
entity.starttime = getCurrentTime();
```

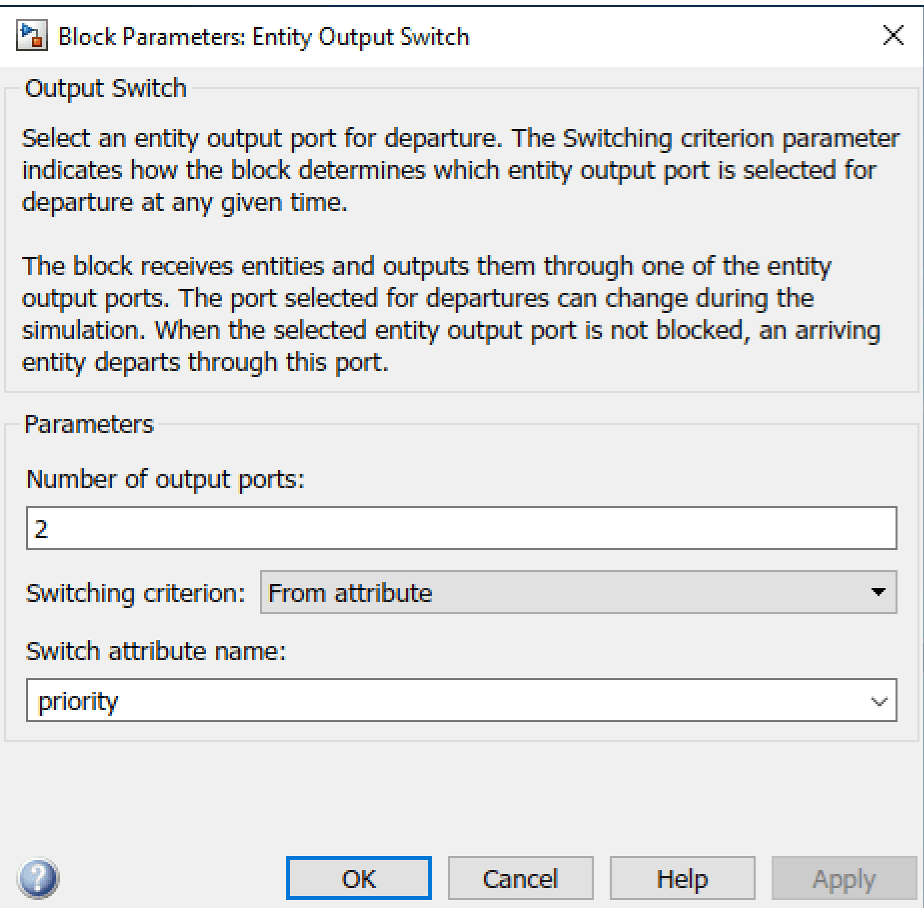
In the Simulink Function block, a Digital Clock block is used to timestamp the entity entering the Entity Queue block.




The Entity Server block represents the service the customer receives.

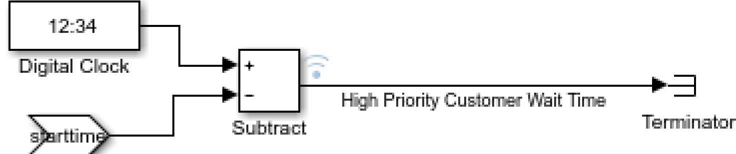
The Entity Output Switch block output the entities for departure.

- **Switching criterion** is set to `From attribute`, which selects the departure path based on an entity attribute.
- **Switch attribute name** is set to `priority`. If the `priority` value is `1`, the block switches to output port `1` and if the `priority` value is `2`, the block switches to output port `2` for entity departure.



When an entity enters the Entity Terminator block, the `recordHighPriorityWaitTimes(starttime)` function is called to calculate the time spent between an entity's arrival at the Entity Queue block and its departure from the Entity Terminator block.

- In the Entity Terminator block, in the **Event actions** tab, in the **Entry**, the `recordHighPriorityWaitTimes(starttime)` function is called.
- The input argument of the function is `startime`, which is the timestamp that was recorded when the entity entered the Entity Queue block.
- The Simulink Function block takes this argument and calculates the difference between the start time and departure time.

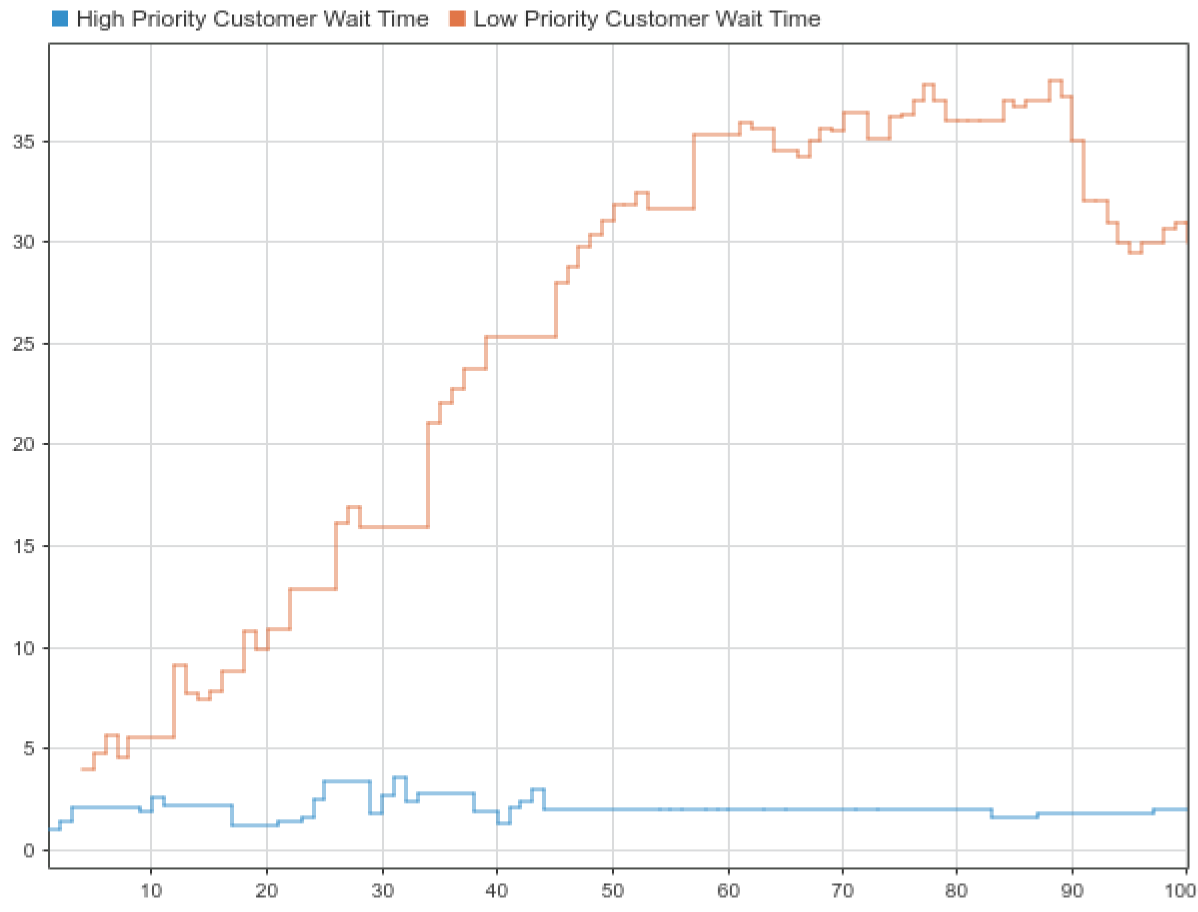



- Similarly, the `recordLowPriorityWaitTimes(starttime)` function calculates the time for the low-priority entities.
- The calculated total service time is displayed by a Dashboard Scope block.

#### Simulate Model and Review Results

The simulation time of the model is set to `100`.

Simulate the model and observe the results displayed in the Dashboard Scope block. The block shows that the waiting time for high-priority customers is significantly less than the low-priority customers.




## See Also

Entity Output Switch | Entity Input Switch | Simulink Function | Entity Queue | Entity Server

## Related Examples


- "Model Basic Queuing Systems" on page 2-2
- "Task Preemption in a Multitasking Processor" on page 2-24
- "Model Server Failure" on page 2-27


## More About

- "Overview of Queues and Servers in Discrete-Event Simulation"

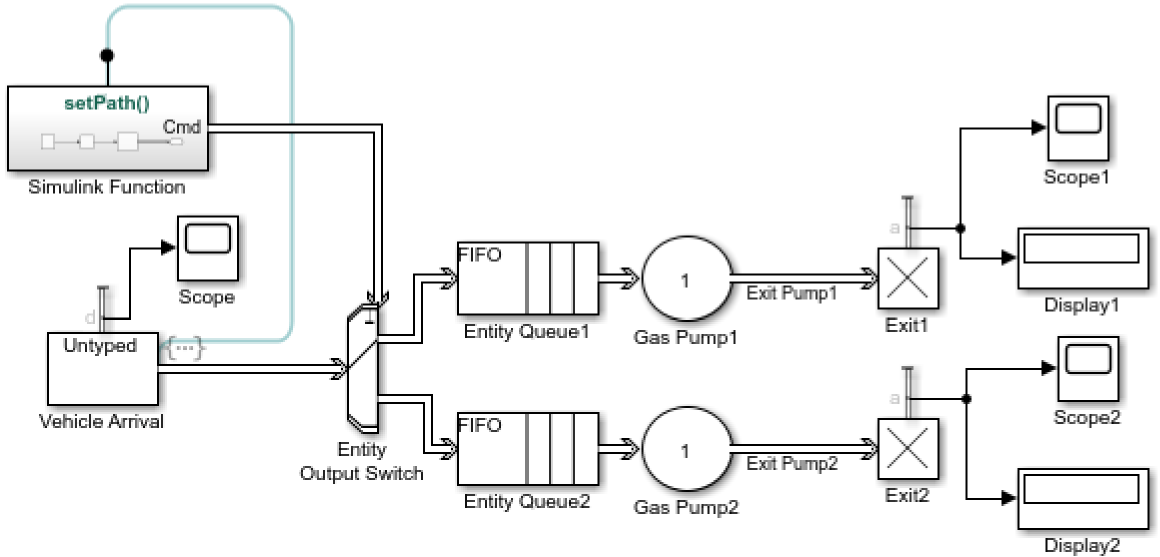
# Routing Techniques


- "Route Vehicles Using an Entity Output Switch Block" on page 3-2
- "Control Output Switch with Event Actions and Simulink Function" on page 3-5
- "Match Entities Based on Attributes" on page 3-7
- "Role of Gates in SimEvents Models" on page 3-9
- "Enable a Gate for a Time Interval" on page 3-11
- "Modeling Message Communication Patterns with SimEvents" on page 3-15
- "Build a Shared Communication Channel with Multiple Senders and Receivers" on page 3-17
- "Model an Ethernet Communication Network with CSMA/CD Protocol" on page 3-22

# Route Vehicles Using an Entity Output Switch Block

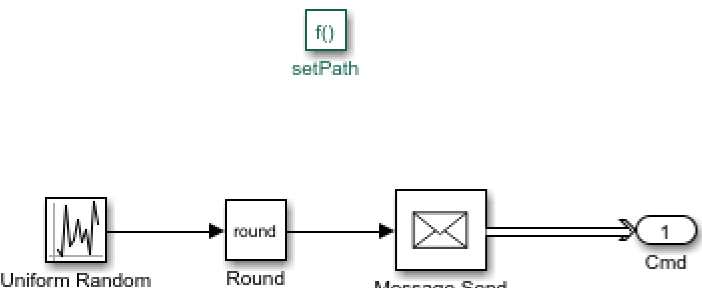
This example shows how to route vehicles to two different pumps in a gas station by controlling an Entity Output Switch block.

In the example, vehicles are generated by an Entity Generator block, which represents vehicle arrival. After their arrival, vehicles are routed to two different gas pumps using an Entity Output Switch block. A Simulink Function block controls the selected output port of the Entity Output Switch block. The vehicle's departure from the Entity Generator block invokes the Simulink Function block.



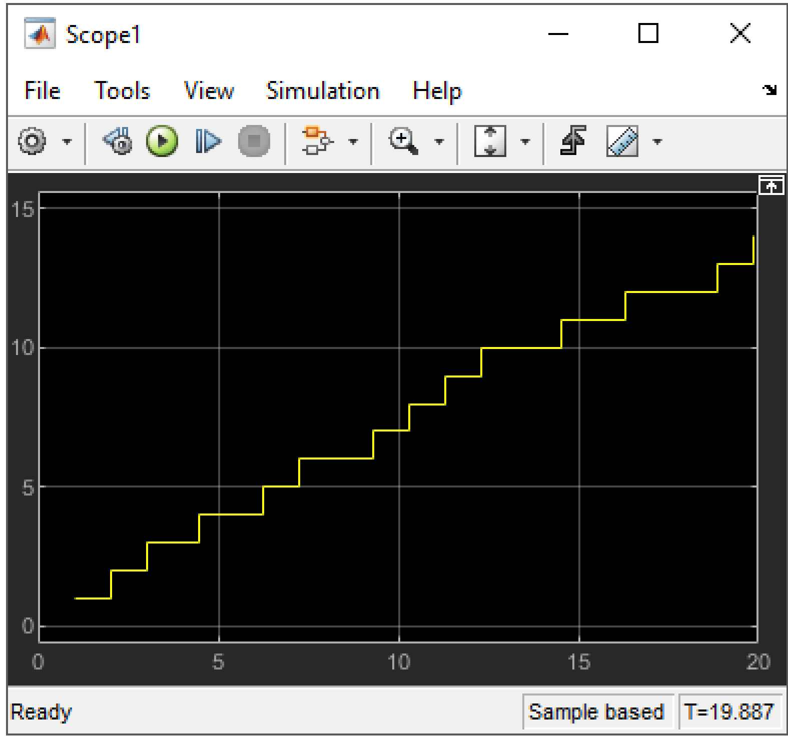

### Control Entity Output Switch Block

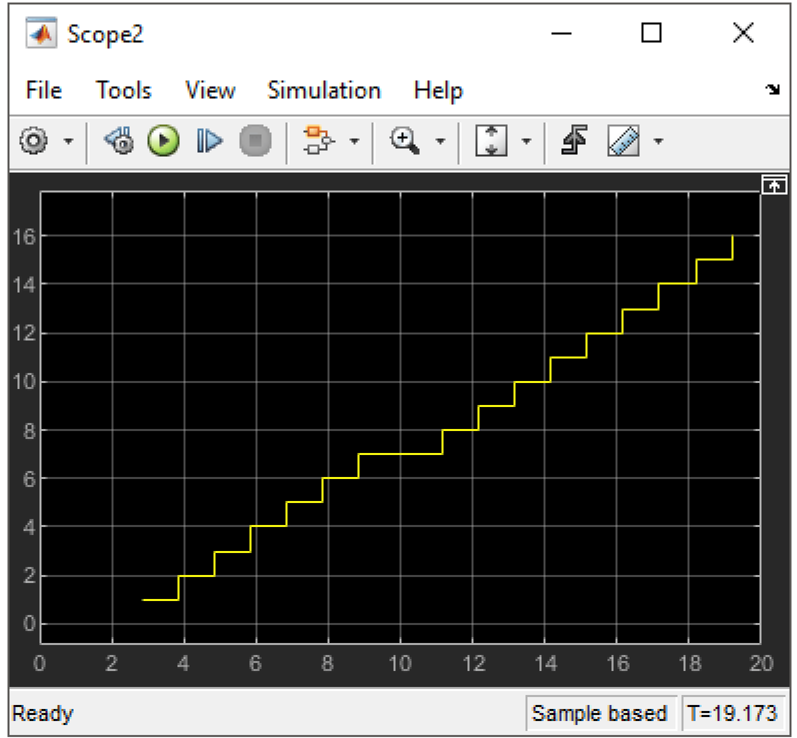
- In the Entity Output Switch Block, set the **Switching Criterion** to `From control port`.
- In the Simulink Function block, use a Uniform Random Number block to generate random numbers between 1 and 2.
- The generated random number is rounded to the integers 1 or 2 by the Round block.
- The integer value of the signal is converted to a message by the Message Send block.
- The output value from the Simulink Function block corresponds to the selected output of the Entity Output Switch block.




### Simulate Model and Review Results

Simulate the model and observe that 14 vehicles use Gas Pump1 and 16 vehicles use Gas Pump2.






## See Also

Entity Server | Entity Terminator | Entity Generator | Entity Output Switch

## More About

- "Control Output Switch with Event Actions and Simulink Function" on page 3-5
- "Match Entities Based on Attributes" on page 3-7

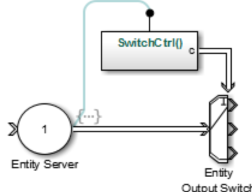
# Control Output Switch with Event Actions and Simulink Function

| In this section... |
|---|
| "Control Output Switch with a Simulink Function Block" on page 3-5 |
| "Specify an Initial Port Selection" on page 3-6 |

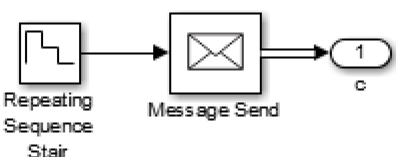
This example shows how to change the selected output port of an Entity Output Switch block to route entities along different paths where a path is selected on a per-entity basis, not on a predetermined time schedule.

## Control Output Switch with a Simulink Function Block

The following example illustrates a scenario in which the Entity Output Switch block is controlled by the Simulink Function block.




1. Double-click the function signature on the Simulink Function block and enter `SwitchCtrl()`.
2. Double-click the Simulink Function block. Add a Repeating Sequence Stair block, and set its **Sample time** parameter to `-1` (inherited), a Message Send block and an Out1 block. Connect the blocks as shown.




3. In the Repeating Sequence Stair block, set the **Vector of output values** to `[3 2 1]`.

   When the Simulink Function block executes, it outputs the next number from a repeating sequence and Message Send block outputs message values `3`, `2`, or `1` based on the sequence of values specified in the Repeating Sequence Stair block.
4. In the Entity Server block, in the **Exit action** field enter this code.

   ```
   SwitchCtrl();
   ```

When service in the Entity Server block is complete, the entity exits the block and advances to the Entity Output Switch block. The departure of the entity from the Entity Server block calls the

`SwitchCtrl()` function which activates the Simulink Function block. Then the output message of the Simulink Function block determines which output port the entity uses when it departs the Entity Output Switch block.

## Specify an Initial Port Selection

When the Entity Output Switch block uses an input message, the block might attempt to use the message before its first sample time hit. If the initial value of the message is out of range (for example, it is unavailable). You should then specify the initial port selection in the Entity Output Switch block's dialog box. To achieve this, you can follow these steps.

1. In the Entity Output Switch, select `From control port` as the **Switching criterion**.
2. Set **Initial port selection** to the desired initial port. The value must be an integer between 1 and **Number of output ports**. The Entity Output Switch block uses **Initial port selection** until the first control port message arrives.

## See Also

Entity Gate | Entity Input Switch | Entity Output Switch | Entity Replicator

## Related Examples


- "Route Vehicles Using an Entity Output Switch Block" on page 3-2
- "Serve High-Priority Customers by Sorting Entities Based on Priority" on page 2-29
- "Model Traffic Intersections as a Queuing Network" on page 5-12
- "Enable a Gate for a Time Interval" on page 3-11


## More About

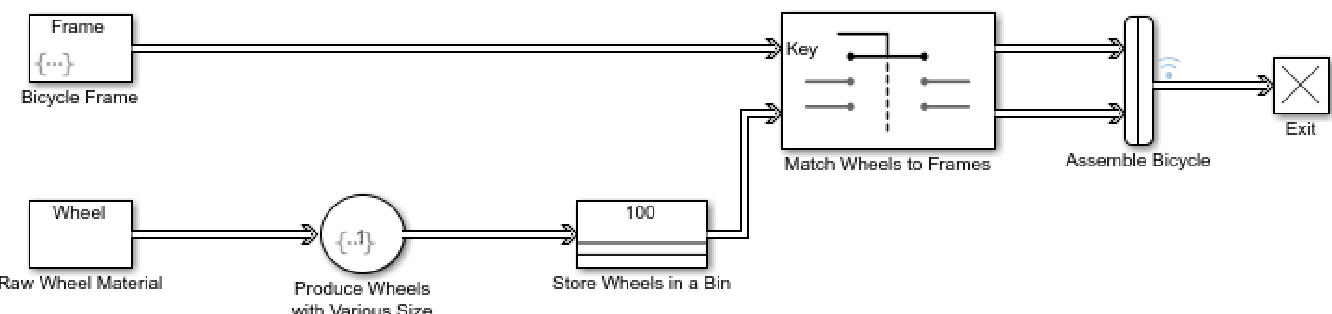

- "Role of Entity Ports and Paths"
- "Role of Gates in SimEvents Models" on page 3-9

# Match Entities Based on Attributes

This example shows how to build a model to store and match entities representing bicycle components. The model uses an Entity Store block for storage and an Entity Selector block to match a set of bicycle wheels to the corresponding size frame for assembly.

### Produce Bicycle Frames and Wheels

Suppose that you are modeling an assembly line that produces bicycles sized small, medium, and large. Each bicycle is manufactured by matching the set of wheels to the corresponding size frame. The wheels are produced at a facility. The frames are ordered from a supplier and they arrive at the facility ready to assemble. Given this arrangement, frame arrival rate is slower than the wheel production rate, and the set of wheels are stored in a bin.

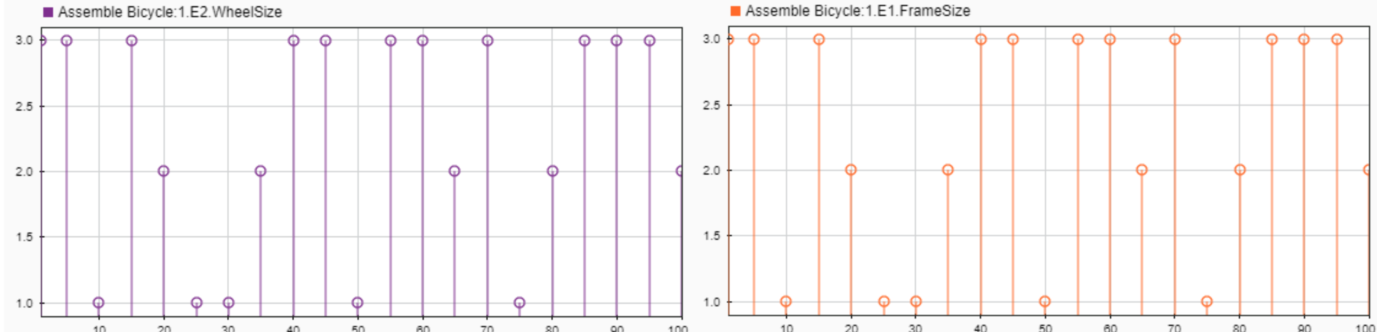



In the model:

- The Bicycle Frame Block generates `Frame` with **Period** `5` to represent slow arrival rate of bicycle frames. A `Frame` can be of size `1`, `2`, or `3`, and each `Frame` carries an attribute `FrameSize` that represents its size.
- The Raw Wheel Material Block generates `Wheel` with **Period** `1`. Each `Wheel` carries a `WheelSize` attribute that represents the size of each generated wheel. The initial value of `WheelSize` is set to `0`.
- In the Produce Wheels with Various Size block, wheels are set to size `1`, `2`, or `3`.
- The Entity Store block is named Store Wheels in a Bin and it stores the processed wheels.
- The Entity Selector block is named Match Wheels to Frames and it matches `'WheelSize'` to the corresponding `'FrameSize'`.

### Simulate the Model and Review Results

Simulate the model. Open the Simulation Data Inspector. Observe that, for the bicycle assembly, the size of the set of wheels and the frames are exactly matched by the Entity Selector block. Although the wheels are generated faster, they are stored in the Entity Store block, and they wait to be matched to the arriving frames for assembly.




## See Also

Composite Entity Creator | Entity Selector | Entity Store | Entity Gate | Entity Server

## Related Examples

- "Control Output Switch with Event Actions and Simulink Function" on page 3-5
- "Enable a Gate for a Time Interval" on page 3-11

## More About

- "Working with Entity Attributes and Entity Priorities" on page 1-32
- "Role of Entity Ports and Paths"
- "Role of Gates in SimEvents Models" on page 3-9

# Role of Gates in SimEvents Models

| In this section... |
| --- |
| “Overview of Gate Behavior” on page 3-9 |
| “Gate Behavior” on page 3-9 |

## Overview of Gate Behavior

By design, certain blocks change their availability to arriving entities depending on the circumstances. For example,

- A queue or server accepts arriving entities as long as it is not already full to capacity.
- An input switch accepts an arriving entity through a single selected entity input port but forbids arrivals through other entity input ports.

Some applications require more control over whether and when entities advance from one block to the next. A gate provides flexible control via its changing status as either open or closed: by definition, an open gate permits entity arrivals as long as the entities would be able to advance immediately to the next block, while a closed gate forbids entity arrivals. You configure the gate so that it opens and closes under circumstances that are meaningful in your model.

For example, you might use a gate

- To create periods of unavailability of a server. For example, you might be simulating a manufacturing scenario over a month long period, where a server represents a machine that runs only 10 hours per day. An enabled gate can precede the server, to make the server's availability contingent upon the time.
- To make departures from one queue contingent upon departures from a second queue. A release gate can follow the first queue. The gate's control input determines when the gate opens, based on decreases in the number of entities in the second queue.
- With the `First port that is not blocked` mode of the Entity Output Switch block. Suppose each entity output port of the switch block is followed by a gate block. An entity attempts to advance via the first gate; if it is closed, then the entity attempts to advance via the second gate, and so on.

## Gate Behavior

The Entity Gate block offers these fundamentally different kinds of gate behavior:

- The enabled gate, which uses a control port to determine time intervals over which the gate is open or closed
- The release gate, which uses a control port to determine a discrete set of times at which the gate is instantaneously open. The gate is closed at all other times during the simulation.

---

**Tip** Many models follow a gate with a storage block, such as a queue or server.

---

## See Also

Entity Gate | Entity Input Switch | Entity Output Switch | Entity Replicator

## Related Examples

- “Control Output Switch with Event Actions and Simulink Function” on page 3-5
- “Enable a Gate for a Time Interval” on page 3-11

## More About

- “Role of Entity Ports and Paths”

# Enable a Gate for a Time Interval

| In this section... |
|---|
| “Behavior of Entity Gate Block in Enabled Mode” on page 3-11 |
| “Sense an Entity Passing from A to B and Open a Gate” on page 3-11 |
| “Control Joint Availability of Two Servers” on page 3-13 |

## Behavior of Entity Gate Block in Enabled Mode

The Entity Gate block uses a control signal at the input port at the top of the block to determine when the gate is open or closed:

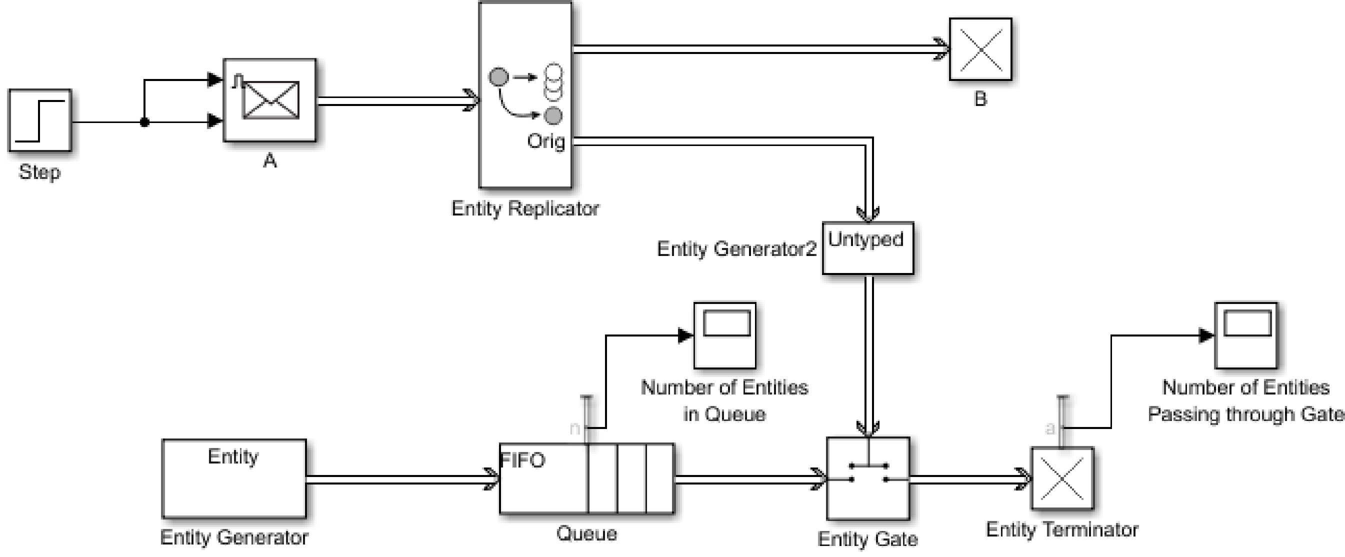
- When an entity with a positive payload arrives at the enable port at the top of the block, the gate is open and an entity can arrive as long as it would be able to advance immediately to the next block.
- When an entity with zero or negative payload arrives at the enable port at the top of the block, the gate is closed and no entity can arrive.

Because that incoming signal can remain positive for a time interval of arbitrary length, an enabled gate can remain open for a time interval of arbitrary length. The length can be zero or a positive number.

Depending on your application, the gating logic can arise from time-driven dynamics, state-driven dynamics, a SimEvents block's statistical output signal, or a computation involving various types of signals. To see the ready-to-use common design patters including the Entity gate block, see “SimEvents Common Design Patterns”.

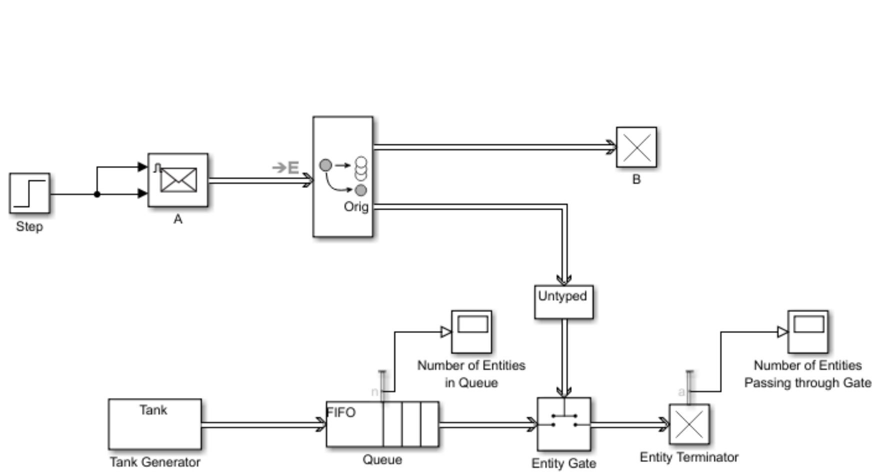
## Sense an Entity Passing from A to B and Open a Gate

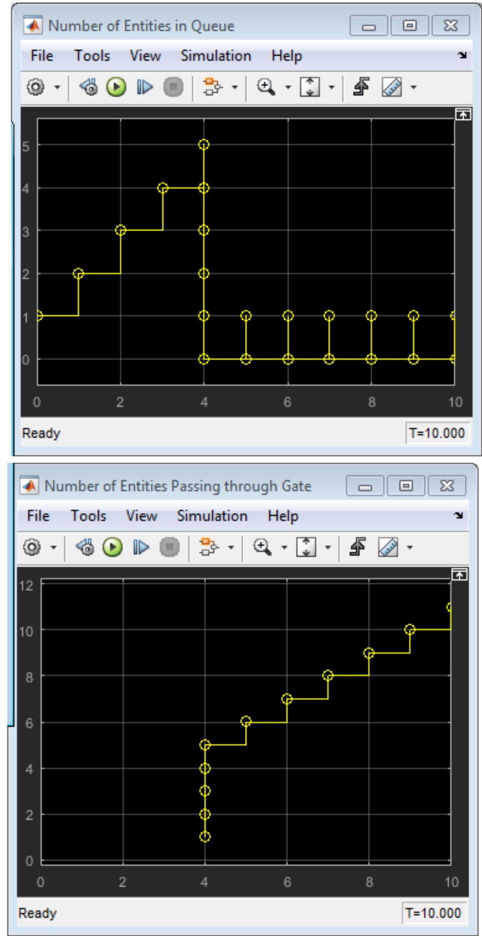
This example shows how to use the `Sense an Entity Passing from A to B and Open a Gate` design pattern. In this example, the Step block generates a step signal at time 4. This signal passes through the Message Send block A. The Entity Replicator block duplicates the entity and passes it to B. It uses the original entity to trigger an event-based entity to enable the Entity Gate block.




1. In a new model, drag the blocks shown in the example and relabel and connect them as shown. For convenience, start with the `Sense an Entity Passing from A to B and Open a Gate` design pattern.
2. In the Step block, set the **Step time** parameter to 4.
3. In the A (Message Send) block, select the **Show enable port** check box. Selecting this check box lets the Step block signal enable the A block to send a message to the Entity Replicator block.
4. In the Entity Generatorblock, in the Entity type tab:

   a. Name the entity type `Entity`.
   b. Add an attribute named `Capacity` with an initial value of `0`.
5. In the Entity Queue block, in the **Statistics** tab, select **Number of entities in block, n**.
6. Save and run the model. Observe the number of entities passing through the gate and the number of entities in the queue at time 4.




## Control Joint Availability of Two Servers

Suppose that each entity undergoes two processes, one at a time, and that the first process does not start if the second process is still in progress for the previous entity. Assume for this example that it is preferable to model the two processes using two Single Server blocks in series rather than one Single Server block whose service time is the sum of the two individual processing times; for example, you might find a two-block solution more intuitive or you might want to access the two Single Server blocks' utilization output signals independently in another part of the model.

If you connect a queue, a server, and another server in series, then the first server can start serving a new entity while the second server is still serving the previous entity. This does not accomplish the stated goal. The model needs a gate to prevent the first server from accepting an entity too soon, that is, while the second server still holds the previous entity.

## See Also

Entity Gate | Entity Input Switch | Entity Output Switch | Entity Replicator | Message Send

## Related Examples

- “Control Output Switch with Event Actions and Simulink Function” on page 3-5

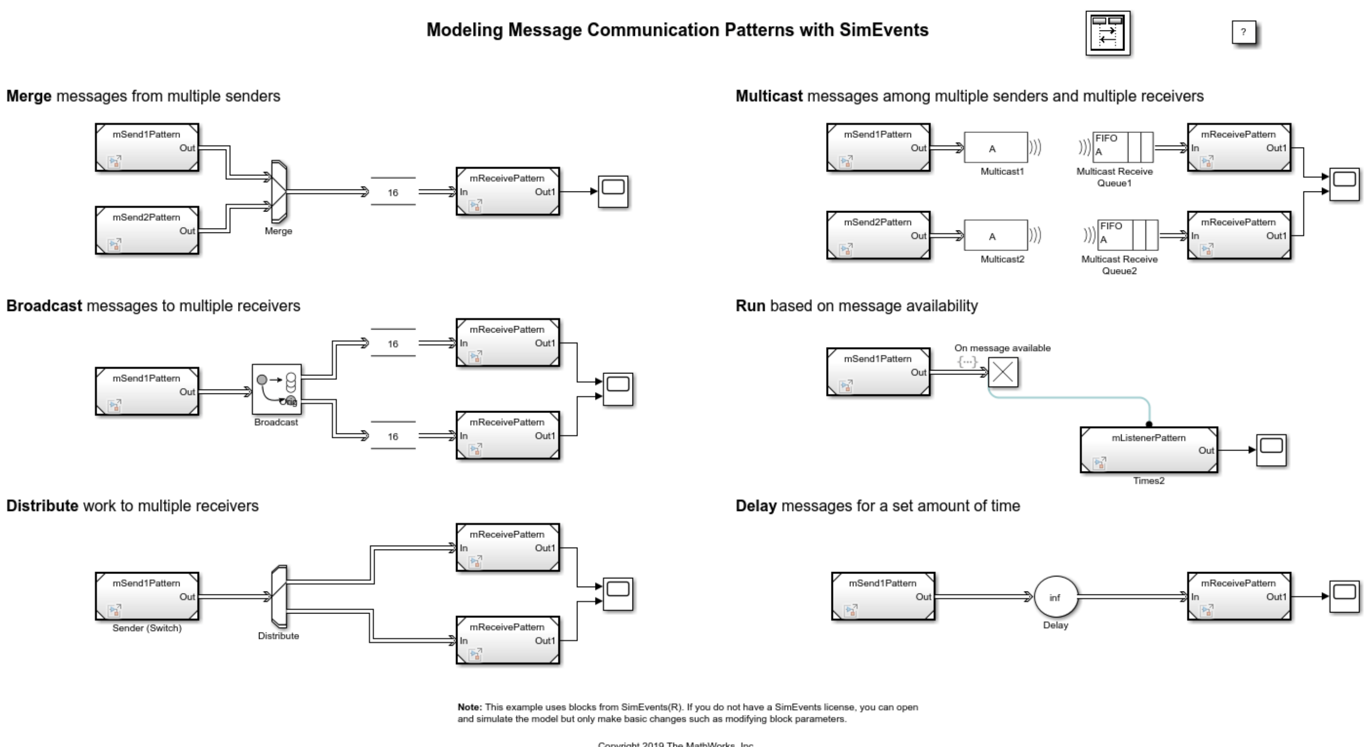
## More About

- “Role of Entity Ports and Paths”
- “Role of Gates in SimEvents Models” on page 3-9

# Modeling Message Communication Patterns with SimEvents

This example shows how to create common communication patterns using SimEvents®. In message-based communication models, you can use SimEvents® to model and simulate middleware, and investigate the effects of communication and the environment on your distributed architecture.

The systems in this example represent common communication patterns created by using SimEvents® blocks that can be used to simulate various network types, such as cabled or wireless communication, and channel behavior such as failure, or packet loss.




The communication patterns involve:

- Merging messages from multiple senders.
- Broadcasting messages to multiple receivers.
- Distributing work to multiple receivers.
- Multicasting messages among multiple senders and multiple receivers.
- Running a component based on message availability and data.
- Delaying messages for a set amount of time.

To create more complex networks and channel behavior, use combinations of these simple patterns.

By using these patterns, you can model:

- *N* -to- *n* communication with multiple senders and receivers with an ideal channel with communication delay. For an example, see “Build a Shared Communication Channel with Multiple Senders and Receivers”.

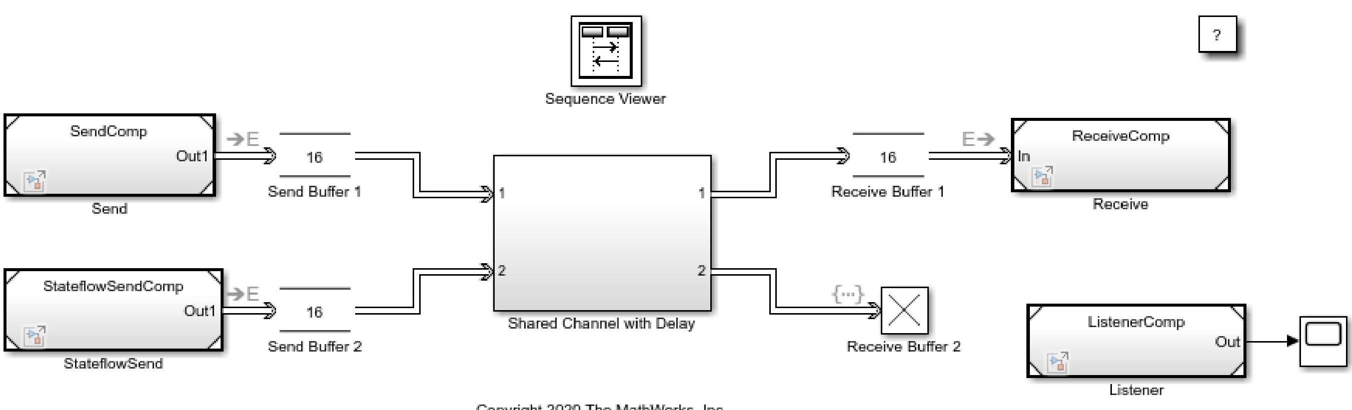
- *N* -to- *n* communication with channel failure and packet loss. For an example, see “Model Wireless Message Communication with Packet Loss and Channel Failure”.
- An *N* -to- *n* Ethernet communication network with an inter-component communication protocol. For an example, see “Model an Ethernet Communication Network with CSMA/CD Protocol”.

# Build a Shared Communication Channel with Multiple Senders and Receivers

This example shows how to model communication through a shared channel with multiple senders and receivers by using Simulink® messages, SimEvents®, and Stateflow®.

For an overview about messages, see "Simulink Messages Overview".

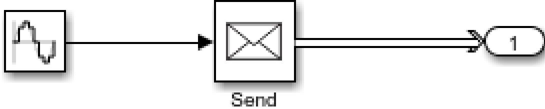
In this model, there are two software components that send messages and two components that receive messages. The shared channel transmits messages with an added delay. SimEvents® blocks are used to create custom communication behavior by merging the message lines, and copying and delaying messages. A Stateflow® chart is used in a send component to send messages based on a decision logic.



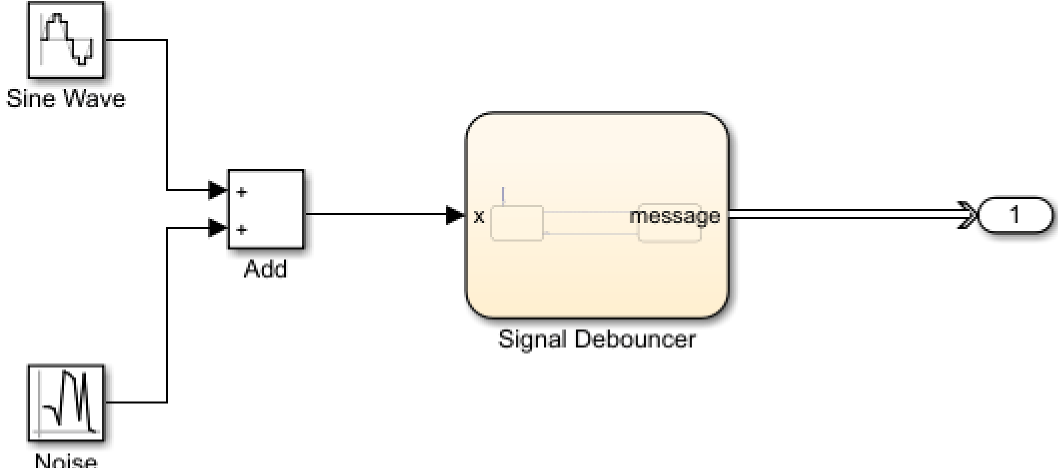

### Create Components to Send Messages

In the model, there are two software components that output messages, Send and StateflowSend.

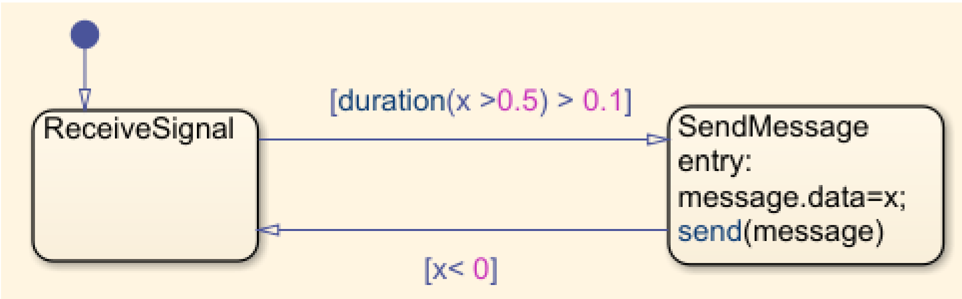
In the Send component, the Sine Wave block is the signal source. The block generates a sine wave signal with an amplitude of `1`. The block's sample time is `0.1`. The Send block converts the signal to a message that carries the signal value as data. The Send component sends messages to Send Buffer 1.




In the StateflowSend component, another Sine Wave block generates a sine wave signal and a Noise block injects noise into the signal. The Noise block outputs a signal whose values are generated from a Gaussian distribution with mean of `0` and variance of `1`. The sample time of the block is `0.1`.




The Stateflow® chart represents a simple logic that filters the signal and decides whether to send messages. If the value of the signal is greater than `0.5` for a duration greater than `0.1`, then the chart sends a message that carries the signal value. If the signal value is below `0`, then the chart transitions to the `ReceiveSignal` state. The StateflowSend component sends messages to Send Buffer 2.



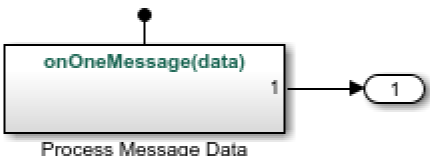

For more information about creating message interfaces, see “Establish Message Send and Receive Interfaces Between Software Components”.

### Create Components to Receive Messages

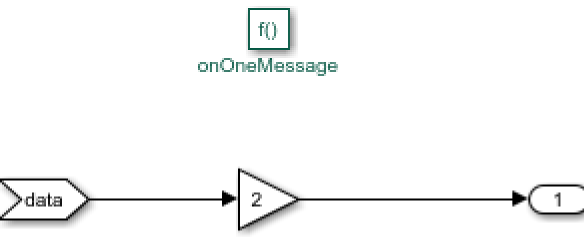
In the model, there are two software components that receive messages, Receive and Listener.

In the Receive component, a Receive block receives messages and converts the message data to signal values.

In the Listener component, there is a Simulink Function block. The block displays the function, onOneMessage(data), on the block face.

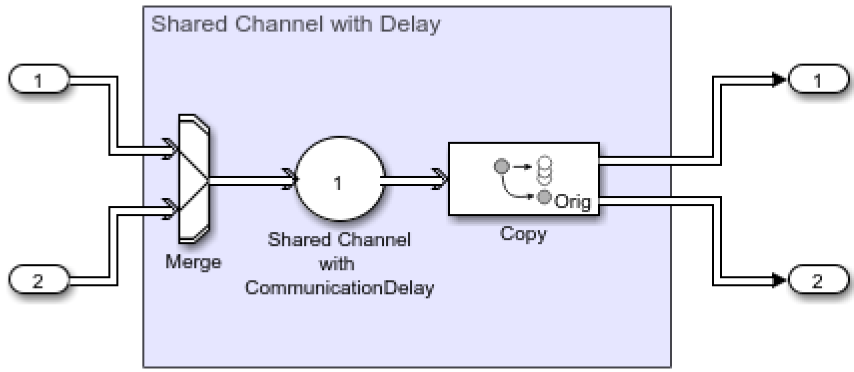



When a message arrives at Receive Buffer 2, the Listener block is notified and it takes the argument `data`, which is the value from the message data, as the input signal. In the block, `data` values are multiplied by `2`. The block outputs the new data value.




#### Routing Messages using SimEvents®

In the shared channel, the message paths originating from the two message-sending components are merged to represent a shared communication channel.




A SimEvents® Entity Input Switch block merges the message lines. In the block:

- **Number of input ports** specifies the number of message lines to be merged. The parameter value is `2` for two message paths.
- **Active port selection** specifies how to select the active port for message departure. If you select `All`, all of the messages arriving at the block are able to depart the block from the output port. If you select `Switch`, you can specify the logic that selects the active port for message departure. For this example, the parameter is set to `All`.

A SimEvents® Entity Server block is used to represent message transmission delay in the shared channel. In the block:

- **Capacity** is set to `1`, which specifies how many messages can be processed at a time.

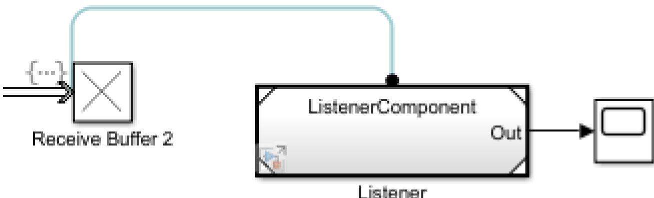
- **Service time value** is set to 1, which specifies how long it takes to process a message

A SimEvents® Entity Replicator block is used to generate identical copies of messages. In the block:

- **Replicas depart from** specifies if the copies leave the block from separate output ports or the same output port as the original messages. The parameter is set to `Separate output ports`.
- **Number of replicas** is set to 1, which specifies the number of copies generated for each message.
- **Hold original entity until all replicas depart** holds the original message in the block until all of its copies depart the block.

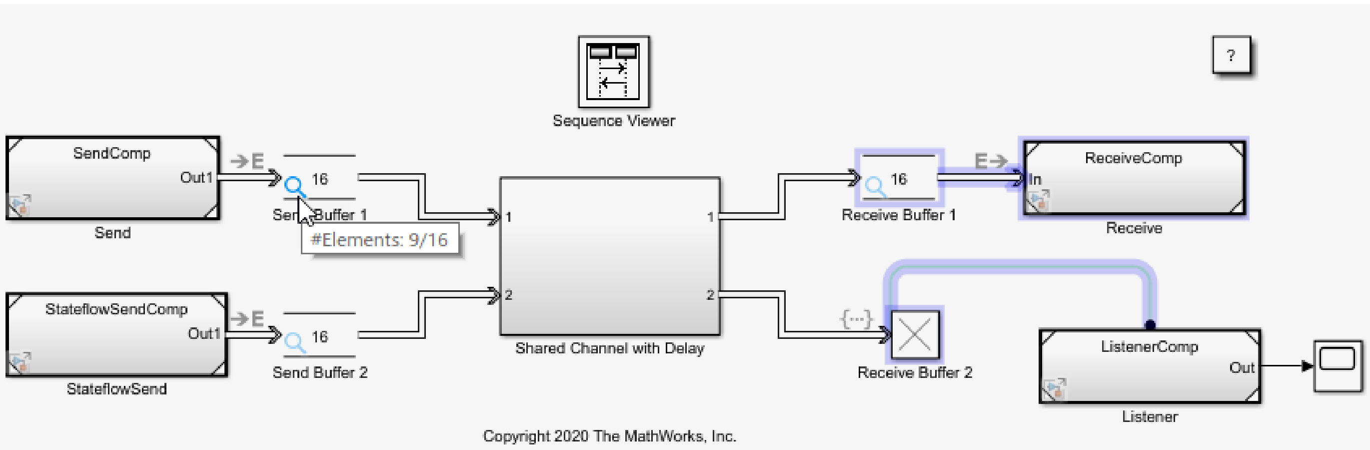
A SimEvents® Entity Terminator block is used to model Receive Buffer 2. In the block:

- Under the **Event actions** tab, in the **Entry action** field, you can specify MATLAB code that performs calculations or Simulink® function calls that are invoked when the message enters the block. In this example, `onOneMessage(entity)` is used to notify the Simulink Function block in the Listener component. To visualize the function call, under **Debug** tab, select **Information Overlays** and then **Function Connectors**.



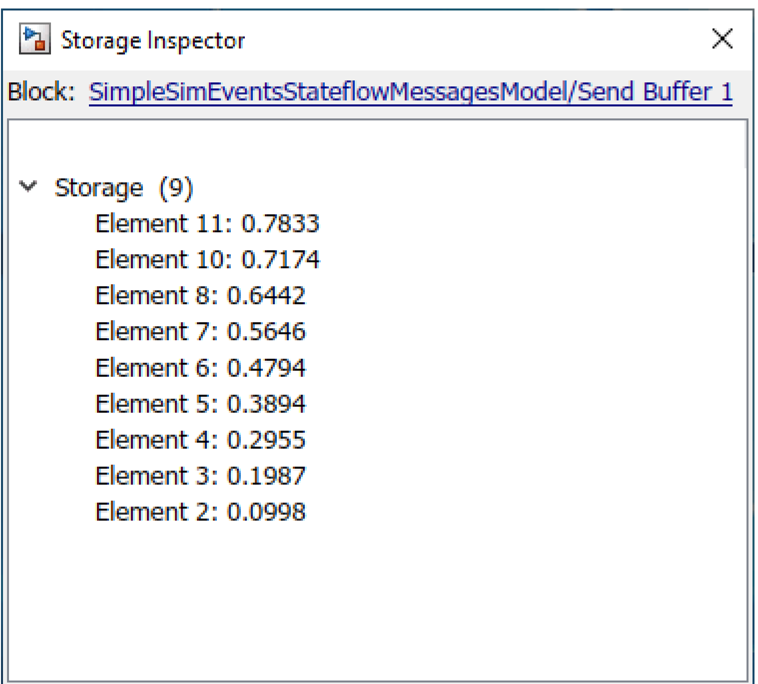

### Simulate the Model and Review Results

Simulate the model. Observe that the animation highlights the messages flowing through the model. You can turn off the animation by right-clicking on the model canvas and setting **Animation Speed** to `None`.



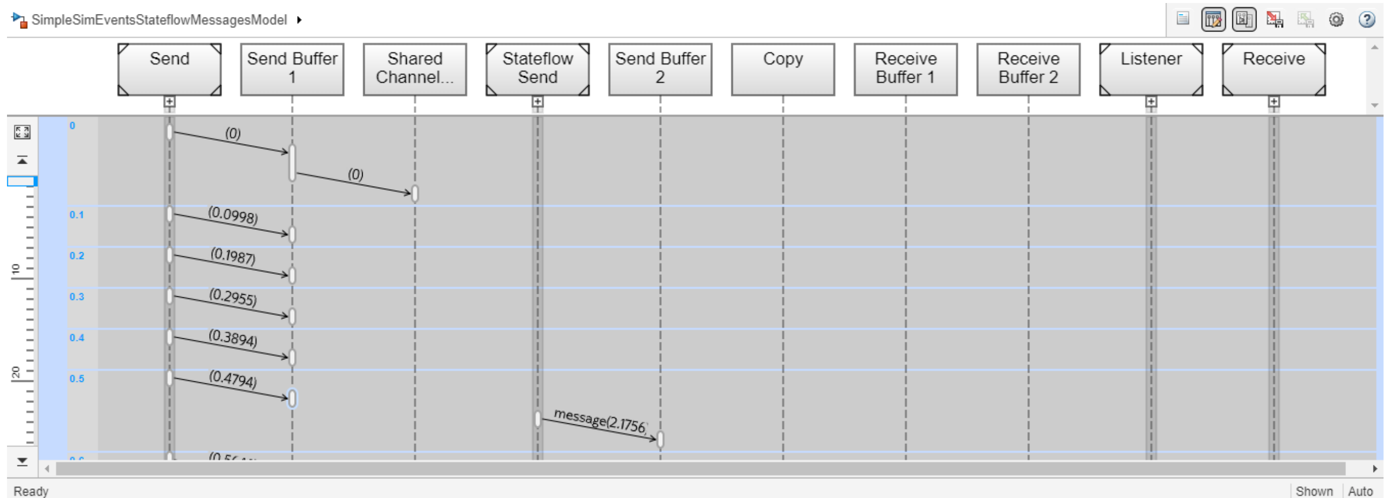

When you pause the animation, a magnifying glass appears on the blocks that store messages. If you point to the magnifying glass, you see the number of messages stored in the block.

To observe which messages are stored in the block, click the magnifying glass to open the Storage Inspector. For instance, the graphic below illustrates the messages stored in Send Buffer 1.



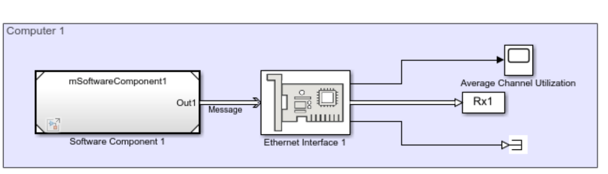

Turn the animation off and open the Sequence Viewer block to observe the Simulink Function calls and the flow of messages in the model.

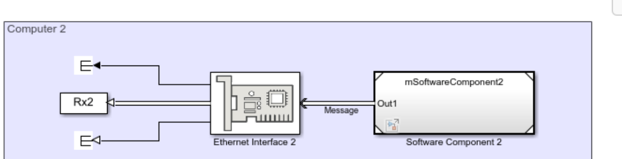
For instance, observe the simulation time `0`, during which a message carrying value `0` is sent from the Send component to Send Buffer 1. From simulation time `0.1` to `0.5`, the Send component keeps sending messages to Send Buffer 1 with different data values. At time `0.5`, the StateflowSend component sends a message to Send Buffer 2. For more information about using the Sequence Viewer block, see “Use the Sequence Viewer to Visualize Messages, Events, and Entities”.

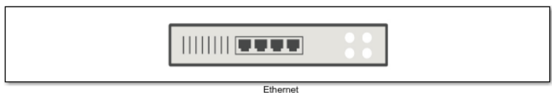


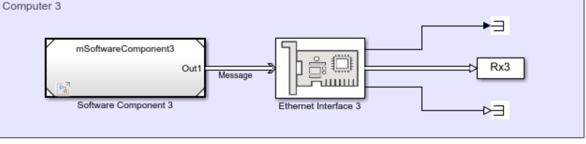
# Model an Ethernet Communication Network with CSMA/CD Protocol

This example shows how to model an Ethernet communication network with CSMA/CD protocol using Simulink® messages and SimEvents®. In the example, there are three computers that communicate through an Ethernet communication network. Each computer has a software component that generates data and an Ethernet interface for communication. Each computer attempts to send the data to another computer with a unique MAC address. An Ethernet interface controls a computer's interaction with the network by using a CSMA/CD communication protocol. The protocol is used to respond to collisions that occur when multiple computers send data simultaneously. The Ethernet component represents the network and the connection between the computers.



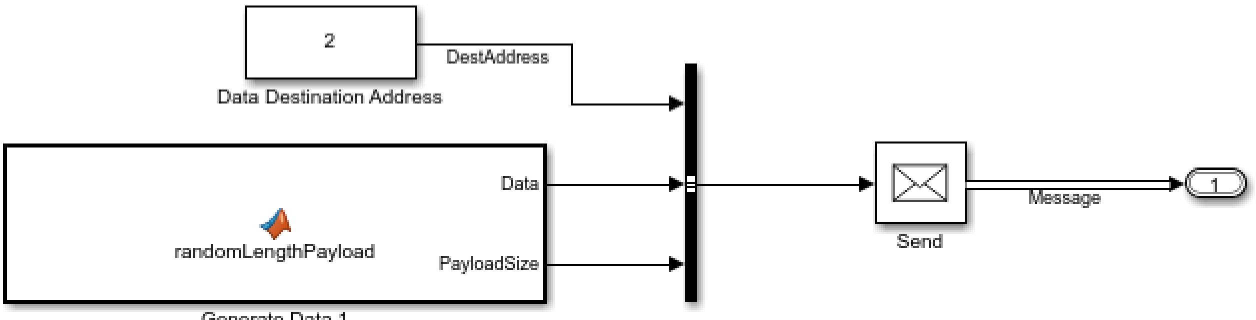

Copyright 2020 The MathWorks, Inc.

### Software Components

In the model, each software component generates data (payload) and combines the data, its size, and its destination into a message. Then, the message is sent to the Ethernet interface for communication.

In each Software Component subsystem:

- A MATLAB Function block generates data with a size between `46` and `1500` bytes [ 1 ].
- A Constant block assigns destination addresses to data.
- A Bus Creator block converts the `Data`, `PayloadSize`, and `DestAddress` signals to a nonvirtual bus object called `dataPacket`.
- A Send block converts `dataPacket` to a message.
- An Outport block sends the message to the Ethernet interface for communication.

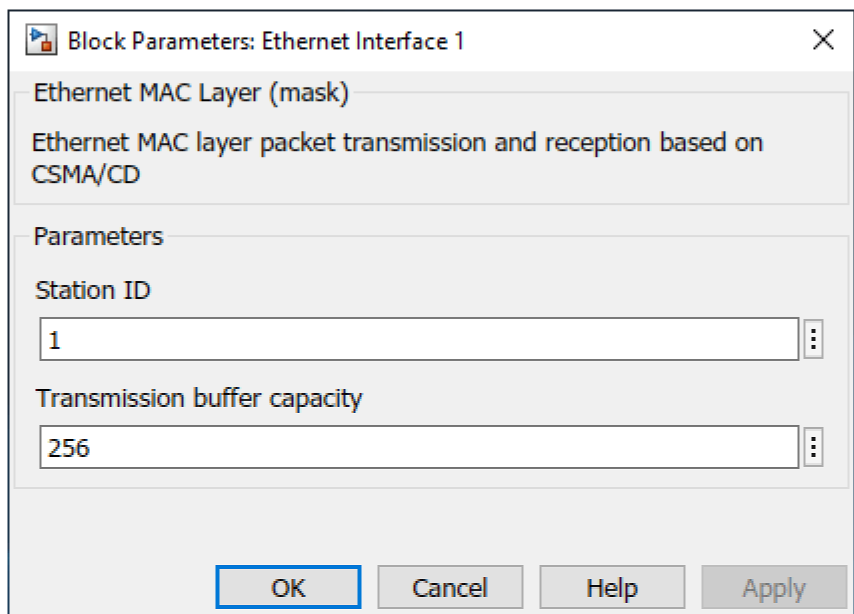



Each computer generates data with a different rate. You can change the data generation rate from the MATLAB Function block's sample time.

To learn the basics of creating message send and receive interfaces, see “Establish Message Send and Receive Interfaces Between Software Components”.

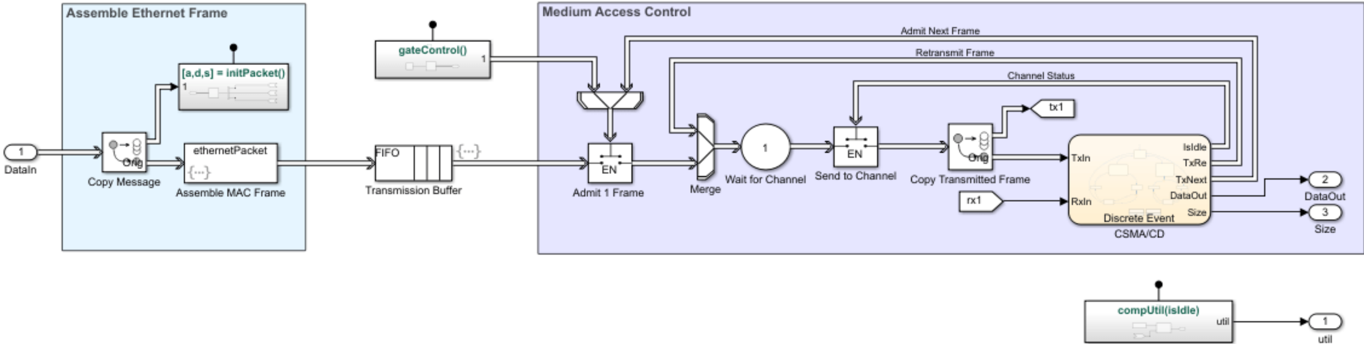
### Ethernet Interface

Double-click Ethernet Interface 1. Observe that you can specify the **Station ID** and **Transmission buffer capacity**.




The Ethernet Interface subsystems have three main parts:

1. Assemble Ethernet Frame — Converts an incoming message to an Ethernet (MAC) frame.
2. Transmission Buffer — Stores Ethernet frames for transmission.
3. Medium Access Control — Implements a CSMA/CD protocol for packet transmission [ 2 ].



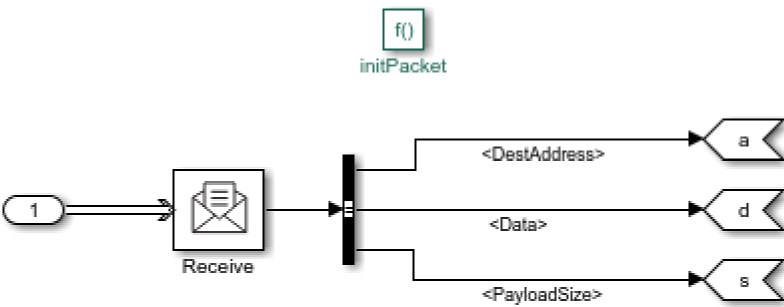

### Assemble Ethernet Frame

The Assemble Ethernet Frame blocks convert messages to Ethernet frames by attaching Ethernet-specific attributes to the message [ 1 ].

In the packet assembly process:

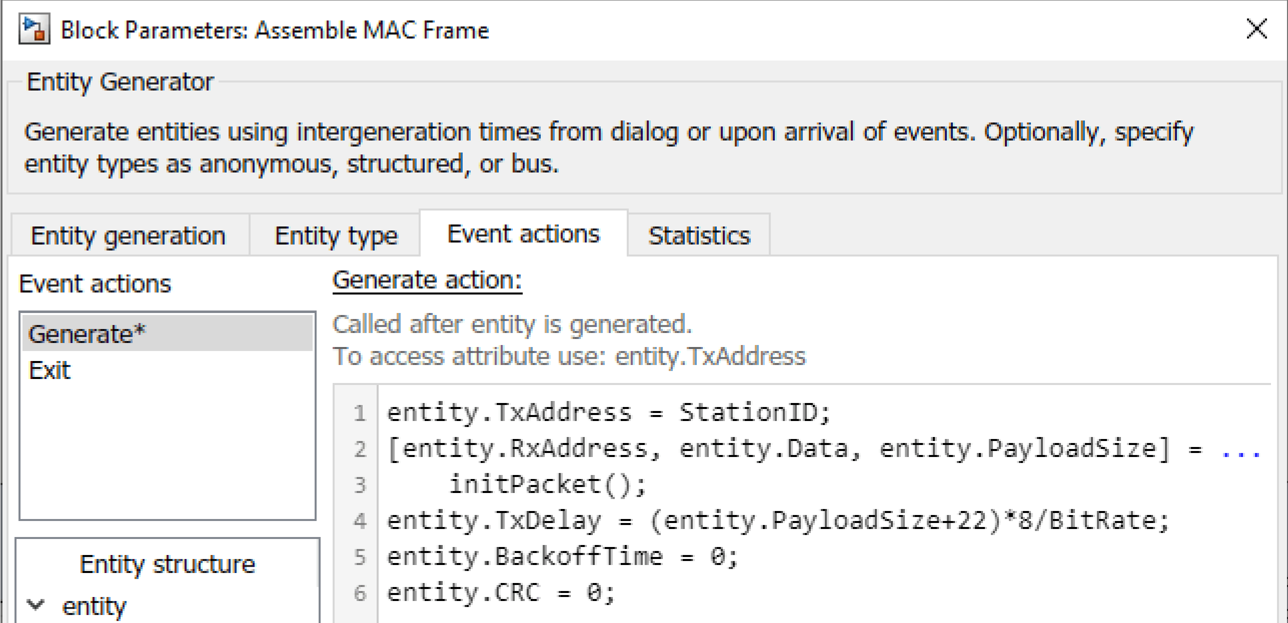
- A SimEvents® Entity Replicator block labeled Copy Message copies an incoming message. The original message is forwarded to a SimEvents® Entity Generator block labeled Assemble MAC Frame. Because the Entity Generator block **Generation method** parameter is set to `Event-based`, it immediately produces an entity when the original message arrives at the block. A copy of the message is forwarded to a Simulink Function block with the `initPacket()` function. The terms *message* and *entity* are used interchangeably between Simulink® and SimEvents®.

- The Simulink Function block transfers the data, its size, and its destination address to the Assemble MAC Frame block for frame assembly.




- The Assemble MAC Frame block generates the Ethernet frames that carry both Ethernet-specific attributes and values transferred from the Simulink Function block.

Assemble MAC Frame block calls the `initPacket()` function as an action that is invoked by each frame generation event.




These are the attributes of the generated Ethernet frame:

- `entity.TxAddress` is `StationID`.
- `entity.RxAddress`, `entity.Data`, and `entity.PayloadSize` are assigned the values from the Simulink Function block.
- `entity.TxDelay` is the transmission delay. It is defined by the payload size and the bitrate. The `Bitrate` parameter is specified by an initialization function in the Model Properties.
- `entity.CRC` is the cyclic redundancy check for error detection.

#### Transmission Buffer

The transmission buffer stores entities before transmission by using a first-in-first-out (FIFO) policy. The buffer is modeled by a Queue block.

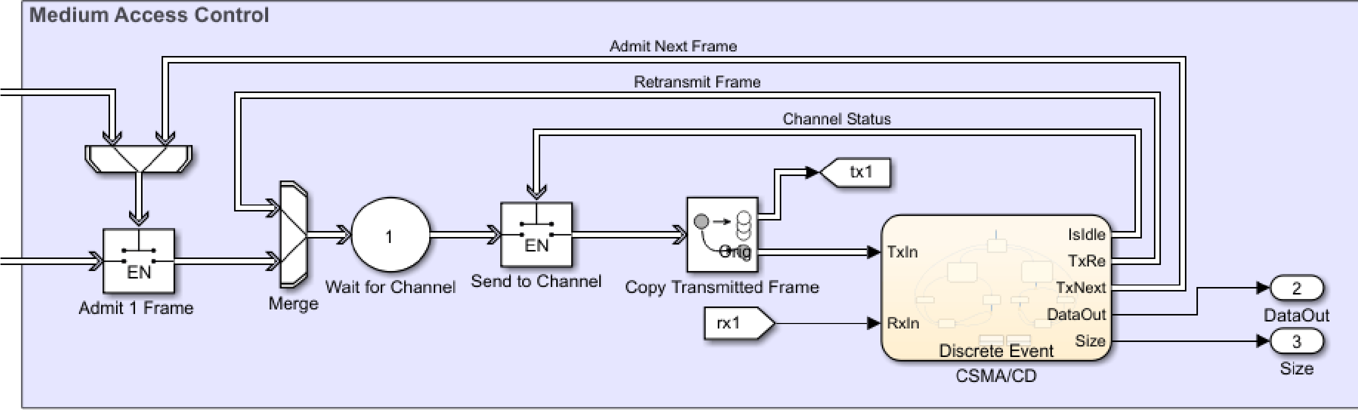
The capacity of the queue is determined by the **Transmission buffer capacity** parameter.

#### Medium Access Control

The Medium Access Control blocks are modeled by using six SimEvents® blocks.

- An Entity Gate block labeled Admit 1 Frame is configured as an enabled gate with two input ports. One input port allows frames from the Transmission Buffer block. The other input port is called the control port, which accepts messages from the CSMA/CD block. The block allows one frame to advance when it receives a message with a positive value from CSMA/CD block.
- An Entity Input Switch block labeled Merge merges two paths. One input port accepts new frames admitted by the Admit 1 frame block and the other input port accepts frames for retransmission that are sent by the CSMA/CD block.
- An Entity Server block labeled Wait for Channel models the back off time of a frame before its retransmission through the channel.

- Another Entity Gate block labeled Send to Channel opens the gate to accept frames when the channel is idle. The channel status is communicated by the CSMA/CD chart.
- An Entity Replicator block labeled Copy Transmitted Frame generates a copy of the frame. One frame is forwarded to the Ethernet network, and the other is forwarded to the CSMA/CD chart.
- A Discrete-Event Chart block labeled CSMA/CD represents the state machine that models the CSMA/CD protocol.



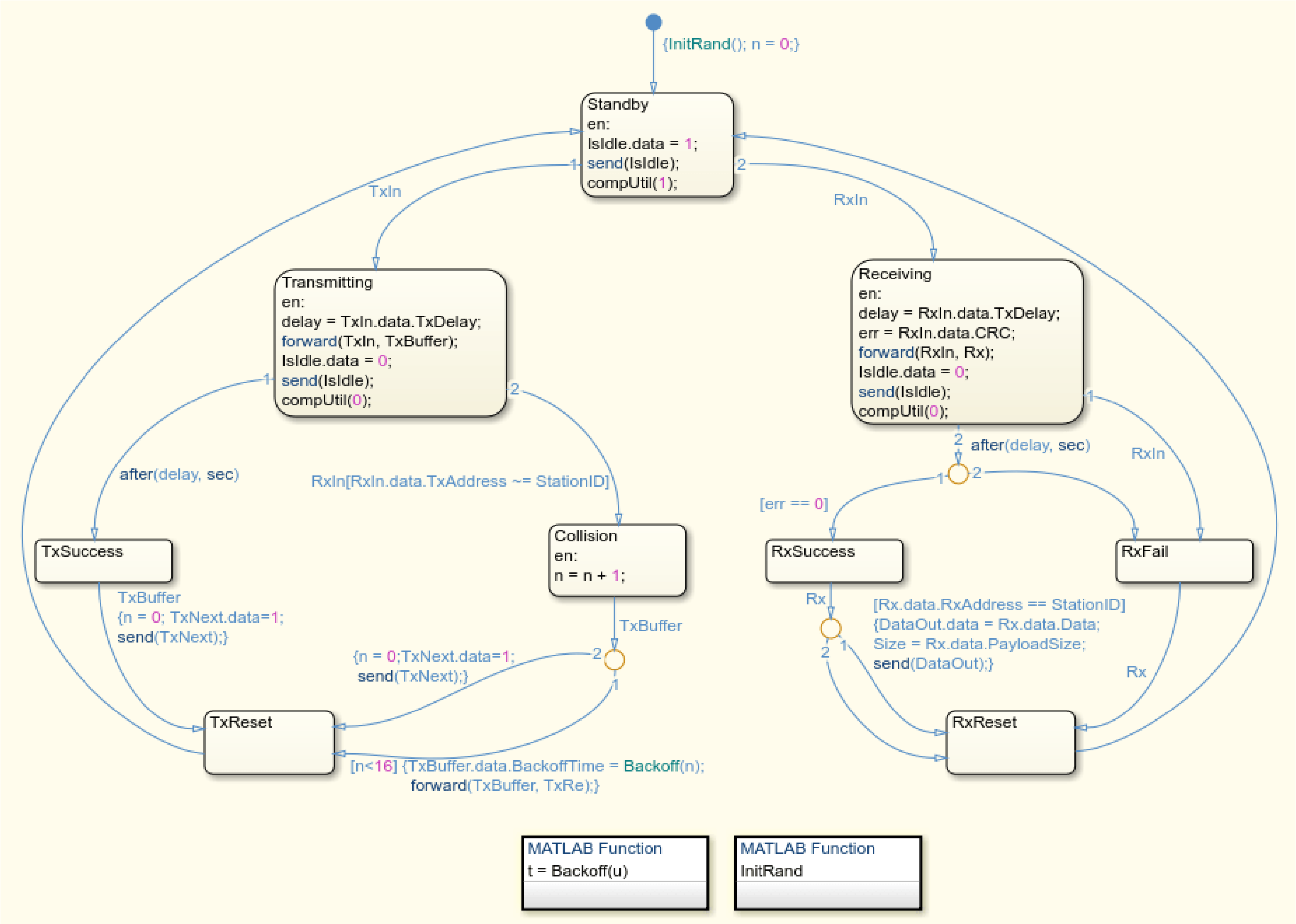

#### CSMA/CD Protocol

The CSMA/CD protocol [ 2 ] is modeled by a Discrete-Event Chart block that has two inputs:

- `TxIn` — Copy of the transmitted frame.
- `RxIn` — Received frame from the Ethernet network.

The chart has five outputs:

- `IsIdle` — Opens the Send to Channel gate to accept frames when the value is `1`, and closes the gate when the value is `0`.
- `TxRe` — Retransmitted frame that is forwarded to the Merge block if there is a collision detected during its transmission.
- `TxNext` — Opens the Admit 1 Frame gate to accept new frames when the value is `1`.
- `DataOut` — Received data.
- `Size` — Size of the received data.




### Transmitting and Receiving Messages

The block is initially in the `Standby` state and the channel is idle.

If the block is transmitting, after a delay, the block attempts to transmit the message and `Isle.data` is set to `0` to declare that the channel is in use.

If the transmission is successful, the block sets `TxNext.data` to `1` to allow a new message into the channel and resets to the `Standby` state.

If there is a collision, the block resends the message after delaying it for a random back off time. *n* is the counter for retransmissions. The block retransmits a message a maximum of `16` times. If all of the retransmission attempts are unsuccessful, then the block terminates the message and allows the entry of a new message. Then it resets to `StandBy`.

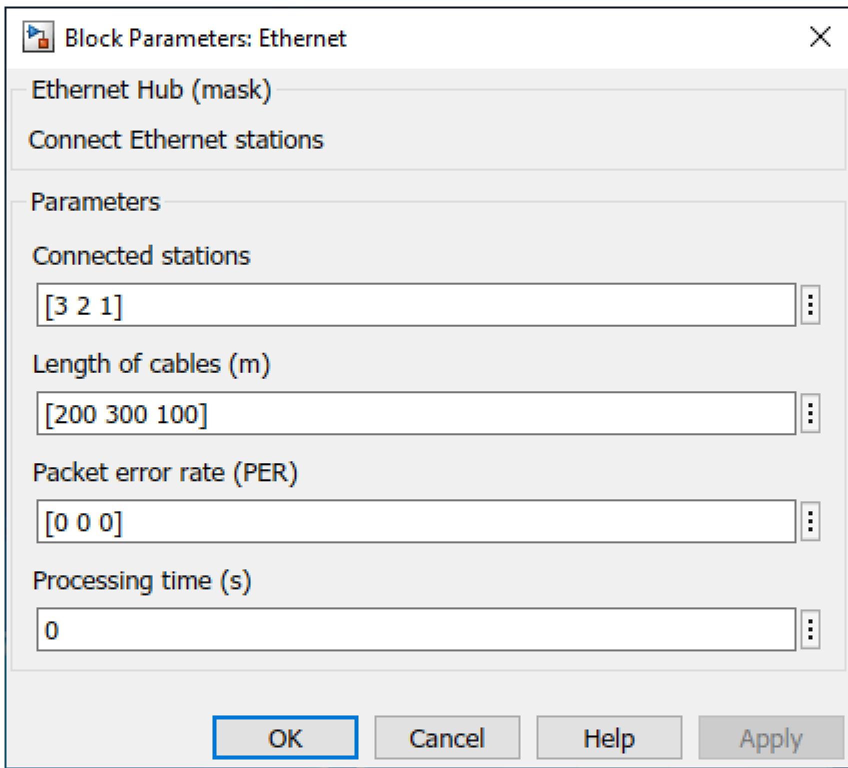
Similarly, the block can receive messages from other computers. If there is no error, the messages are successfully received and the block outputs the received data and its size.

### Ethernet Hub

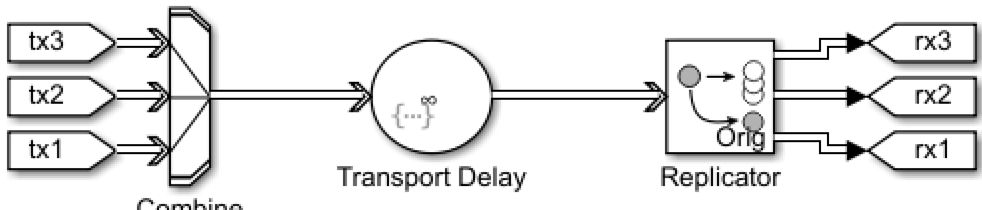
The Ethernet component represents the communication network and the cabled connections of the computers to the network.

Double-click the Ethernet block to see its parameters.

- **Connected stations** — These values are assigned to `Stations`, which is a vector with the station IDs as elements.
- **Length of cables (m)** — These values are assigned to `CableLength` and represent the length of the cables, in meters, for each computer connected to the hub.
- **Packet error rate (PER)** — These values are assigned to `PER` and represent the rate of error in message transmission for each computer.
- **Processing time (s)** — These values are assigned to `ProcessingTime` and it represents the channel transmission delay.

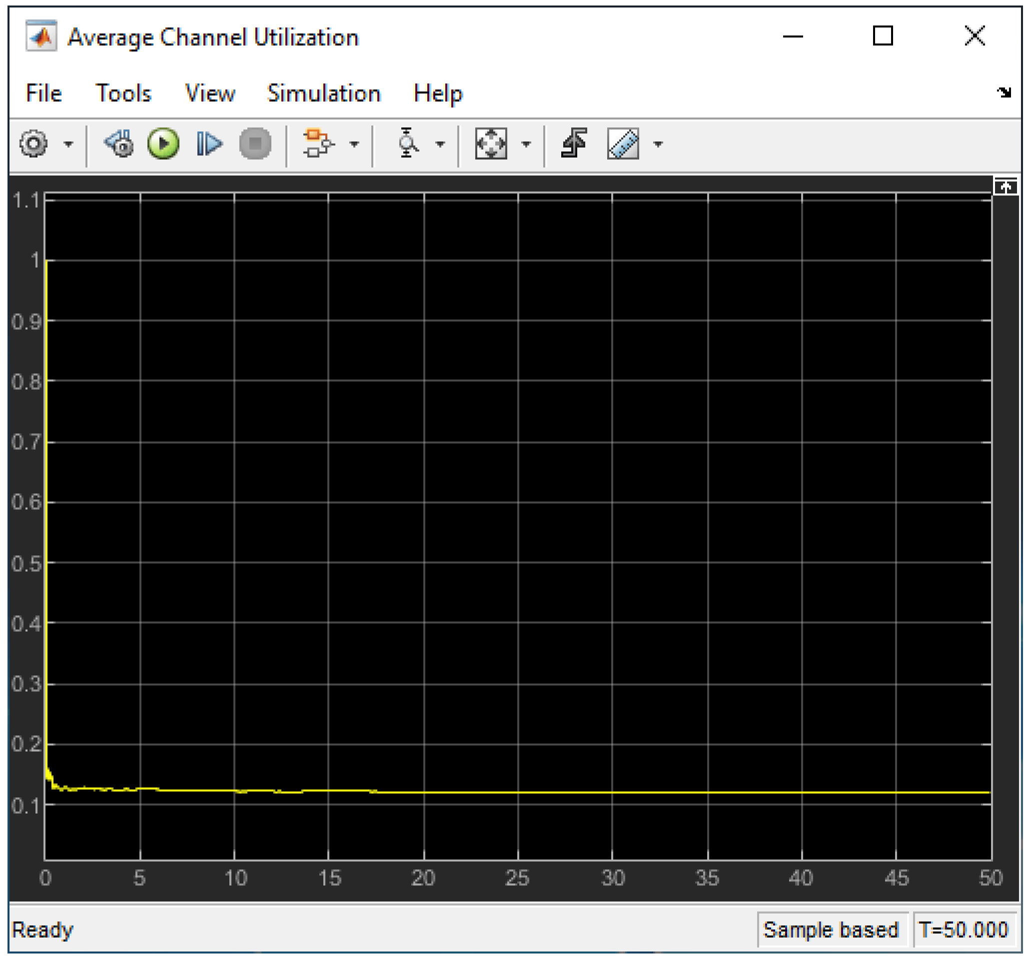



Three SimEvents® blocks are used to model the Ethernet network. The three computer connections are merged by using an Entity Input Switch block. An Entity Server block is used to model the channel transmission delay based on the cable length. An Entity Replicator block copies the transmitted message and forwards it to the three computers.

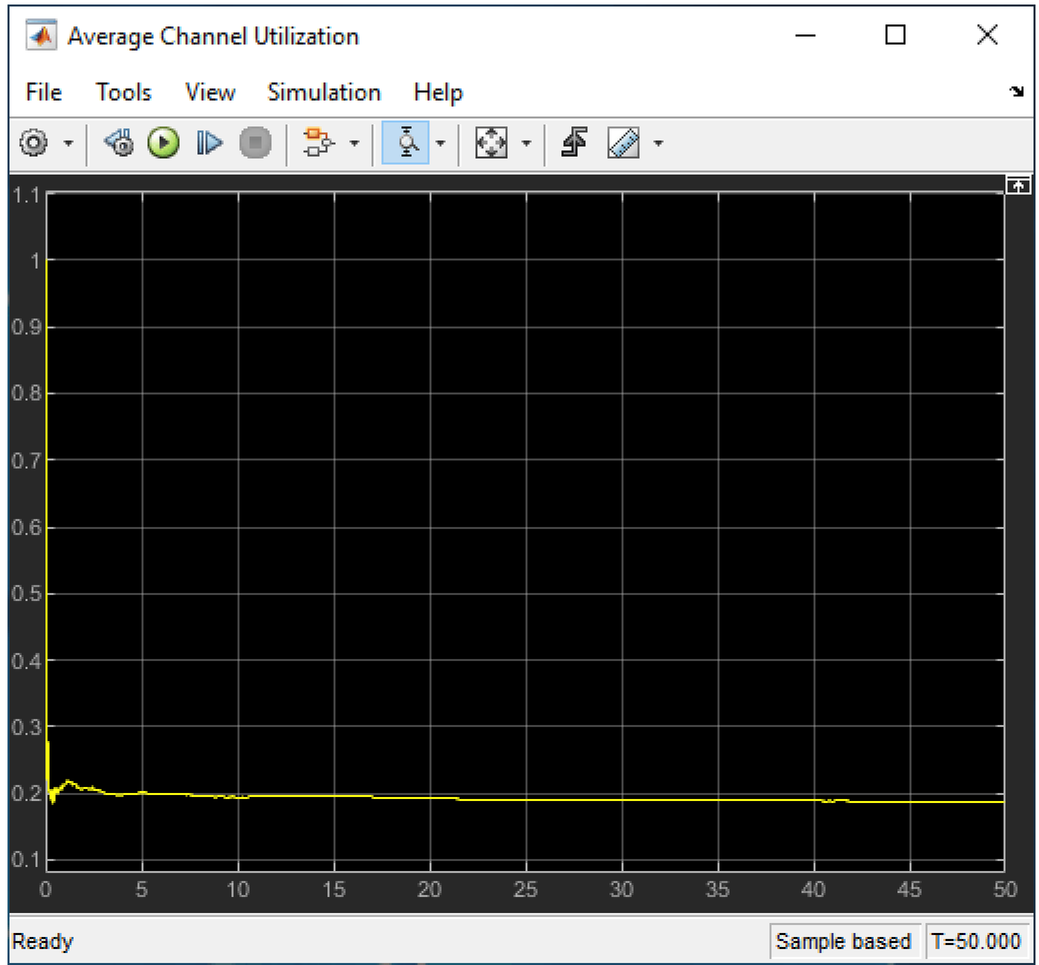


### Simulate the Model and Review the Results

Simulate the model and open the Scope block that displays the average channel utilization. The channel utilization converges to approximately `0.12`.




Open Software Component 1 as a top model and change the data generation rate by setting the **Sample time** of the Generate Data 1 block to `0.01`. Run the simulation again and observe that the channel utilization increases to `0.2`.

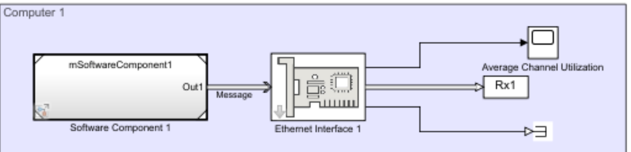


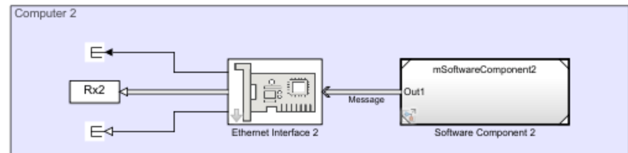

### Connect New Computers to the Network

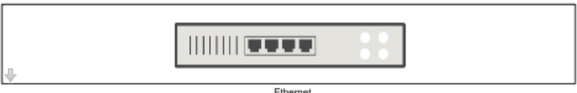
You can connect more computers to the network.

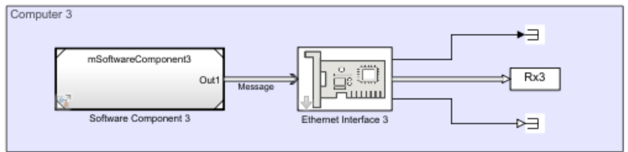
To add a new computer to the network:

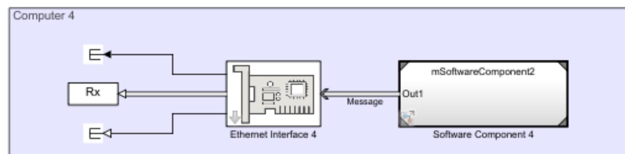
- Copy an existing computer and assign a new ID by double-clicking the Ethernet Interface block. In this example, new computer has ID 4.



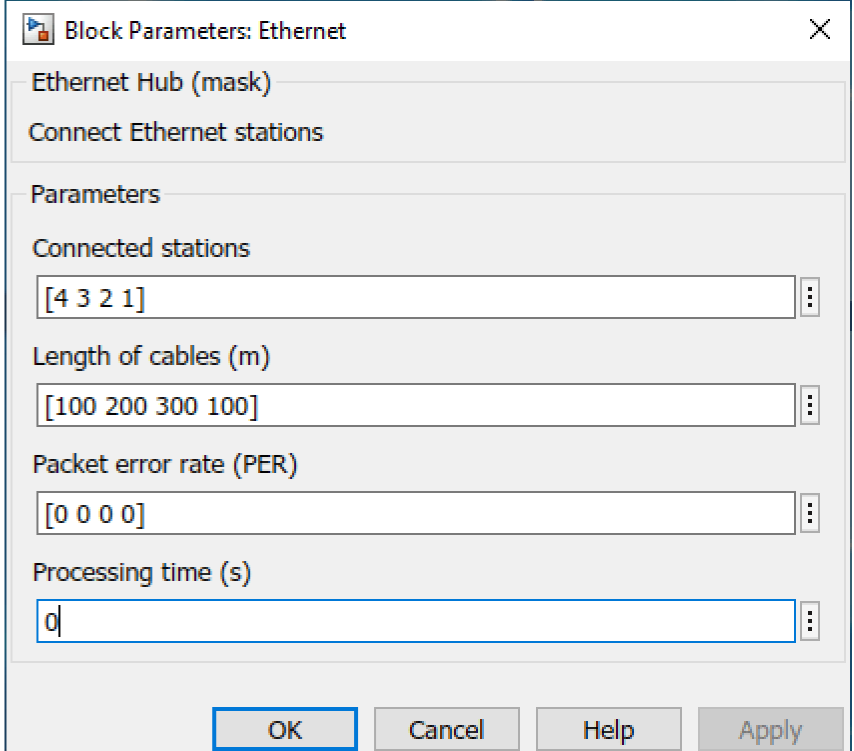




Copyright 2019 The MathWorks, Inc.

- Double-click the Ethernet block and add a station ID, cable length, and packet error rate for the new computer.




## References

**1** Ethernet frame - Wikipedia (https://en.wikipedia.org/wiki/Ethernet_frame)

2 Carrier-sense multiple access with collision detection - Wikipedia (https://en.wikipedia.org/wiki/Carrier-sense_multiple_access_with_collision_detection)

# Work with Resources


- “Model Using Resources” on page 4-2
- “Set Resource Amount with Attributes” on page 4-4
- “Group Entities Using Batching” on page 4-6
- “Find and Extract Entities in SimEvents Models” on page 4-10

# Model Using Resources

| In this section... |
| --- |
| "Resource Blocks" on page 4-2 |
| "Resource Creation Workflow" on page 4-2 |

## Resource Blocks

Resources are commodities shared by entities in your model. They are independent of entities and attributes, and can exist in the model even if no entity exists or uses them. Resources are different from attributes, which are associated with entities and exist or disappear with their entity.

For example, if you are modeling a restaurant, you can create tables and food as resources for customer entities. Entities can access resources from types of resources.

The SimEvents software supplies the following resource allocation blocks:

| Action | Block |
| --- | --- |
| Acquire resource | Resource Acquirer |
| Define resource | Resource Pool |
| Release resource | Resource Releaser |

## Resource Creation Workflow

1. Specify resources using the Resource Pool block. Define one resource per Resource Pool block. Multiple Resource Pool blocks can exist in the model with multiple entities sharing the resources.
2. Identify resources to be used with the Resource Acquirer block. You can identify these resources before specifying them in a Resource Pool block, or select them from the available resources list. However, the resource definitions must exist by the time you simulate the model. Multiple Resource Acquire blocks can exist in the model.
3. To release resources, include one or more Resource Releaser blocks. You can configure Resource Release blocks to release some or all resources for an entity. Alternatively, you can release all resources for an entity directly using the Entity Terminator block.

---

**Tip** To determine how long an entity holds a resource, insert a server block after the Resource Acquire block. In the **Service time** parameter, enter how long you want the entity to hold the resource.

---

An entity implicitly releases held resources when it:

- Is destroyed.
- Enters an Entity Replicator block and the block creates multiple copies of that entity.
- Is combined with other entities using the Composite Entity Creator block.
- Is split into its component entities using the Composite Entity Splitter block.

## See Also

Resource Acquirer | Resource Pool | Resource Releaser

# Set Resource Amount with Attributes

Use the **Selected Resources** table of the Resource Acquirer block to receive the resource amount definition from the block dialog box or an entity attribute. Using attributes as the source for the resource requires synchronicity between these blocks:

- Entity Generator block with the attribute definition that Resource Acquirer wants to supply the source amount
- Resource Pool block that defines the resource
- Resource Acquirer block the acquires the resource

This example shows this synchronicity.

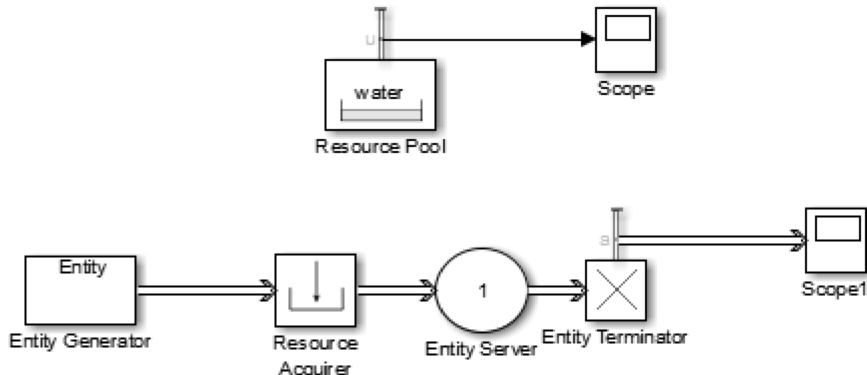
1. Open a new model and add Resource Pool, Entity Generator, and Resource Acquirer blocks. For the Resource Pool block:

   - Set **Resource name** to `water`.
   - Set **Resource amount** to `20`.
   - In the **Statistics** tab, select **Amount in use, #u**.
2. In the Entity Generator block dialog box, click the **Entity type** tab and in the **Define attributes** table:

   - Enter the attribute name, `water_amount`, to indicate that the attribute defines the amount of the resource.
   - Set the value to `10`.
3. In the Resource Acquirer block dialog box, click the **Entity type** tab and under Available Resources, select `water` and move it to the **Selected Resources** table.
4. In the **Selected Resources** table, in the `water` entry:

   - For **Amount Source**, select `Attribute`.
   - For **Amount**, enter `water_amount` to match the attribute name defined in the Entity Generator block.
5. To complete the model, add the following blocks and connect them as shown in the figure:

   - Entity Terminator (select the **Statistics** tab **Number of entities arrived, #a** check box)
   - Two Scope blocks



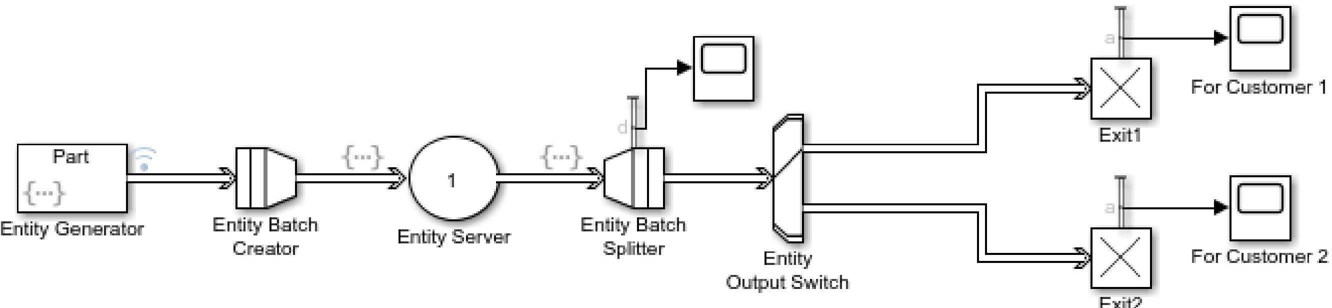
6. Simulate the model and observe the amount of resources in use (Scope).

## See Also

Resource Acquirer | Resource Pool | Resource Releaser

# Group Entities Using Batching

This example shows how to create, process, and split batched entities using Entity Batch Creator and Entity Batch Splitter blocks. In the model, An Entity Generator block is used to represent produced parts in a facility. The parts are batched by an Entity Batch Creator block. A batch is processed by an Entity Server block. When the processing is complete, the batch is split into individual parts by the Entity Batch Splitter block for their delivery.




In the model:

- Use an Entity Generator block to generate a `Part` with two attributes, `Color` and `Customer`, representing color and delivery destination. To generate three different colors and two different delivery destinations for each `Part`, in the **Event actions** tab, in the **Generate action** field enter this code. field:

```
entity.Color = randi([1 3]);
entity.Customer = randi([1 2]);
```

- Use an Entity Batch Creator block to generate a batch that contains four parts.

- Use an Entity Server block to process and change the color of the third `Part` in each batch. In the **Event actions** tab, in the **Entry** field enter this code.

```
entity.batch(3).Color = 5;
```

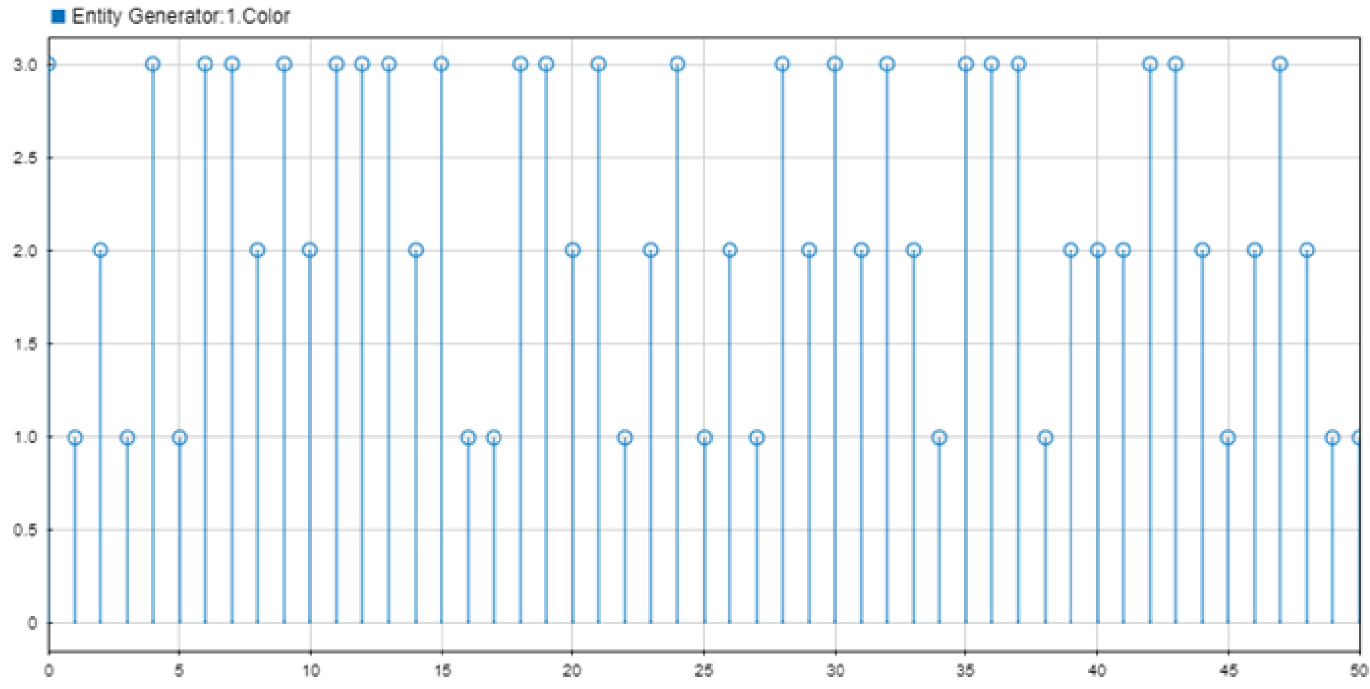
- Use an Entity Batch Splitter block to split parts. In the **Entry action**, use `disp(entity.batch(3).Color)` to display the color of the third `Part` in each processed batch.

- Use an Entity Output Switch block to route a `Part` to the corresponding customer based on its `Customer` attribute.

### Simulate Model and Review Results

Simulate the model.

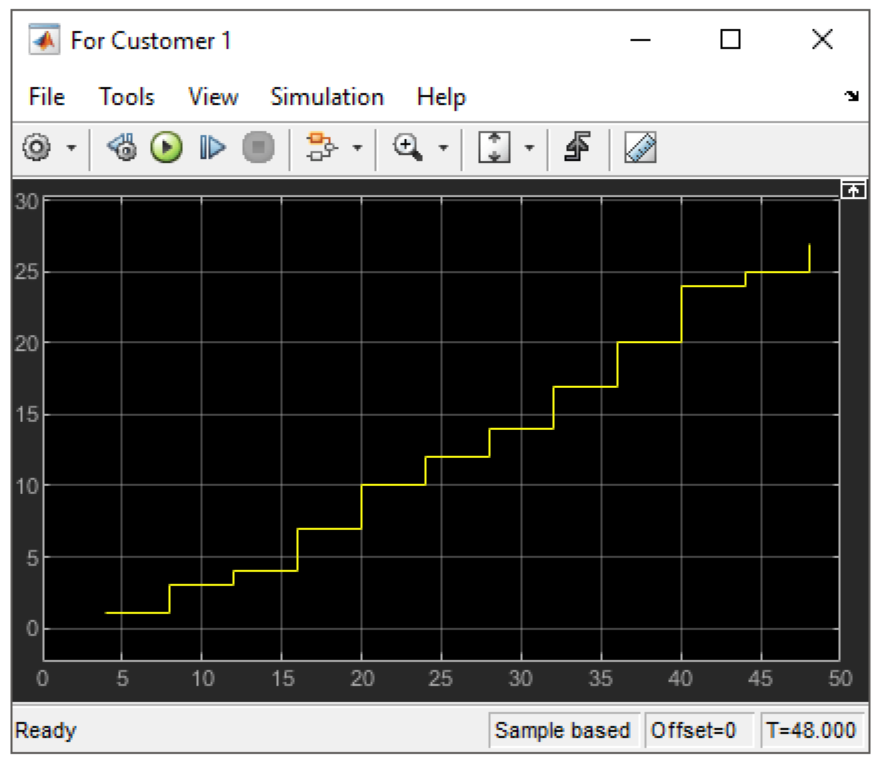
Open the Simulation Data Inspector and observe that the parts are generated with `Color` values `1`, `2`, or `3`.

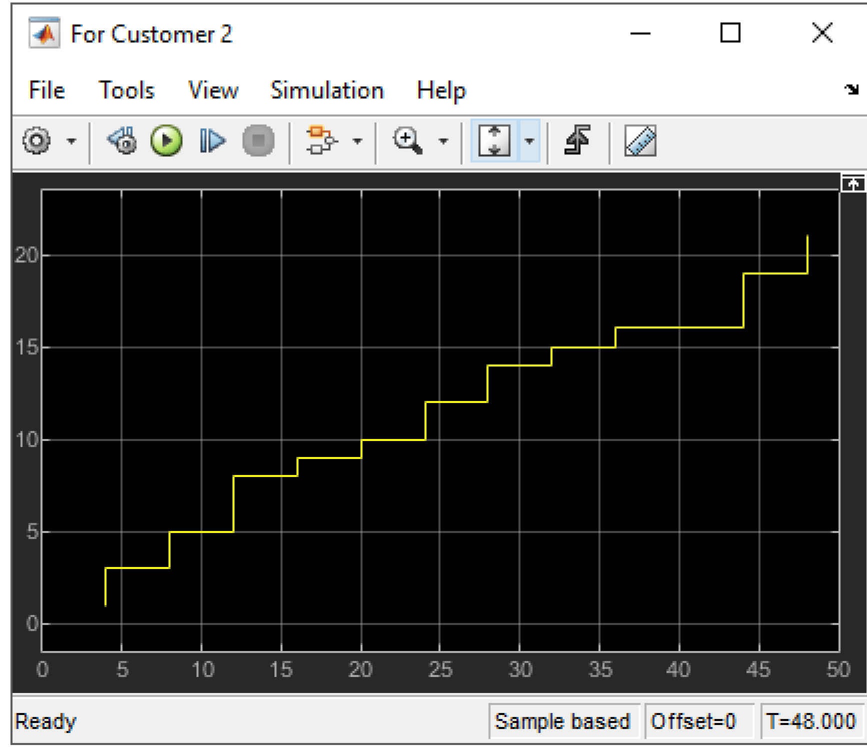



- Observe that the Diagnostic Viewer displays `Color` values of the third entity in each batch after batch processing.

```
5
5
5
5
5
5
5
5
5
5
5
5
```

- Scope blocks labeled as For Customer 1 and For Customer 2 display the number of parts delivered to each customer.




## See Also

Entity Batch Creator | Entity Batch Splitter | Entity Generator | Entity Output Switch | Entity Server

## More About


- “Model Using Resources” on page 4-2
- “Optimize SimEvents Models by Running Multiple Simulations” on page 5-20
- “Find and Extract Entities in SimEvents Models” on page 4-10
- “Resource Scheduling Using MATLAB Discrete-Event System and Data Store Memory Blocks” on page 9-58

# Find and Extract Entities in SimEvents Models

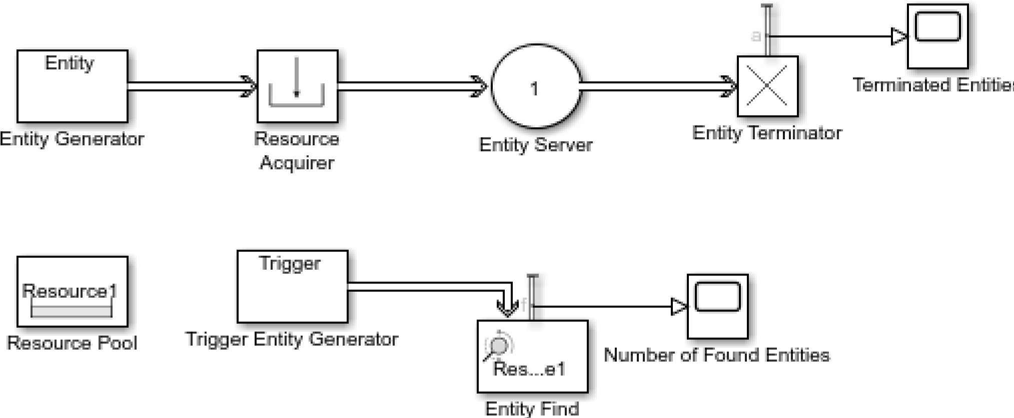
You can find entities in a SimEvents model by using an Entity Find block. The block searches and finds entities that use a particular resource from a Resource Pool block and acquire it through a Resource Acquirer block.

You can use the Entity Find block for these applications.

- Model a supply chain to monitor perishable items and update the inventory records. For instance, you can modify the price of an item when it is closer to its expiration date.
- Model timers and perform actions on products based on timers.
- Model recall of products from a supply chain. You can reroute recalled products back to the supply chain after repair.

## Finding and Examining Entities

The Entity Find block helps you find and examine entities at their location. In this example, the block finds entities that are tagged with a `Resource1` resource from the Resource Pool block. Then, an additional filtering condition helps to further filter the found entities.




1. Add an Entity Generator block, Resource Pool block, Resource Acquirer block, Entity Server block, and Entity Terminator block.

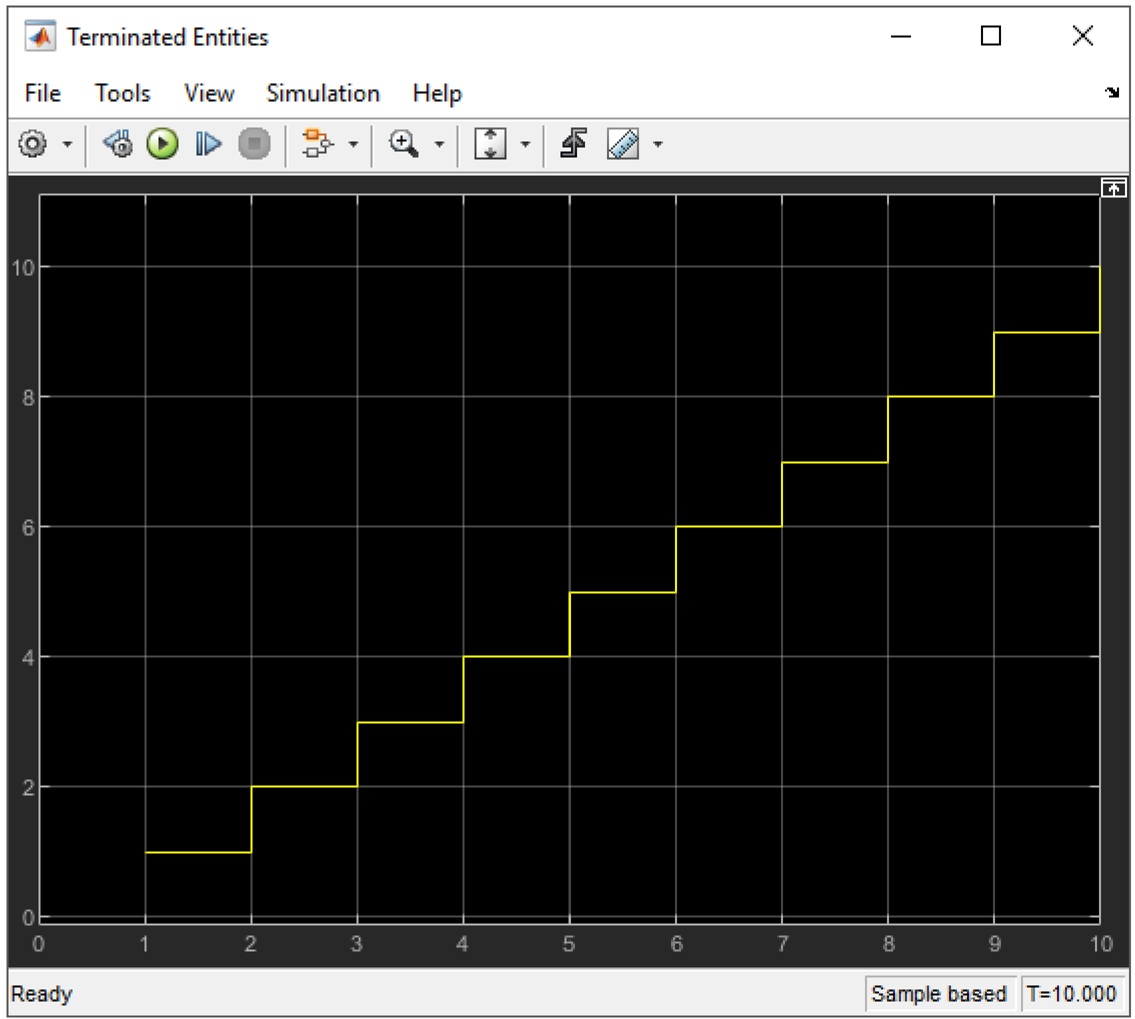
   The top model represents the flow of entities that acquires a `Resource1` resource.
2. In the Entity Terminator block, output the **Number of entities arrived, a** statistic and connect to a scope.
3. Add an Entity Find block. Output the **Number of entities found, f** statistic and connect it to a scope.

   By default, the block finds entities with the `Resource1` tag.
4. Add another Entity Generator block and label it Trigger Entity Generator. Connect it to the input port of the Entity Find block. In the block, change the **Entity type name** to `Trigger` and **Entity priority** to `100`.

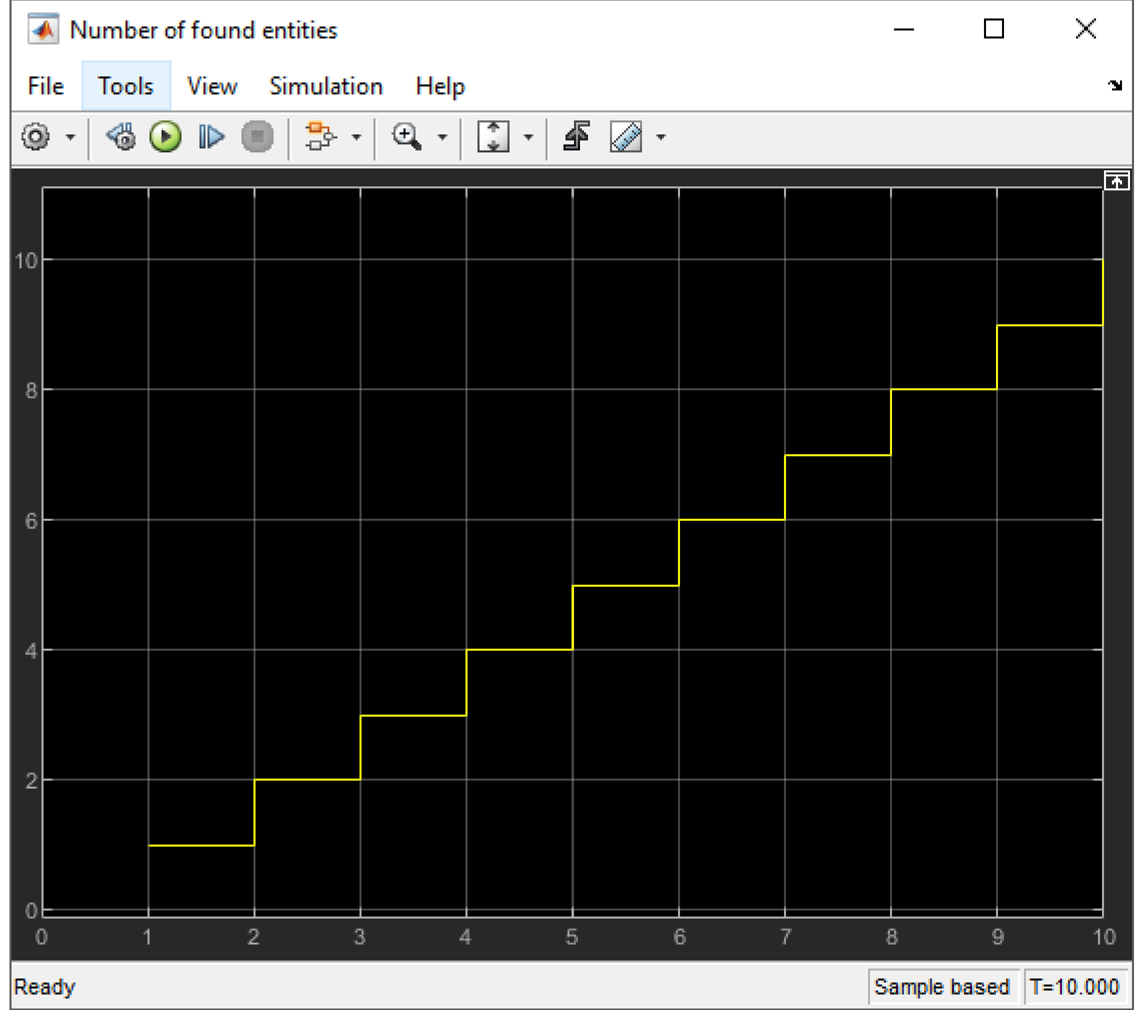
   Every time the Trigger Entity Generator generates a trigger entity, the Entity Find block is triggered to find entities.

**Note** The entities in the model have priority `300` and the priority of the trigger entity is set to `100` to make trigger entities higher priority in the event calendar. This prevents the termination of the entities before they are found by the Entity Find block.

5. Simulate the model and observe that the number of terminated entities is `10`, which is equal to the number of found entities by the Entity Find block. Every generated entity acquires a `Resource1` tag and there is no blocking of entities in the model.




The Entity Find block finds entities with the `Resource1` resource for every generated trigger entity.




6 In the Entity Generator Block Parameters dialog box, in the **Generate action** field, add this code.

```
entity.Attribute1 = randi([1,2]);
```

The entities are generated with a random `Attribute1` value `1` or `2`.

7 In the Entity Find Block Parameters dialog box, select the **Additional filtering condition** check box. Add this code to replace any existing code and to set the filtering condition.

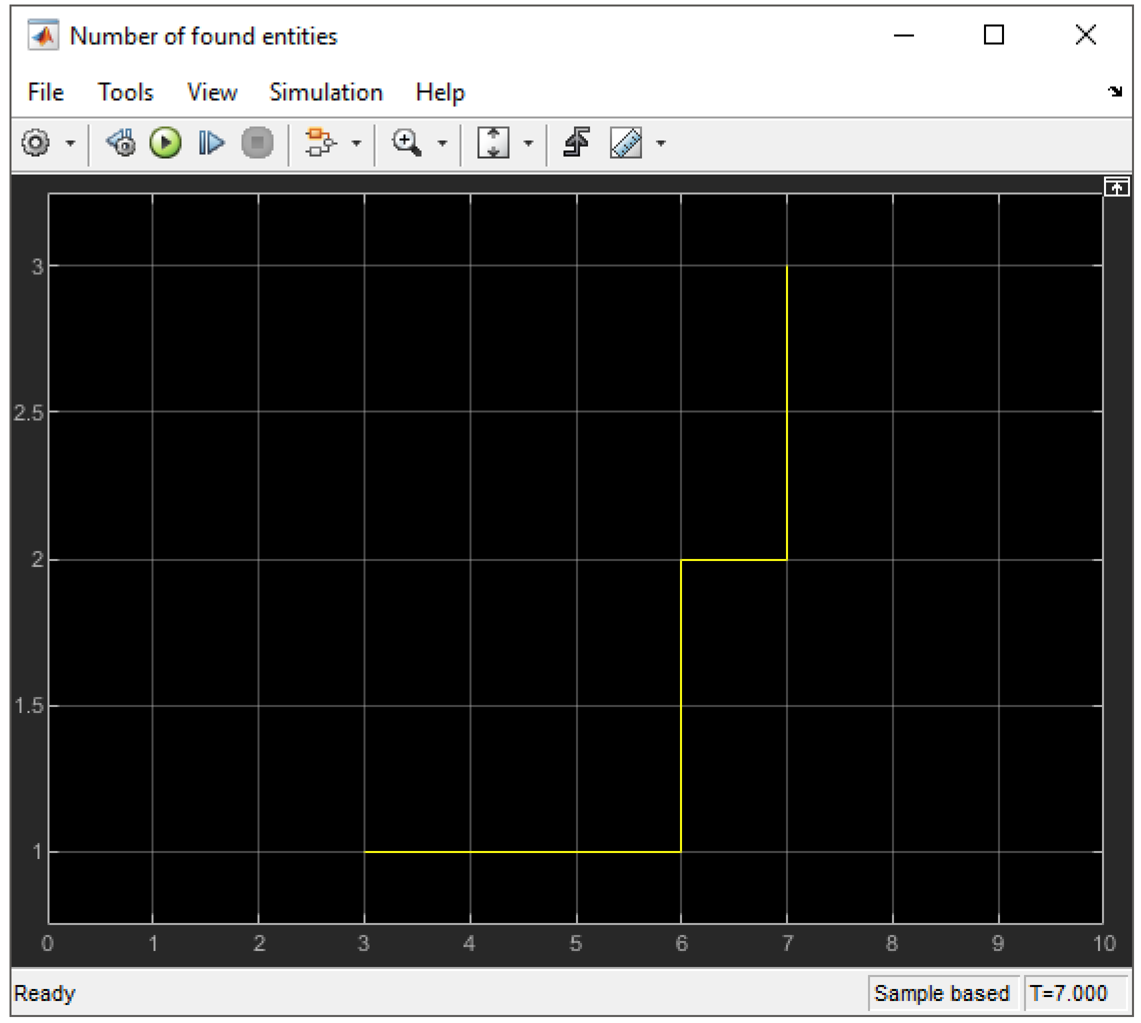
```
match = isequal(trigger.Attribute1, entity.Attribute1);
```

The block finds the entities that acquire the `Resource1` tag when the `match` is `true`. That is, the `Attribute1` value of an entity is equal to the trigger entity `Attribute1` value.

8 In the Trigger Entity Generator, observe that the `Attribute1` value is `1`.

9 Simulate the model, observe that the number of found entities decreased to `3` because entities with the `Attribute1` value `2` are filtered out by the additional matching condition.

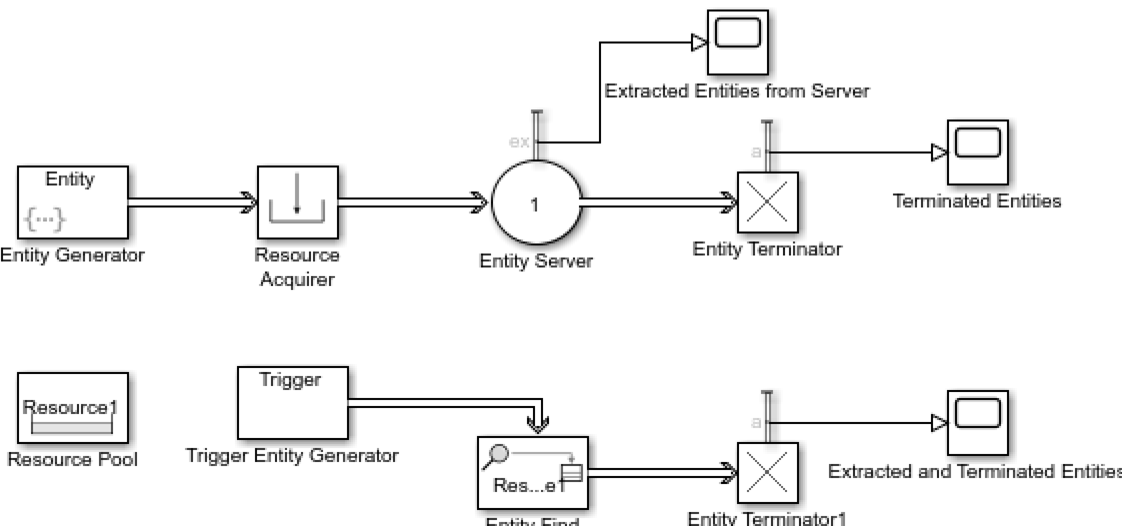
The trigger entity `Attribute1` value is `1`. The block finds entities that acquire `Resource1` tag and have the `Attribute1` value `1`.




## Extracting Found Entities

You can use the Entity Find block to find entities and extract them from their location to reroute. In this example, 3 entities found in the previous example are extracted from the system to be terminated.

To open the model, see `Extract Found Entities Example`.

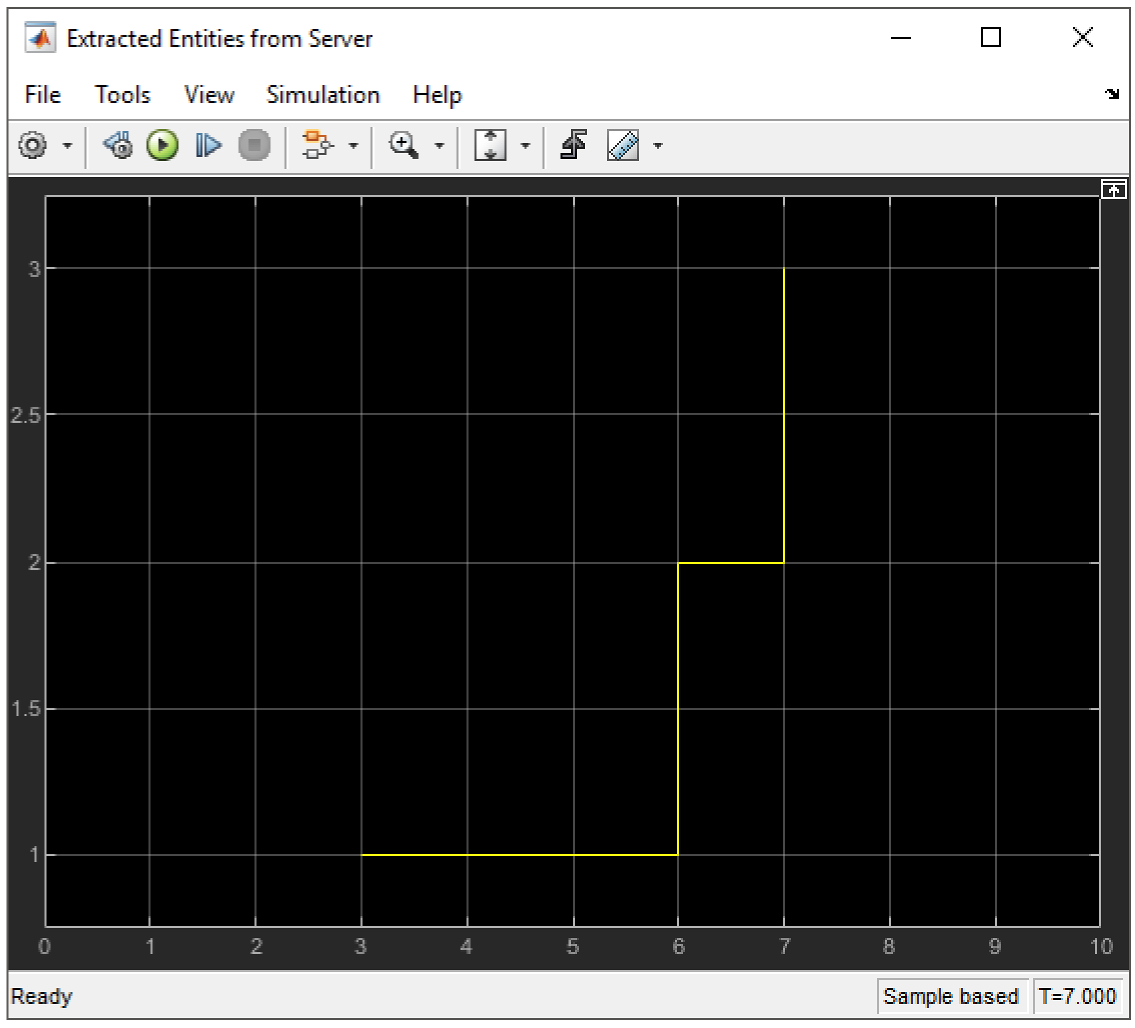



1. In the Entity Find Block Parameters dialog box, select the **Extract found entities** check box.

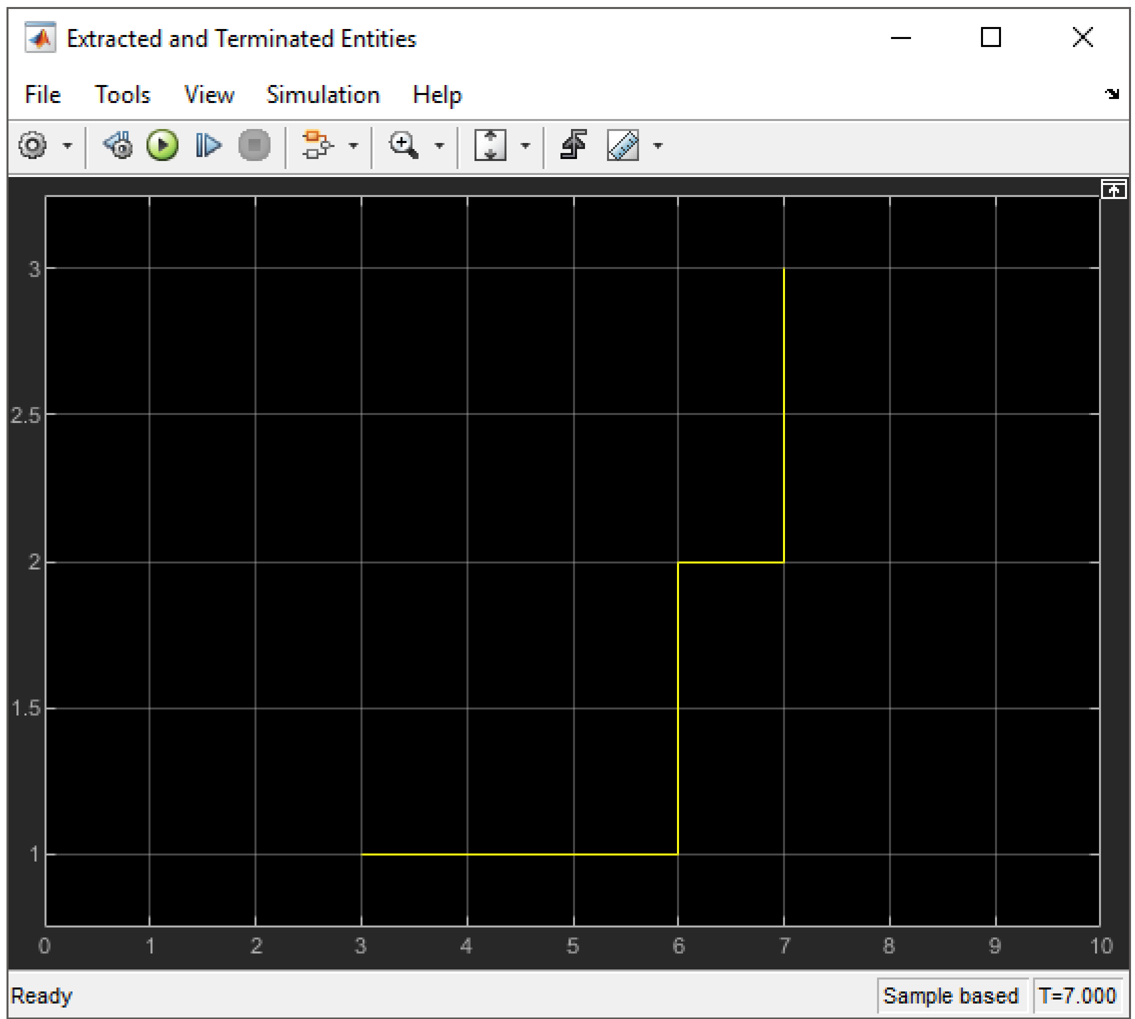
   Observe that a new output port appears at the Entity Find block for the extracted entities.
2. Connect the output of the Entity Find block to a new Entity Terminator1 block.
3. Output the **Number of entities extracted, ex** statistic from the Entity Server block and connect it to a scope.

   Visualize the number of extracted entities from the server.
4. Output the **Number of entities arrived, a** statistic from the Entity Terminator1 block and connect it to a scope.

   The statistic is used to observe the number of found and extracted entities from the system.
5. Simulate the model. Observe that the **Number of entities extracted, ex** is 3.




6. Observe that 3 found entities are extracted from the Entity Server block and terminated in the Entity Terminator1 block.




As a result, 7 entities arrive at the Entity Terminator block in the model.

## Changing Found Entity Attributes

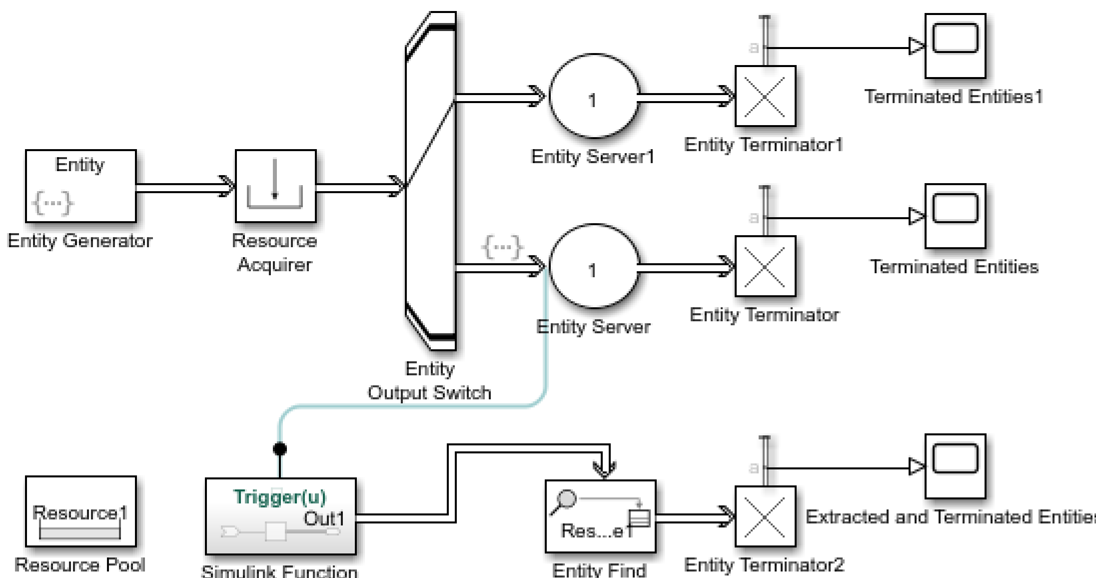
You can change the attributes of the found entities at their location or with extraction.

1. Change the attributes of found entities at their location by entering MATLAB code in the `OnFound action` field of the **OnFound** event action. For more information about events and event actions, see "Events and Event Actions" on page 1-2.
2. Change the attributes of found and extracted entities when they enter, exit, or are blocked by the Entity Find block. Enter MATLAB code in the `Entry action`, `Exit action`, and `Blocked action`, field of the **Event actions** tab.

## Triggering Entity Find Block with Event Actions

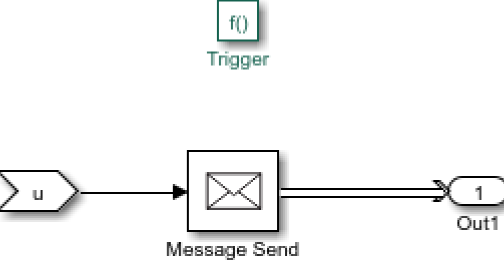
You can trigger the Entity Find block with event actions. In this example, the Entity Find block is triggered when an entity enters the Entity Server block. Modify the previous example by removing the Trigger Entity Generator and by adding the Entity Output Switch, Entity Server1, Entity Terminator2 and Scope blocks to the model and connect them as shown.

To open the model, see `Trigger Entity Find Example`.




1. In the Entity Output Switch block, set the **Switching criterion** to `Equiprobable`.

   Entities flow through the Entity Server and Entity Server1 blocks with equal probability.
2. Replace the Trigger Entity Generator block by a Simulink Function block to trigger Entity Find block. On the Simulink Function block, double-click the function signature and enter `Trigger(u)`.
3. In the Simulink Function block, add the Message Send block and connect it to an Out1 block.

   The `Trigger(u)` function call generates a message to trigger the Entity Find block every time an entity enters the Entity Server1 block.
4. In the Entity Server block, in the **Entry action** field, enter this code.

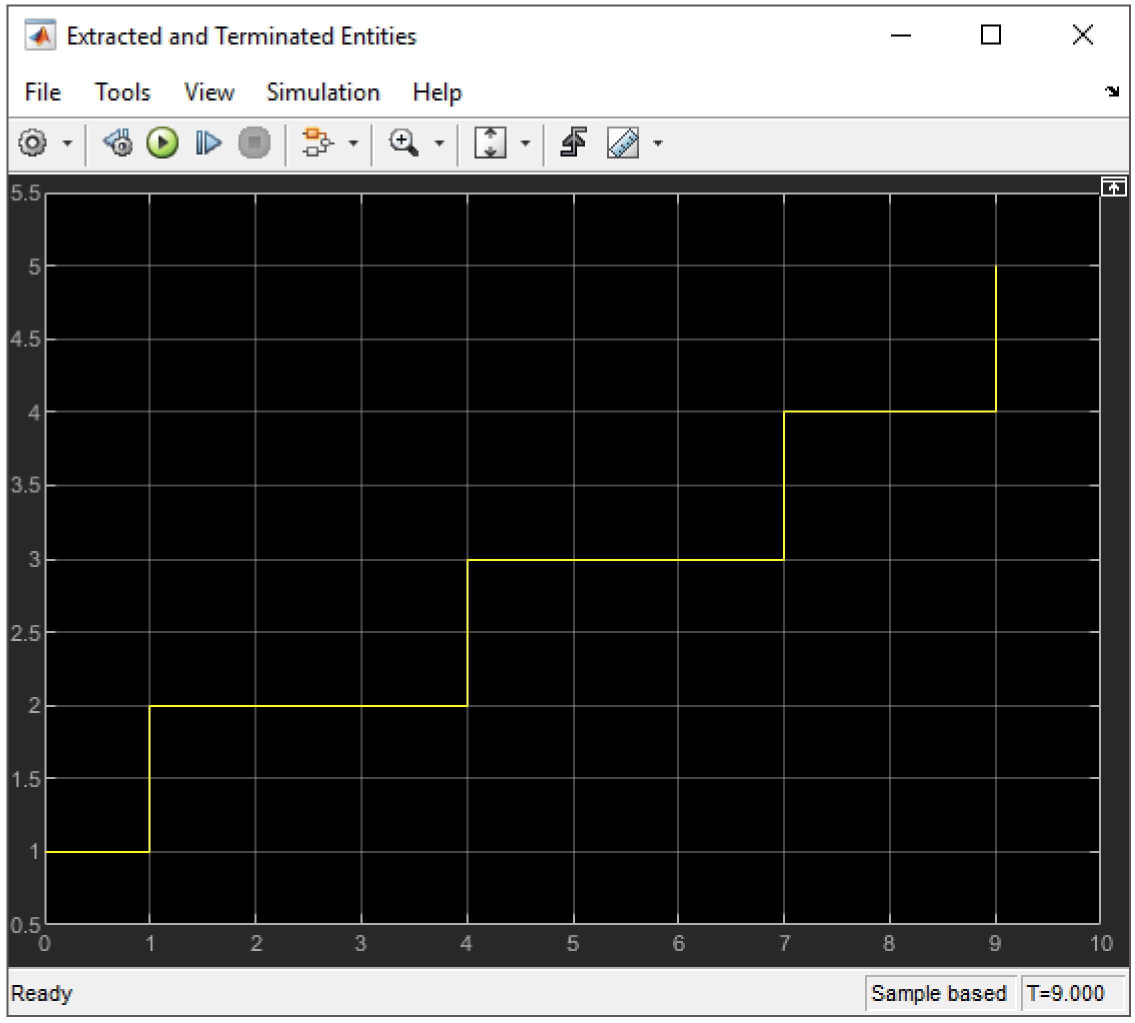
   ```
   Trigger(double(1));
   ```

   Every entity entry calls the `Trigger(u)` function in the Simulink Function block that triggers the Entity Find block.
5. In the Entity Find block, select the **Additional filtering condition** check box. Enter this code.

   ```
   match = isequal(2, entity.Attribute1);
   ```

   Found entities have the `Attribute1` value 2.

6 Simulate the model. Observe the scope that displays the extracted and terminated entities when the Entity Find block is triggered by the entity entry to the Entity Server block.

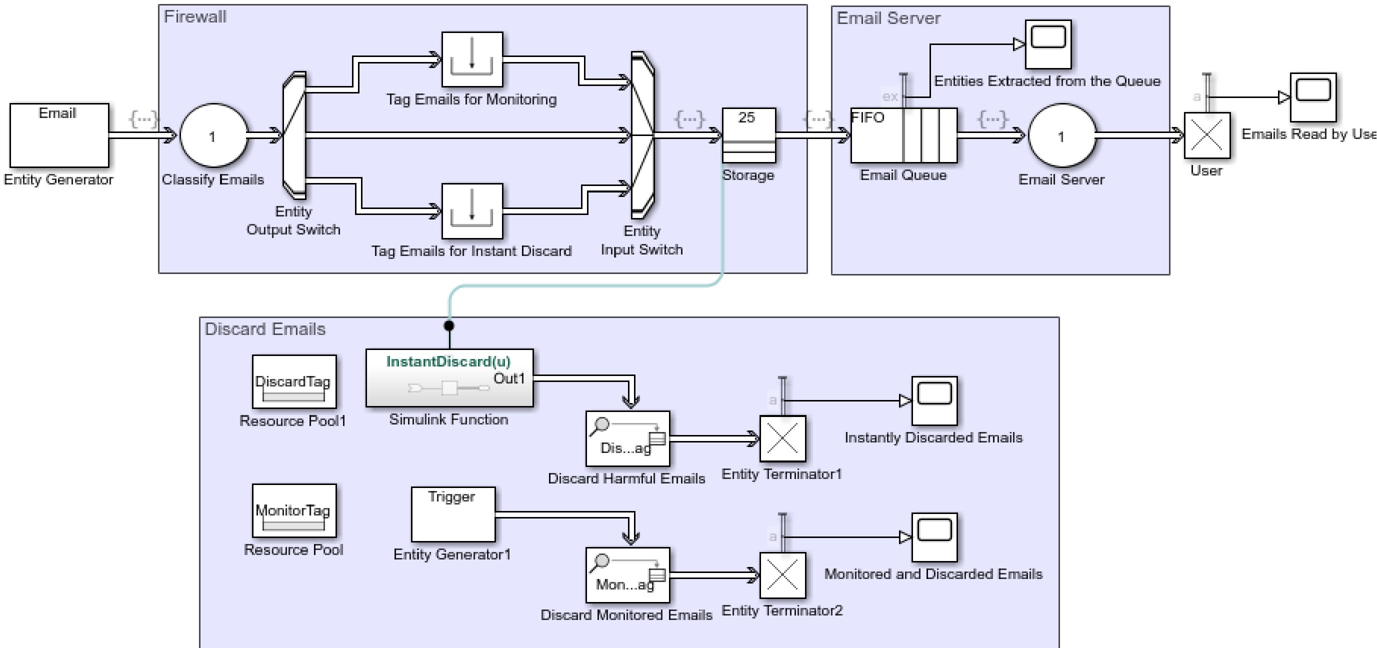



## Building a Firewall and an Email Server

You can use the Entity Find block to monitor multiple blocks in a model to examine or extract entities and modify entity attributes.

This example represents an email server with a firewall to track, monitor, and discard harmful emails before they reach the user. In the model, emails arrive from the Internet through an Entity Generator block. In the Firewall component, emails are classified as harmful for instant discarding, suspicious for monitoring, or safe based on their source. Harmful emails are tagged with a `DiscardTag` resource from the Resource Pool block and instantly discarded from the system. Suspicious emails are tagged with `MonitorTag` and tracked throughout the system for suspicious activity. If a suspicious activity is detected, the email is discarded before it reaches the user. Safe emails are not monitored or discarded.

To open the model, see `Email Monitoring Example`.




### Build Firewall and Email Server Components

1. Add an Entity Generator block. In the block, set the **Entity type name** to `Email` and attach two attributes as `Source` and `Suspicious` with initial value `0`.
2. Add an Entity Server block. In the block, select the **Event actions** tab, and in the **Entry action** field enter this code.

   ```
   entity.Source = randi([1,3]);
   ```

   The `Source` attribute value is randomly generated and it is `1` for a suspicious, `2` for a safe, and `3` for a harmful email source.
3. Add an Entity Output Switch block. In the block, set the **Number of output ports** to `3`, the **Switching criterion** to `From attribute`, and the **Switch attribute name** to `Source`.
4. Add two Resource Pool blocks and set their **Resource name** parameters to `MonitorTag` and `DiscardTag`.
5. Add a Resource Acquirer block labeled Tag Emails for Monitoring. In the block, select `MonitorTag` as **Selected Resources**.
6. Add another Resource Acquirer block labeled Tag Emails for Instant Discard. In the block, select `DiscardTag` as **Selected Resources**
7. Add an Entity Input Switch block. In the block, set the **Number of input ports** to `3`.
8. Add an Entity Store block. In the block, select the **Event actions** tab, and in the **Entry action** field enter this code.

   ```
   InstantDiscard(1);
   entity.Suspicious = randi([1,2]);
   ```
9. Add an Entity Queue block. In the block, select the **Event actions** tab, and in the **Entry action** field enter this code.

   ```
   entity.Suspicious = randi([1,2]);
   ```

The `Suspicious` attribute of an email changes in the entry. If the `Suspicious` attribute value is `2`, the email is extracted and terminated. This represents the randomly observed suspicious activity in the system.

10 Add another Entity Server block. In the block, set the **Service time value** to `3`, select the **Event actions** tab, in the **Entry action** field, enter this code.

```
entity.Suspicious = randi([1,2]);
```

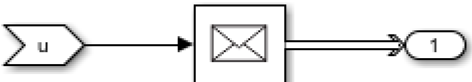
11 Add an Entity Terminator block labeled Emails Read by User, and connect all the blocks as shown in the model.

### Monitor and Discard Emails with Entity Find Block

1 Add a Simulink Function block.

   a Double-click the function signature on the Simulink Function block and enter `InstantDiscard(u)`.

   b Double-click the Simulink Function block. Add a Message Send block and an Out1 block.




2 In the parent model, add an Entity Find block. In the block, set **Resource** to `DiscardTag` and select **Extract found entities** check box.

   Any email entry calls the `InstantDiscard()` function and triggers the Entity Find block to find and discard harmful emails.

3 Add another Entity Terminator block labeled Instantly Discarded Emails.

4 Add another Entity Find block. In the block, set the **Resource** to `MonitorTag` and select the **Extract found entities** and the **Additional filtering condition** check boxes. In the **Matching condition** field, enter this code.

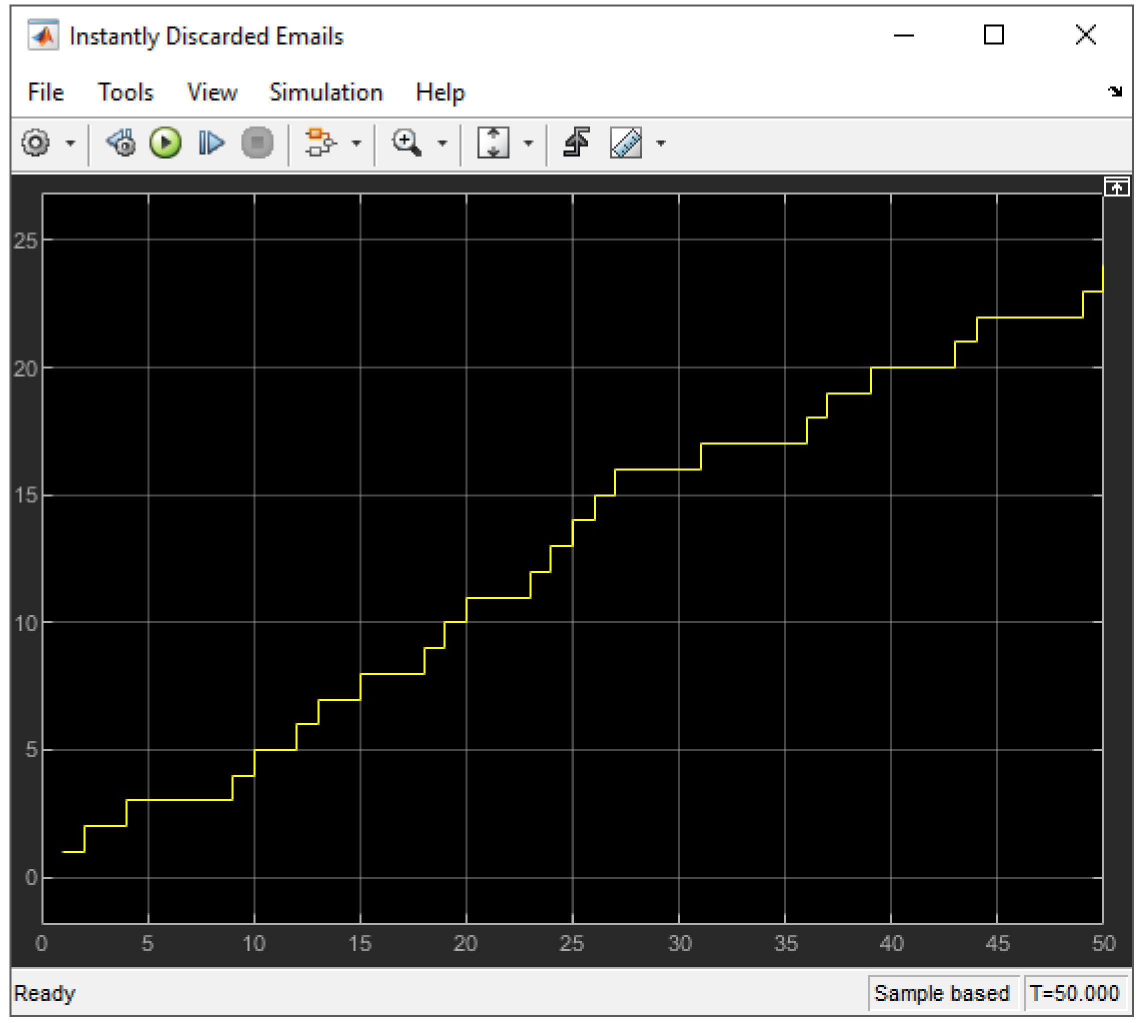
```
match = isequal(trigger.Attribute1, entity.Suspicious);
```

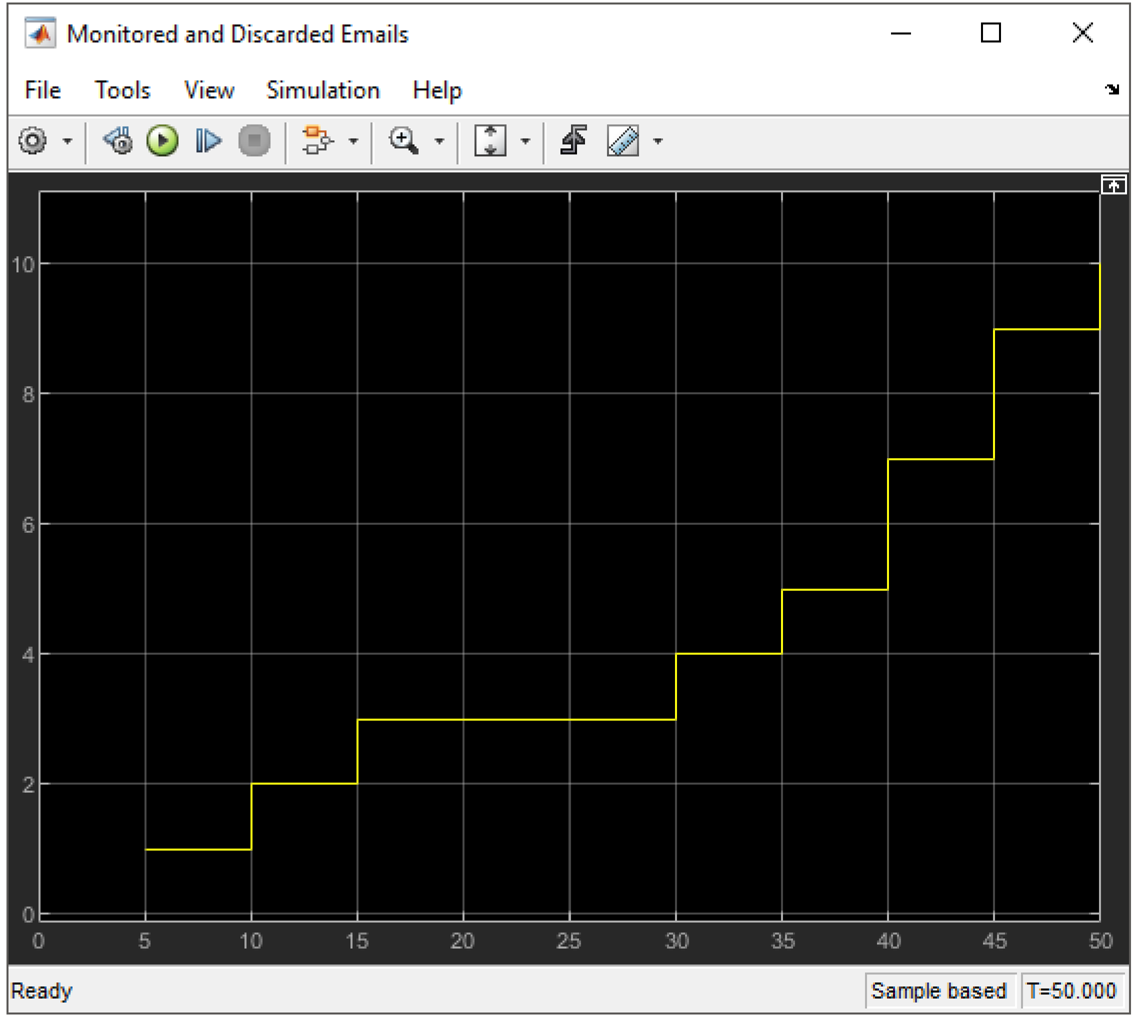
5 Add another Entity Generator block labeled Entity Generator1. In the block, set the **Period** to `5`, the **Entity priority** to `100`, the **Entity type name** to `Trigger`, and the **Attribute Initial Value** to `2`.

6 Add another Entity Terminator block labeled Monitored and Discarded Emails. Connect all the blocks as shown in the model.

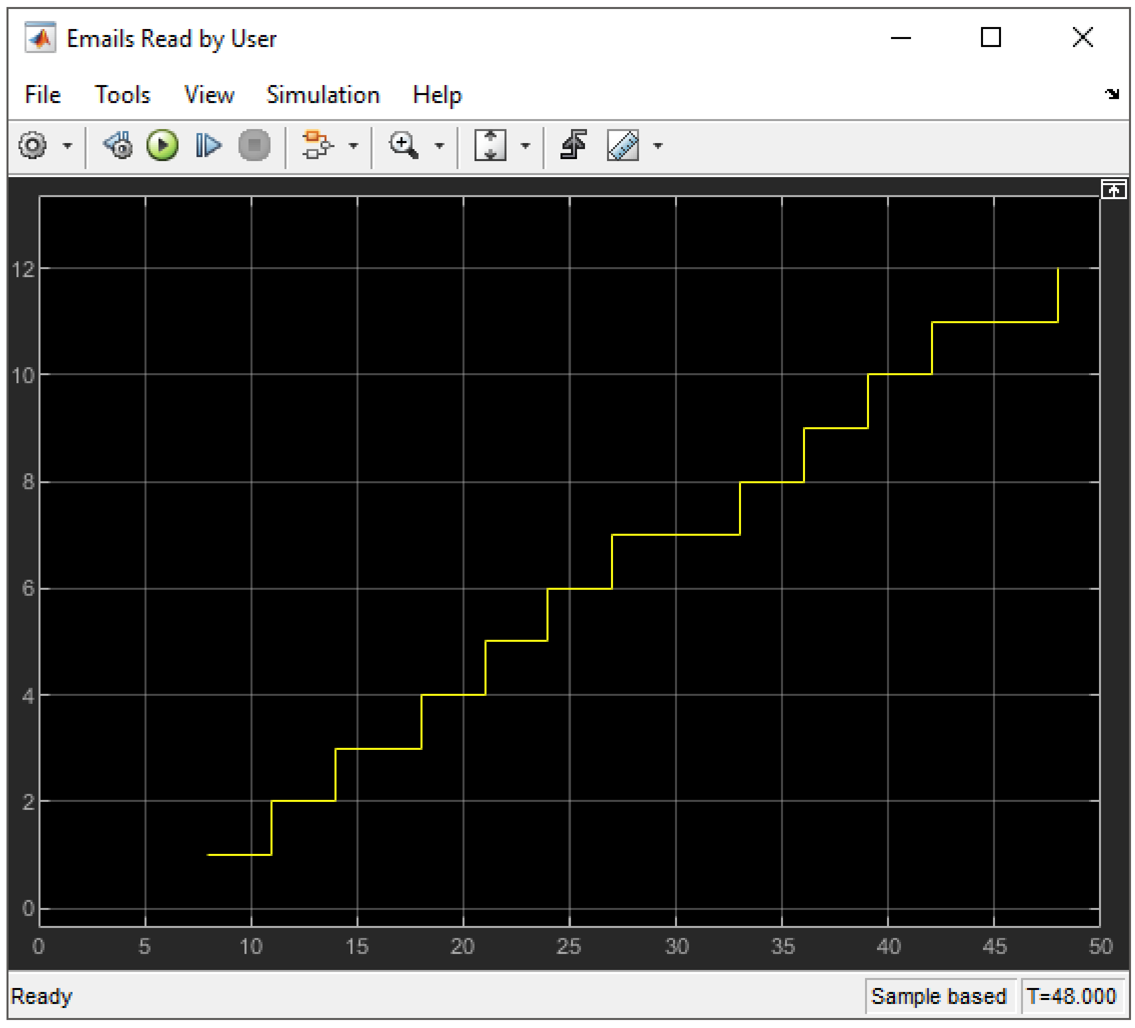
7 Output the **Number of entities arrived, a** statistic from all of the Entity Terminator blocks, and connect them to the Scope blocks for visualization.

8 Increase the simulation time to `50` and simulate the model. Observe the emails that are instantly discarded or discarded after monitoring.

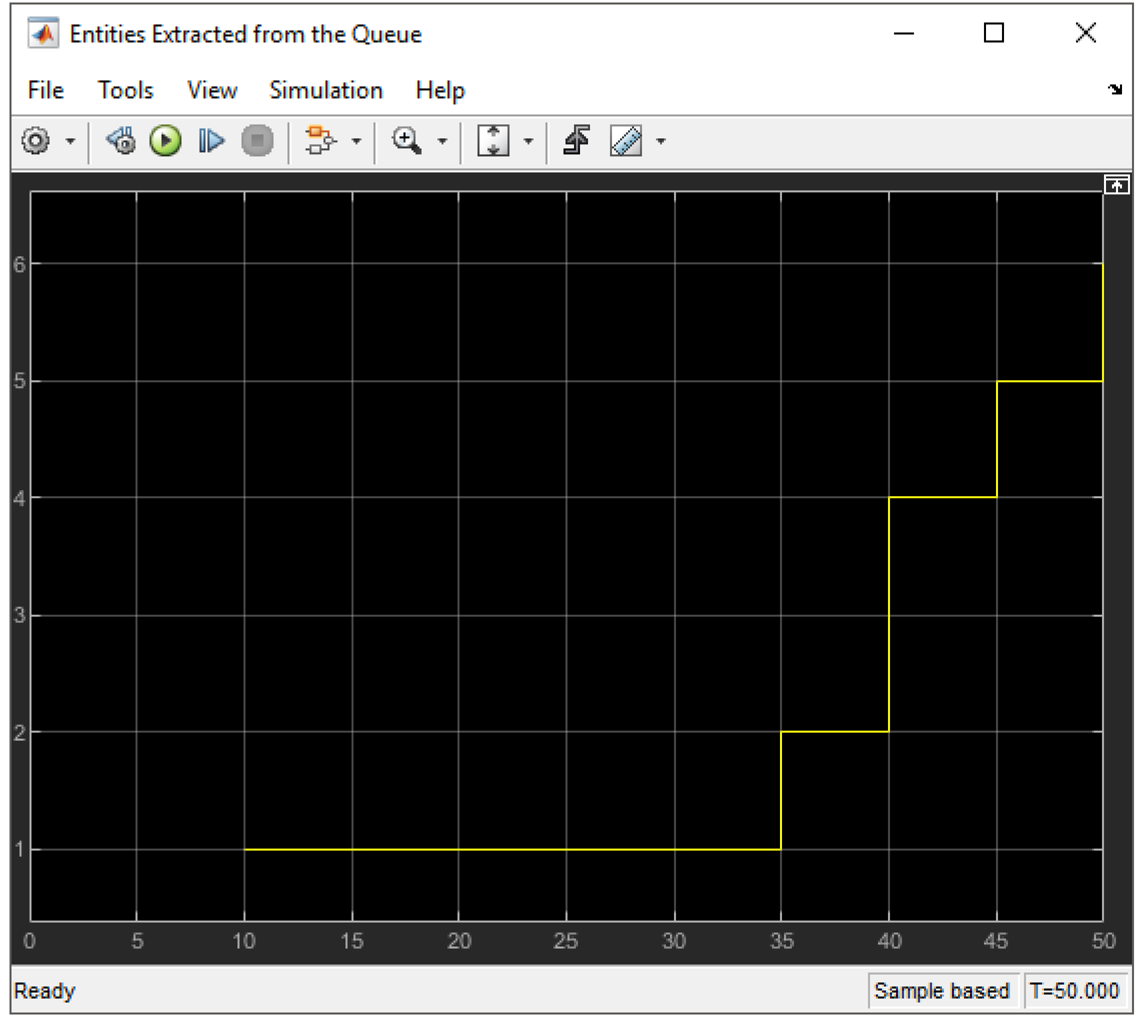

Instantly Discarded Emails
File
Tools
View
Simulation
Help
25
20
15
10
5
0
0
5
10
15
20
25
30
35
40
45
50
Ready
Sample based
T=50.000




Observe the emails that reach the user after the filtering.




**9** Optionally, visualize the number of extracted emails from any block in the model. For instance, in the Email Queue, select the **Number of entities extracted, ex** statistic and connect to a scope. Observe that six emails are extracted from the queue.




## See Also

Resource Acquirer | Resource Pool | Resource Releaser

## Related Examples

- "Optimize SimEvents Models by Running Multiple Simulations" on page 5-20

## More About

- "Model Using Resources" on page 4-2
- "Set Resource Amount with Attributes" on page 4-4

# Visualization, Statistics, and Animation


- “Interpret SimEvents Models Using Statistical Analysis” on page 5-2
- “Visualization and Animation for Debugging” on page 5-10
- “Model Traffic Intersections as a Queuing Network” on page 5-12
- “Optimize SimEvents Models by Running Multiple Simulations” on page 5-20
- “Use the Sequence Viewer to Visualize Messages, Events, and Entities” on page 5-24

# Interpret SimEvents Models Using Statistical Analysis

**In this section...**


"Output Statistics for Data Analysis" on page 5-2
"Output Statistics for Run-Time Control" on page 5-2
"Average Queue Length and Average Store Size" on page 5-4
"Average Wait" on page 5-6
"Number of Entities Arrived" on page 5-8
"Number of Entities Departed" on page 5-8
"Number of Entities Extracted" on page 5-8
"Number of Entities in Block" on page 5-8
"Number of Pending Entities" on page 5-8
"Pending Entity Present in Block" on page 5-9
"Utilization" on page 5-9


Choosing the right statistical measure is critical for evaluating the model performance. You can use output statistics from the SimEvents library blocks for data analysis and run-time control.

## Output Statistics for Data Analysis

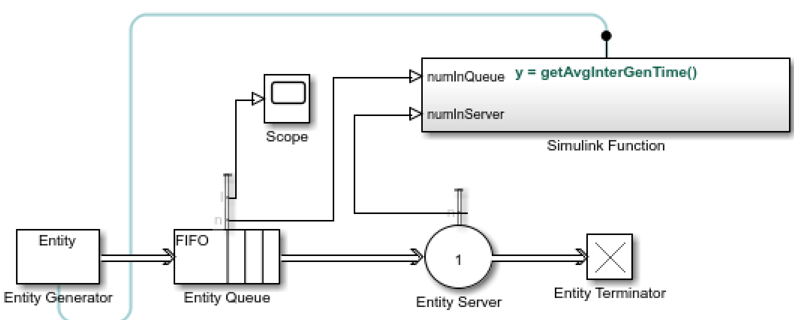
Consider these statistical measures for more efficient behavior interpretation.

- Identify the appropriate size of the samples to compute more meaningful statistics.
- Decide if you want to investigate the transient behavior, the steady-state behavior, or both.
- Specify the number of simulations that ensures sufficient confidence in the results.

For an example, see "Explore Statistics and Visualize Simulation Results".

## Output Statistics for Run-Time Control

Some systems rely on statistics to influence the dynamics. In this example, a queuing system with discouraged arrivals has a feedback loop that adjusts the arrival rate throughout the simulation based on the statistics reported by the queue and the server. To learn more details about this example, see "Adjust Entity Generation Times Through Feedback" on page 1-24.



A subset of the blocks in SimEvents library provides statistics output for run-time control. When you create simulations that use statistical signals to control the dynamics, you access the current statistical values at key times throughout the simulation, not just at the end of the simulation.

This table lists SimEvents blocks that output commonly used statistics for data analysis and run-time control.

| Block Name | Statistics Parameter | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | **Average queue length/ store size, l** | **Average wait, w** | **Number of entities arrived, a** | **Number of entities departed, d** | **Number of entities extracted, ex** | **Number of entities in block, n** | **Number of pending entities, np** | **Pending entity present in block, pe** | **Utilization, util** |
| Conveyor System | | | | ✓ | | ✓ | | ✓ | |
| Entity Batch Creator | | | ✓ | ✓ | | | | ✓ | |
| Entity Batch Splitter | | | ✓ | ✓ | | | | ✓ | |
| Entity Find | ✓ | ✓ | | ✓ | ✓ | | | | |
| Entity Generator | | | | ✓ | | | | ✓ | |
| Entity Queue | ✓ | ✓ | | ✓ | ✓ | ✓ | | | |
| Entity Selector | | | | ✓ | ✓ | ✓ | | | |
| Entity Server | | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Entity Store | ✓ | ✓ | | ✓ | ✓ | ✓ | | | |
| Entity Terminator | | | ✓ | | | | | | |
| Multicast Receive Queue | ✓ | ✓ | | ✓ | ✓ | ✓ | | | |
| Resource Acquirer | | ✓ | | ✓ | ✓ | ✓ | | | |
| Resource Pool | | | | | | | | | ✓ |

The statistical parameters are updated on particular events during the simulation. For example, when a full N-server advances one entity to the next block, the statistical signal representing the number of entities in the block assumes the value N-1. However, if the departure causes another entity to arrive at the block at the same time instant, then the statistical signal assumes the value N. The value of N-1, which does not persist for a positive duration, is a zero-duration value.. This phenomenon occurs in many situations.

This table lists the events that update the block statistics.

| Statistics Port | Updated on Event | | | | |
|---|---|---|---|---|---|
| | Entry | Exit | Blocked | Preempted | Extracted |
| **Average queue length/store size, l** | ✓ | ✓ | | | ✓ |
| **Average wait, w** | | ✓ | | ✓ | ✓ |
| **Number of entities arrived, a** | ✓ | | | | |
| **Number of entities departed, d** | | ✓ | | | ✓ |
| **Number of entities extracted, ex** | | | | | ✓ |
| **Number of entities in block, n** | ✓ | | | | ✓ |
| **Number of pending entities, np** | | ✓ | ✓ | ✓ | |
| **Pending entity present in block, pe** | | ✓ | ✓ | ✓ | |
| **Utilization, util** | ✓ | ✓ | | ✓ | ✓ |

## Average Queue Length and Average Store Size

### The formula to compute average queue length or store size

**Average queue length, l** is the accumulated time-weighted average queue. To compute **Average queue length, l** at time $t_f$, the block:

1. Multiplies the size of the queue $n$ by its duration, $t = t_i - t_{i-1}$, to calculate the time-weighted queue.
2. Sums over the time-weighted queue and averages it over total time $t_f$.

$$l = \frac{1}{t_f} \sum_{i=1}^{f} n_t \times t$$

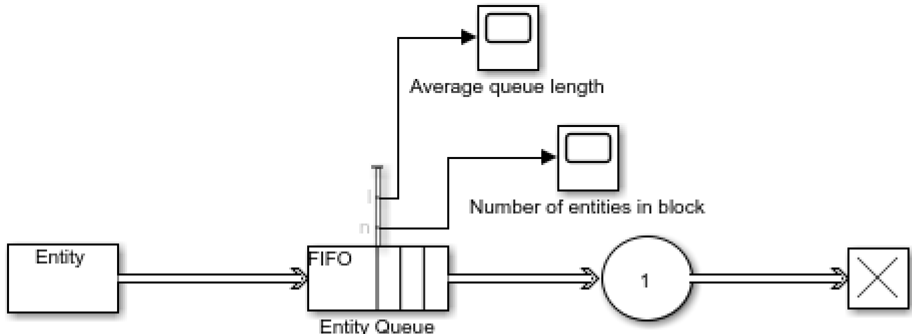
Where:

- $t$ is the time between the entity arrival and / or the number of departure events.
- $f$ is the total number of entity arrival and / or the number of departure events between $t_0$ and $t_f$.
- $i = 1$ for simulation time $t_0 = 0$ .

**Average store size, l** is computed similarly by replacing the queue length with the store size.

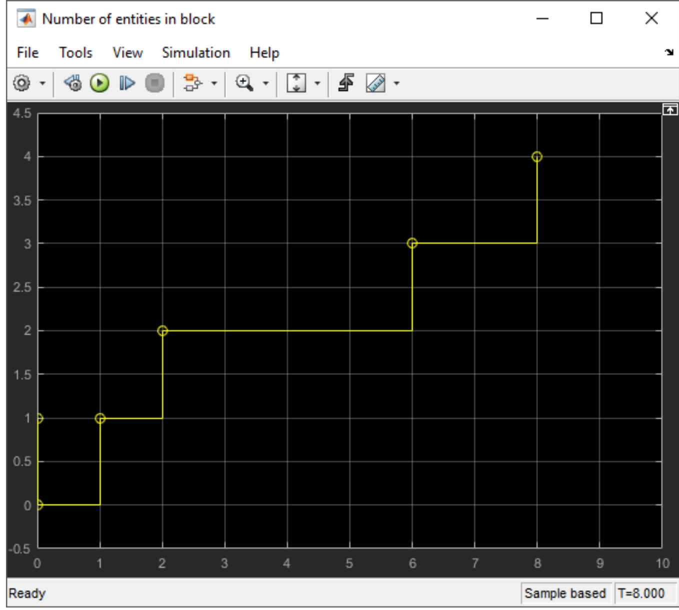
### Average queue length example in the Entity Queue block

This example shows the average queue length of the entities in the Entity Queue block.




### Calculate average queue length in the simple queuing system example

The service time for the Entity Server block is larger than the entity intergeneration time of the Entity Generator block. The entities are queued and sorted in the Entity Queue block. The scope displays the number of entities.



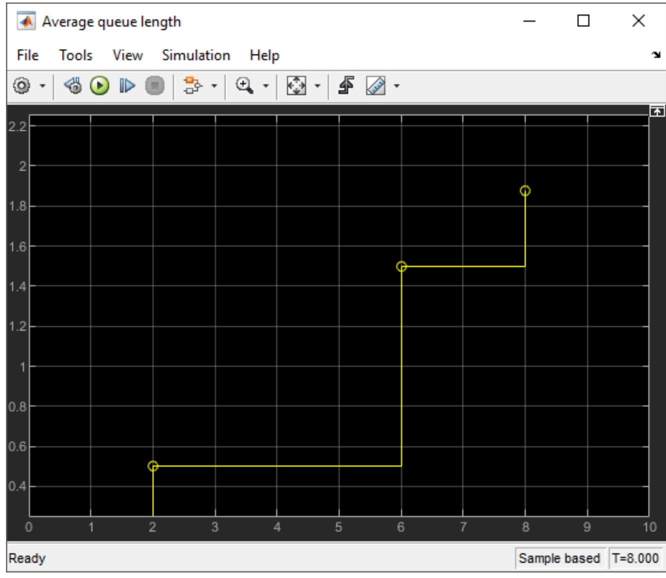
For the duration between 0 and 1, the average queue length is 0 because the size of the queue is 0. Between 1 and 2 the queue length is 1. Average queue length at time $t_f = 2$ is calculated as follows.

$$l = \frac{1}{2}\sum_{i=1}^{2} n_t \times t = \frac{1}{2}(0 + 1 \times 1) = 0.5$$

The queue size is 2 between the times 2 and 6 for the duration of 4. Average queue length at time $t_f = 6$ is calculated using this equation.

$$l = \frac{1}{6}\sum_{i=1}^{6} n_t \times t = \frac{1}{6}(0 + 1 \times 1 + 2 \times 4) = 1.5$$

The average queue size is calculated for each duration. The Scope block displays its value for the duration of the simulation.




## Average Wait

### The formula to compute average wait

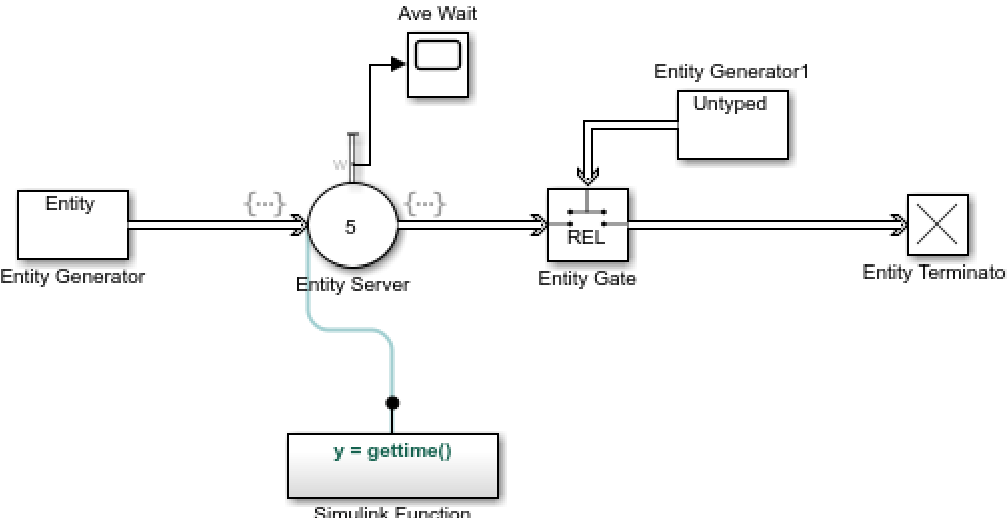
The **Average wait, w** parameter represents the sum of the wait times for entities departing the block, divided by their total number, *n*.

Wait time, $w_j$, is the simulated time that an entity resides within a block. This wait time is not necessarily equivalent to the time an entity is blocked. It is the duration between an entity's entry into and exit from a block. For instance, wait time is 1 for an entity that travels through an unblocked Entity Server with a service time of 1s.

$$w = \frac{\sum_{j=1}^{n} w_j}{n}$$

#### Average wait of entities example in the Entity Server block

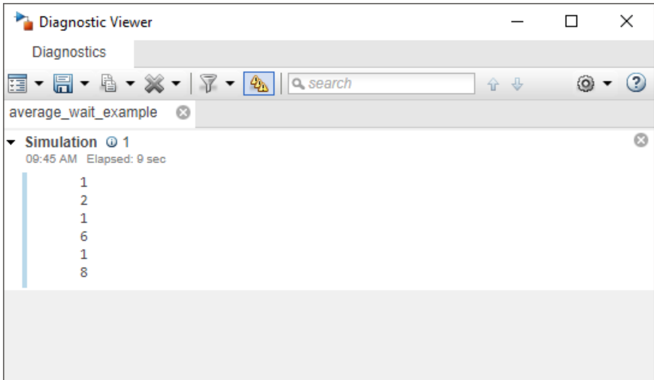
This example shows the average wait time for entities that are served in the Entity Server block.



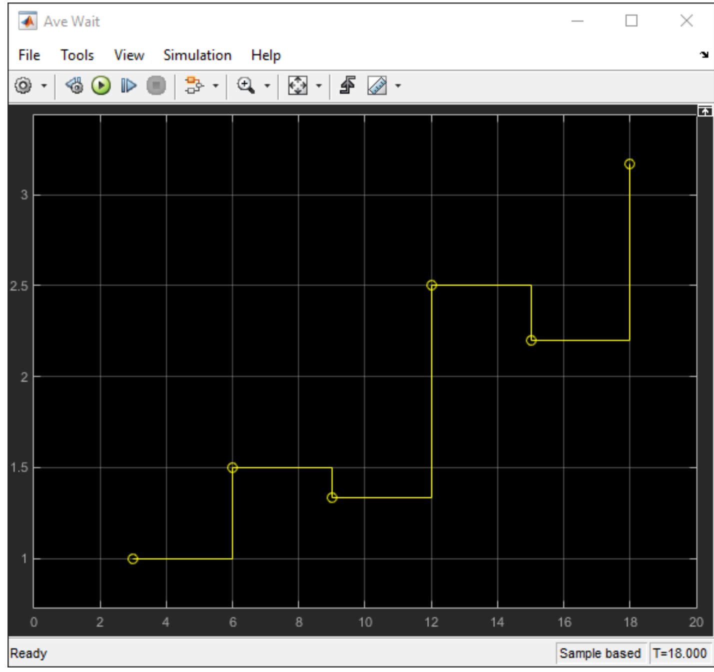

#### Calculate average wait in the example

The duration of an entity's entry into and exit from the Entity Server block is computed by the `gettime()` function in the Simulink Function block.

The Diagnostic Viewer displays the duration between the entry and exit of six consecutive entities.




The Scope block shows the average wait time for each entity departure event from the Entity Server block. For instance, the wait time for the first entity is `1` and the wait time for the second entity is `2`. The average wait time calculated for the first two entities is `1.5`. The plot displays this value at the simulation time `6`. For the first four entities, the sum of the wait times is `10` and the average wait time at simulation time `12` becomes `2.5`.




## Number of Entities Arrived

The **Number of entities arrived, a** parameter outputs the cumulative count for the number of entities that arrive at the block.

## Number of Entities Departed

The **Number of entities departed, d** parameter outputs the cumulative count for the number of entities that depart the block.

## Number of Entities Extracted

Entity Find block finds entities in a SimEvents model and extracts them from their location to reroute. The **Number of entities extracted, ex** parameter outputs the number of entities that are extracted from a block.

## Number of Entities in Block

The **Number of entities in block, n** parameter outputs the number of entities that are in the block.

## Number of Pending Entities

The **Number of pending entities, np** parameter outputs the number of pending entities the block has served that have yet to depart.

## Pending Entity Present in Block

The **Pending entity present in block, pe** parameter indicates whether an entity that is yet to depart is present in the block. The value is `1` if there are any pending entities, and `0` otherwise.

## Utilization

The **Utilization, util** parameter indicates the average time a block is occupied. The block calculates utilization for each entity departure event, which is the ratio of the total wait time for entities to the server capacity, *C*, multiplied by the total simulation time, $t_f$ . Utilization for *n* entities is calculated using this equation.

$$util = \frac{\sum_{j=1}^{n} w_j}{C \times t_f}$$

## References

[1] Cassandras, Christos G. *Discrete Event Systems: Modeling and Performance Analysis*. Homewood, Illinois: Irwin and Aksen Associates, 1993.

## See Also

Entity Generator | Entity Queue | Entity Server | Entity Terminator | Multicast Receive Queue | Resource Acquirer

## More About

- “Count Entities”
- “Visualization and Animation for Debugging” on page 5-10
- “Explore Statistics and Visualize Simulation Results”

# Visualization and Animation for Debugging

**In this section...**


“Which Debugging Tool to Use” on page 5-10

“Observe Entities with Animation” on page 5-11

“Explore the System Using the Simulink Simulation Stepper” on page 5-11

“Information About Race Conditions and Random Times” on page 5-11


Visualize and animate simulations in SimEvents models using tools available in Simulink and SimEvents software.

- You can place many Simulink Sink blocks directly on the entity line to observe entities, including the To Workspace and dashboard scopes.
- If the entity type is anonymous, you can place a Scope block.
- To observe bus or structured type entities, use the Simulation Data Inspector or dashboard scopes. The Scope and Display blocks do not support buses.

## Which Debugging Tool to Use

These tools help you explore various elements of a SimEvents model.

<table>
<tr><th>Items to Observe</th><th>Visualization Tool and Its Purpose</th></tr>
<tr><td>Statistics</td><td rowspan="2">• Simulation Data Inspector — Show the statistic throughout the simulation. For more information, see “Inspect and Analyze Simulation Results”.<br>• Simulink To Workspace block — Write the data set to the MATLAB workspace when the simulation stops or pauses.<br>• Simulink Scope block — Create a plot using the statistic.<br>• Simulink Display block — Show the statistic throughout the simulation.<br>• Simulink To File block — Write the data set into a MAT-file.<br>• Simulink Dashboard Scope block — Create a plot using the statistic.</td></tr>
<tr><td>Entities passing through model</td></tr>
<tr><td>Entity animation</td><td>Animation — Highlight active entities in the simulation.</td></tr>
<tr><td>Step of a Simulation</td><td>Simulink Simulation Stepper — Step forward and back through a simulation. For more information, see “Use Simulation Stepper”.</td></tr>
<tr><td>Custom animation</td><td>Use SimEvents custom visualization API — Create custom observers of the entities and events in a model. For more information, see “Use SimulationObserver Class to Monitor a SimEvents Model” on page 10-2.</td></tr>
</table>

**Note** The Simulink Floating Scope does not support SimEvents models.

Simulation Data Inspector is a unified user interface for viewing both entities and signal (for example, statistics) data. For more information, see “Inspect and Analyze Simulation Results”.

## Observe Entities with Animation

During simulation, animation provides visual verification that your model behaves as you expect. Animation highlights active entities in a model as execution progresses. You can control the speed of entity activity animation during simulation, or turn off animation. In a model window, right-click and select **Animation Speed**.

- **Fast**
- **Medium**
- **Slow**
- **None**

The **Fast** animation speed shows the active highlights at each time step. To add delay with each time step, set the animation speed to **Medium** or **Slow**. To turn off the animation, select **None**.

## Explore the System Using the Simulink Simulation Stepper

Simulation Stepper enables you to step through major time steps of a simulation. Use this tool to explore your discrete-event system. For more information, see “Simulation Stepper”.

## Information About Race Conditions and Random Times

You can vary the processing sequence for simultaneous events or make the intergeneration times or service times random.

## See Also

Entity Generator | Entity Queue | Entity Server | Entity Terminator | Multicast Receive Queue | Resource Acquirer

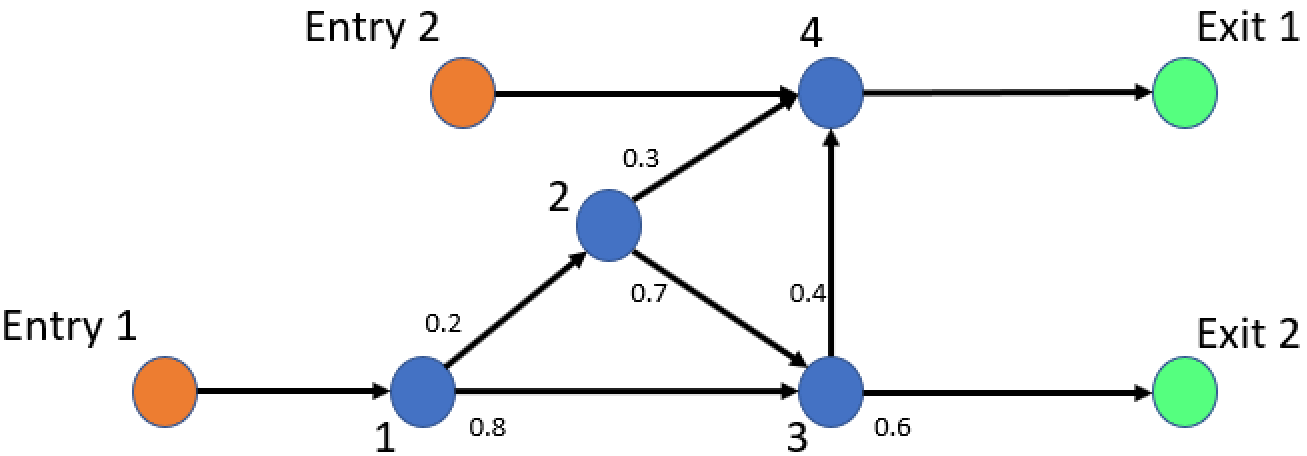
## More About

- “Explore Statistics and Visualize Simulation Results”
- “Count Entities”

# Model Traffic Intersections as a Queuing Network

This example shows how to create a SimEvents® model to represent a vehicle traffic network and to investigate mean waiting time of vehicles when the network is in steady-state.

Suppose a vehicle traffic network consists of two vehicle entry and two vehicle exit points, represented by brown and green nodes in the next figure. Each blue node in the network represents a route intersection with a traffic light, and the arrows represent the route connections at each intersection. The values next to the arrows represent the percentage of vehicles taking the route in that intersection.

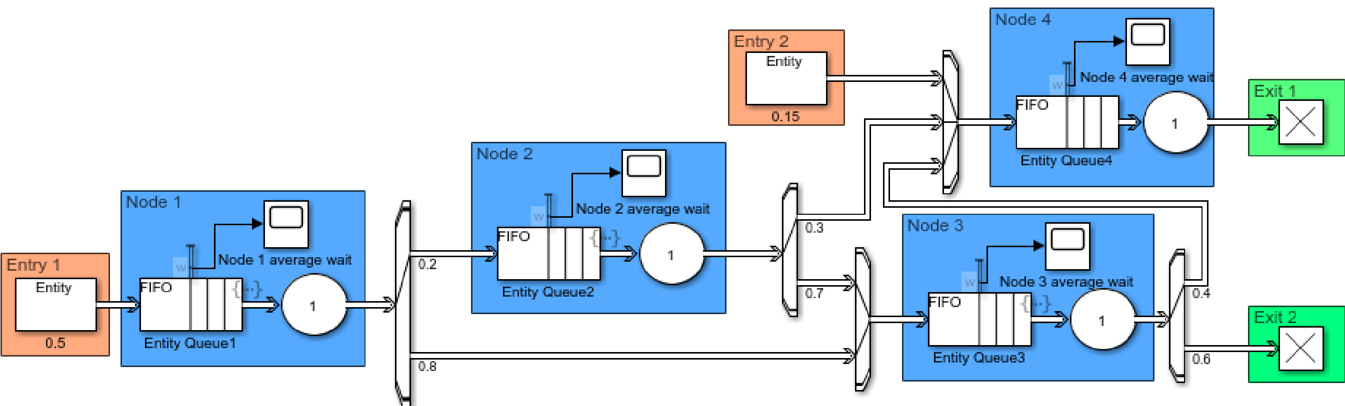



The rate of vehicle entries into the network are represented by the Poisson processes with rates `0.5` for Entry `1` and `0.15` for Entry `2`. Service rates represent the time vehicles spend at each intersection, which are drawn from exponential distribution with mean `1`. The arrow values are the probabilities of choosing a route for vehicles in the intersection.

#### Model Traffic Network

To represent a vehicle traffic network, this model uses Entity Generator, Entity Server, Entity Queue, Entity Input Switch, Entity Output Switch, and Entity Terminator blocks.

```
model = 'QueueServerTransportationNetwork';
open_system(model);
```




### Model Vehicle Arrivals

The two Entity Generator blocks represent the network entry points. Their entity intergeneration time is set to create a Poisson arrival process.

This is the code in the **Intergeneration time action** field of the Entry 1 block.

```
% Random number generation
coder.extrinsic('rand');
ValEntry1 = 1;
ValEntry1 = rand();
% Pattern: Exponential distribution
mu = 0.5;
dt = -1/mu * log(1 - ValEntry1);
```

In the code, `mu` is the Poisson arrival rate. The `coder.extrinsic('rand')` is used because there is no unique seed assigned for the randomization. For more information about random number generation in event actions, see "Event Action Languages and Random Number Generation" on page 1-8. To learn more about extrinsic functions, see "Working with mxArrays".

### Model Vehicle Route Selection

Entities have a `Route` attribute that takes value `1` or `2`. The value of the attribute determines the output port from which the entities depart an Entity Output Switch block.

This code in the **Entry action** of the Entity Server `1` represents the random route selections of vehicles at the intersection represented by Node `1`.

```
Coin1 = 1;
coder.extrinsic('rand');
Coin1 = rand;
if Coin1 <= 0.2
    entity.Route = 1;
else
    entity.Route = 2;
end
```

This is an example of random `Route` attribute assignments when entities enter the Entity Server 1 block. The value of `Route` is assigned based on the value of the random variable `rand` that takes

values between 0 and 1. Route becomes 1 if rand is less than or equal to 0.2, or 2 if rand is greater than 0.2.

#### Model Route Intersections

Each blue node represents a route intersection and includes an infinite capacity queue, and a server with service time drawn from an exponential distribution with mean 1.

Entity Server 1 contains this code.

```
% Pattern: Exponential distribution
coder.extrinsic('rand');
Val1 = 1;
Val1 = rand();
mu = 1;
dt = -mu * log(1 - Val1);
```

#### Calculate Mean Waiting Time for Vehicles in the Network

The network is constructed as an open Jackson network that satisfies these conditions.

- All arriving vehicles can exit the network.
- Vehicle arrivals are represented by Poisson process.
- Vehicles depart an intersection as first-in first-out. The wait time in an intersection is exponentially distributed with mean 1.
- A vehicle departing the intersection either takes an available route or leaves the network.
- The utilization of each traffic intersection queue is less than 1.

In the steady state, every queue in an open Jackson network behaves independently as an M/M/1 queue. The behavior of the network is the product of individual queues in equilibrium distributions. For more information about M/M/1 queues, see “M/M/1 Queuing System” on page 6-37.

The vehicle arrival rate for each node $i$ is calculated using this formula.

$$\lambda_i = r_i + \sum_{j=1}^{N} \theta_{ji} \lambda_j,$$

In the formula:

- $r_i$ is the rate of external arrivals for node $i$.
- $j = 1, \ldots, N$ is the total number of incoming arrows to node $i$.
- $\theta_{ji}$ is the probability of choosing the node $i$ from node $j$.
- $\lambda_j$ is the total vehicle arrival rate to node $j$.

For all of the nodes in the network, the equation takes this matrix form.

$$\lambda = R(I - \theta)^{-1}$$

Here, $\theta$ is the routing matrix, and each element represents the probability of transition from node $j$ to node $i$.

For the network investigated here, this is the routing matrix.

$$\theta = \begin{bmatrix} 0 & 0.2 & 0.8 & 0 \\ 0 & 0 & 0.7 & 0.3 \\ 0 & 0 & 0 & 0.4 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$R$ is the vector of external arrivals to each node.

$$R = \begin{bmatrix} 0.5 & 0 & 0 & 0.15 \end{bmatrix}$$

Using these values, the mean arrival rate is calculated for each node.

$$\lambda = \begin{bmatrix} 0.5 & 0.1 & 0.47 & 0.368 \end{bmatrix}$$

Each node behaves as an independent M/M/1 queue, and the mean waiting time for each node $i$ is calculated by this formula. See “M/M/1 Queuing System” on page 6-37.

$$\rho_i = \frac{\lambda_i}{1-\lambda_i}$$

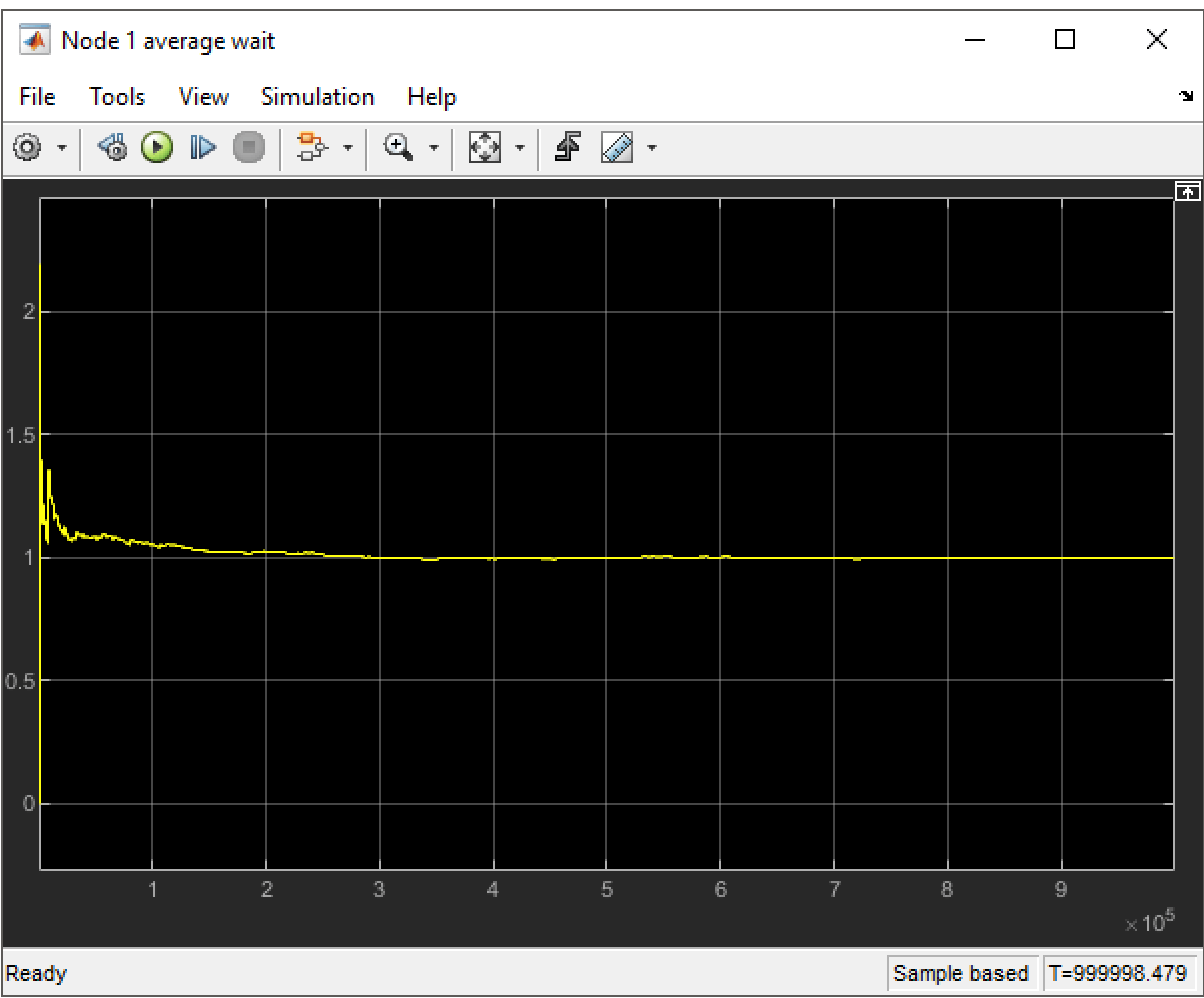
Mean waiting time for each node is calculated by incorporating each element of $\lambda$.

$$\rho = \begin{bmatrix} 1 & 0.11 & 0.88 & 0.58 \end{bmatrix}$$

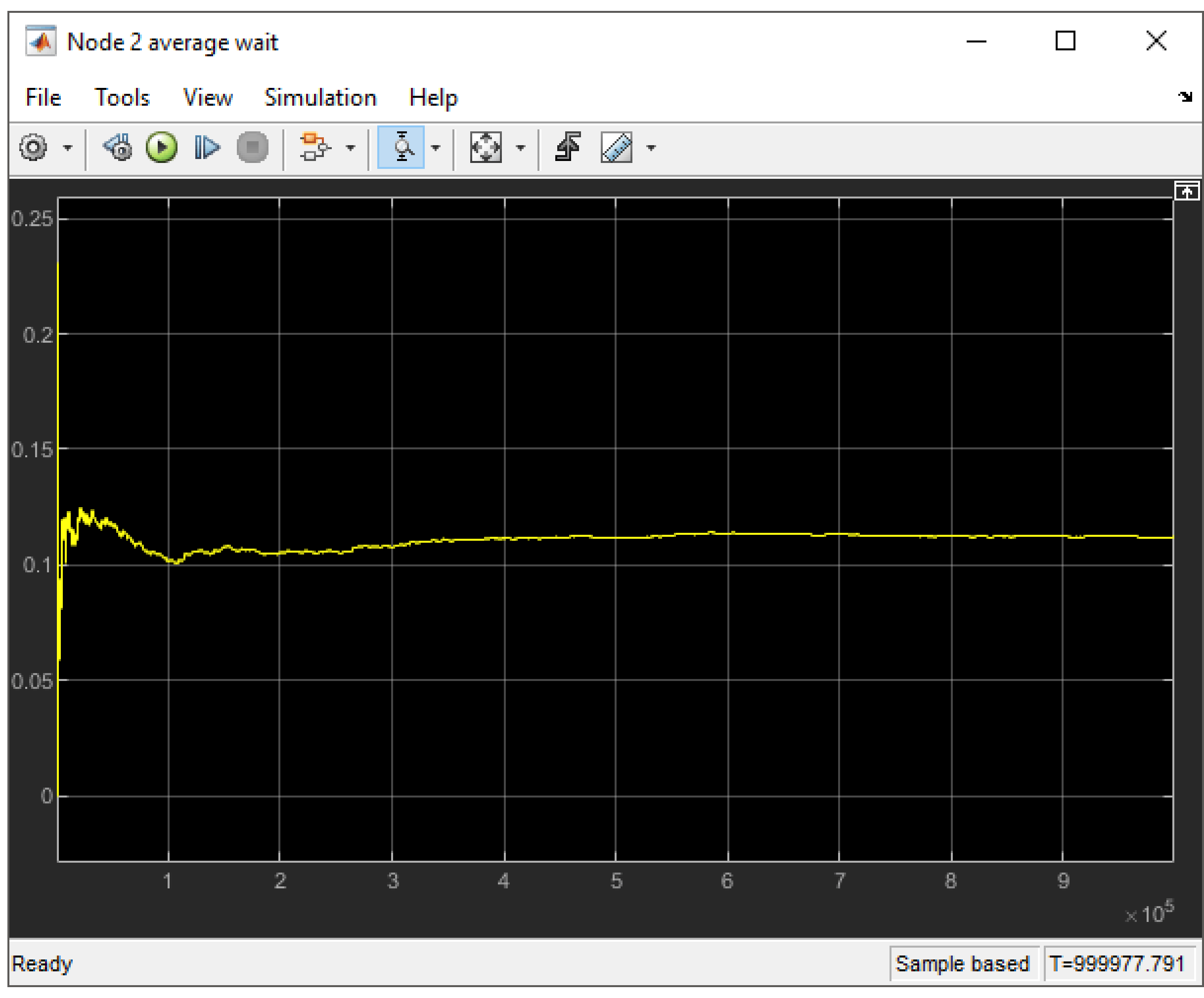
### View Simulation Results

Simulate the model and observe that the mean waiting time in each queue in the network matches the calculated theoretical results.

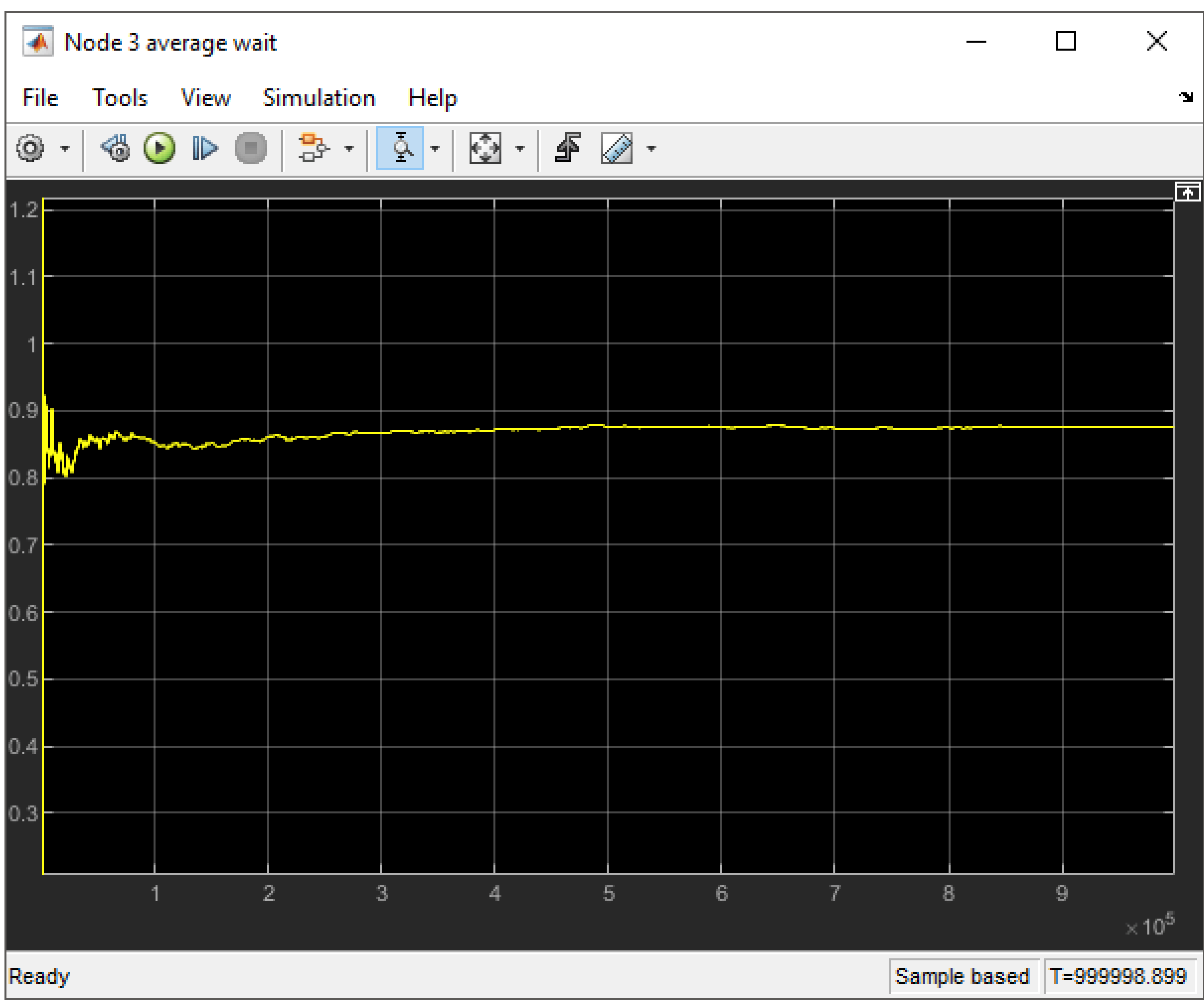
- The waiting time for the queue in node `1` converges to `1`.



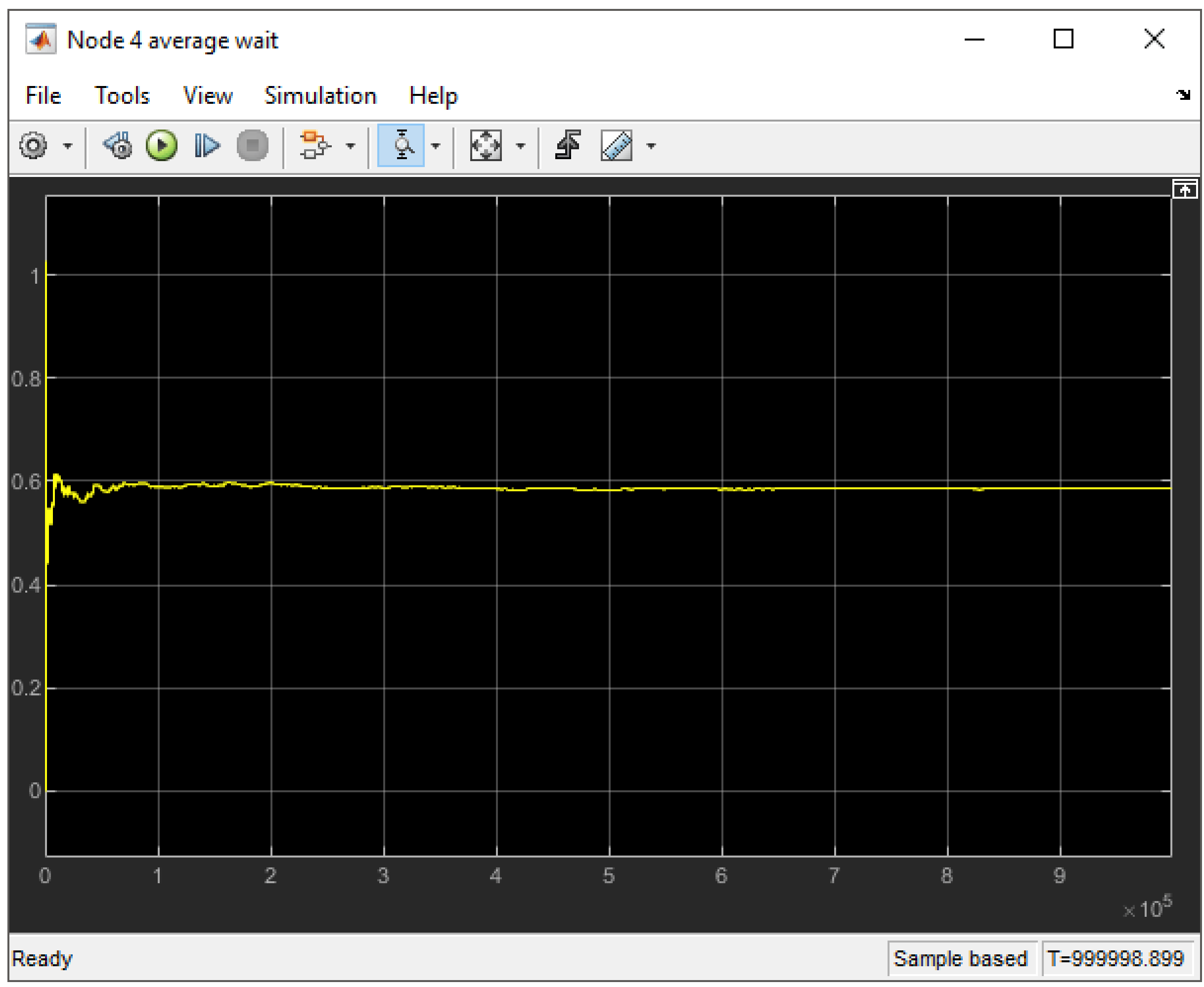

- The waiting time for the queue in node `2` converges to `0.11`.




- The waiting time for the queue in node `3` converges to `0.88`.




- The waiting time for the queue in node 4 converges to `0.58`.




## References

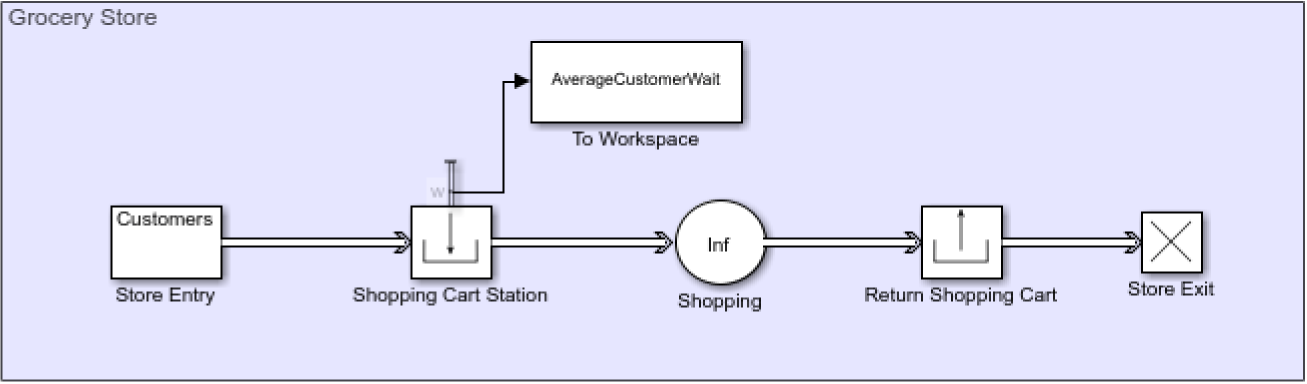
[1] Jackson, James R. *Operations research* Vol. 5, No. 4 (Aug., 1957), pp 518-521.

# Optimize SimEvents Models by Running Multiple Simulations

To optimize models in workflows that involve running multiple simulations, you can create simulation tests using the `Simulink.SimulationInput` object.

## Grocery Store Model

The grocery store example uses multiple simulations approach to optimize the number of shopping carts required to prevent long customer waiting lines.




In this example, the Entity Generator block represents the customer entry to the store. The customers wait in line if necessary and get a shopping cart through the Resource Acquirer block. The Resource Pool block represents the available shopping carts in the store. The Entity Server block represents the time each customer spends in the store. The customers return the shopping carts through the Resource Releaser block, while the Entity Terminator block represents customer departure from the store. The Average wait, w statistic from the Resource Acquirer block is saved to the workspace by the To Workspace block from the Simulink® library.

## Build the Model

Grocery store customers wait in line if there are not enough shopping carts. However, having too many unused shopping carts is considered a waste. The goal of the example is to investigate the average customer wait time for a varying number of available shopping carts in the store. To compute the average customer wait time, multiple simulations are run by using the `sim` command. For each simulation, a single available shopping cart value is used. For more information on the `sim` command, see “Run Parallel Simulations”.

In the simulations, the available shopping cart value ranges from `20` to `50` and in each simulation it increases by `1`. It is assumed that during the operational hours, customers arrive at the store with a random rate drawn from an exponential distribution and their shopping duration is drawn from a uniform distribution.

1. In the Entity Generator block, set the **Entity type name** to `Customers` and the **Time source** to `MATLAB action`. Then, enter this code.

   ```
   persistent rngInit;
   if isempty(rngInit)
       seed = 12345;
       rng(seed);
       rngInit = true;
   end

   % Pattern: Exponential distribution
   mu = 1;
   dt = -mu*log(1-rand());
   ```

   The time between the customer arrivals is drawn from an exponential distribution with mean `1` minute.
2. In the Resource Pool block, specify the **Resource name** as `ShoppingCart`. Set the **Resource amount** to `20`.

   Initial value of available shopping carts is `20`.
3. In the Resource Acquirer block, set the `ShoppingCart` as the **Selected Resources**, and set the **Maximum number of waiting entities** to `Inf`.

   The example assumes a limitless number of customers who can wait for a shopping cart.
4. In the Entity Server block, set the **Capacity** to `Inf`.

   The example assumes a limitless number of customers who can shop in the store.
5. In the Entity Server block, set the **Service time source** to `MATLAB action` and enter the code below.

   ```
   persistent rngInit;
   if isempty(rngInit)
       seed = 123456;
       rng(seed);
       rngInit = true;
   end

   % Pattern: Uniform distribution
   % m: Minimum, M: Maximum
   m = 20;
   M = 40;
   dt = m+(M-m)*rand;
   ```

   The time a customer spends in the store is drawn from a uniform distribution on the interval between `20` minutes and `40` minutes.
6. Connect the **Average wait, w** statistic from the Resource Acquirer block to a To Workspace block and set its **Variable name** to `AverageCustomerWait`.
7. Set the simulation time to `600`.

   The duration of one simulation is `10` hours of operation which is `600` minutes.
8. Save the model.

   For this example, the model is saved with the name `GroceryStore_ShoppingCartExample`.

## Run Multiple Simulations to Optimize Resources

1. Open a new MATLAB script and run this MATLAB code for multiple simulations.

   a. Initialize the model and the available number of shopping carts for each simulation, which determines the number of simulations.

   ```
   % Initialize the Grocery Store model with
   % random intergeneration time and service time value
   mdl = 'GroceryStore_ShoppingCartExample';
   isModelOpen = bdIsLoaded(mdl);
   open_system(mdl);

   % Range of number of shopping carts that is
   % used in each simulation
   ShoppingCartNumber_Sweep = (20:1:50);
   NumSims = length(ShoppingCartNumber_Sweep);
   ```

   In each simulation, number of available shopping carts is increased by 1.

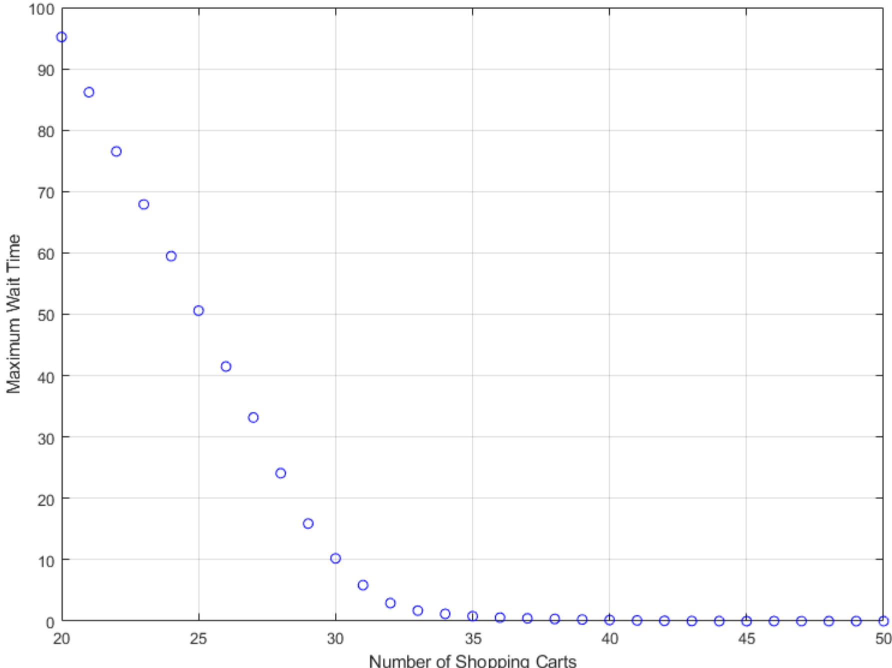
   b. Run each simulation with the corresponding available shopping cart value and output the results.

   ```
   % Run NumSims number of simulations
   NumCustomer = zeros(1,NumSims);
   for i = 1:1:NumSims
       in(i) = Simulink.SimulationInput(mdl);
       % Use one ShoppingCartNumber_sweep value for each iteration
       in(i) = setBlockParameter(in(i), [mdl '/Resource Pool'], ...
       'ResourceAmount', num2str(ShoppingCartNumber_Sweep(i)));
   end

   % Output the results for each simulation
   out = sim(in);
   ```

   c. Gather and visualize the results.

   ```
   % Compute maximum average wait time for the
   % customers for each simulation
   MaximumWait = zeros(1,NumSims);
   for i=1:NumSims
       MaximumWait(i) = max(out(1, i).AverageCustomerWait.Data);
   end
   % Visualize the plot
   plot(ShoppingCartNumber_Sweep, MaximumWait,'bo');
   grid on
   xlabel('Number of Available Shopping Carts')
   ylabel('Maximum Wait Time')
   ```

2. Observe the plot that displays the maximum average wait time for the customers as a function of available shopping carts.




The plot displays the tradeoff between having 46 shopping carts available for zero wait time versus 33 shopping carts for a 2-minute customer wait time.

## See Also

Entity Generator | Entity Queue | Entity Server | Entity Terminator | Resource Acquirer | Resource Pool | Resource Releaser

## Related Examples

- "Optimization of Shared Resources in a Batch Production Process" on page 6-102
- "Explore Statistics and Visualize Simulation Results"

## More About

- "Interpret SimEvents Models Using Statistical Analysis" on page 5-2
- "Count Entities"
- "Visualization and Animation for Debugging" on page 5-10
- "Adjust Entity Generation Times Through Feedback" on page 1-24
- "Save SimEvents Simulation Operating Point" on page 6-4

# Use the Sequence Viewer to Visualize Messages, Events, and Entities

To see the interchange of messages and events between the blocks from the Simulink Messages & Events library, Stateflow charts in Simulink models, and SimEvents blocks, you can:

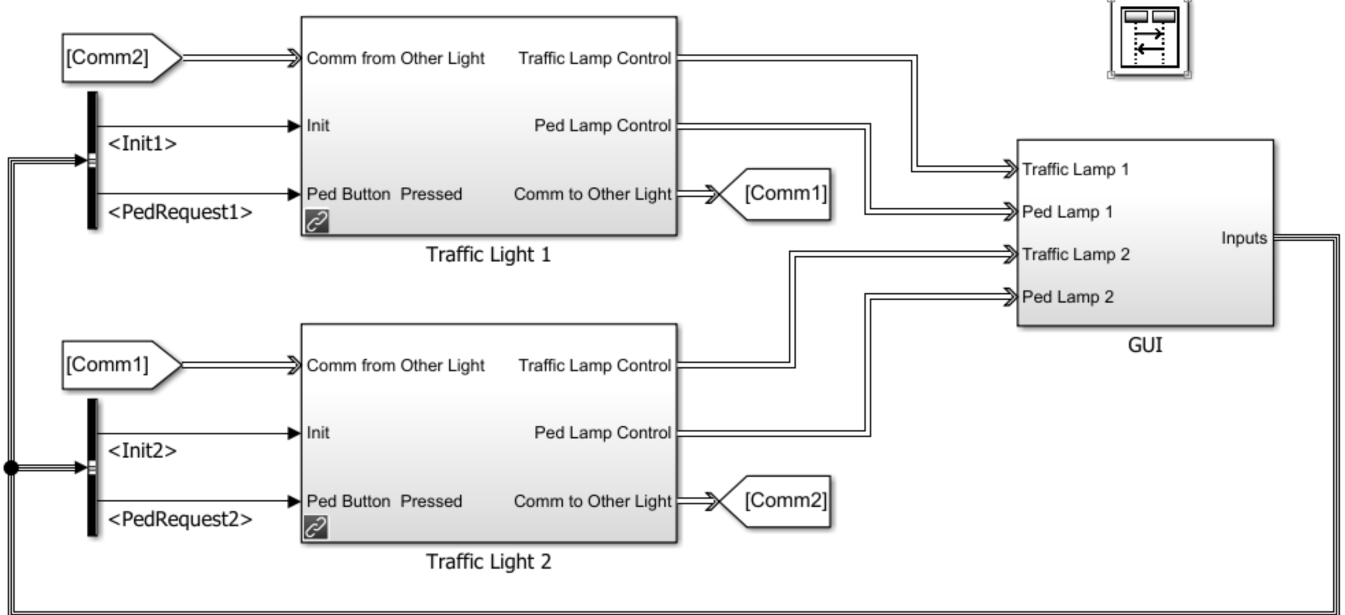
- Use the Sequence Viewer tool from the Simulink toolstrip.
- Add a Sequence Viewer block to your Simulink model.

The Sequence Viewer allows you to visualize message transition events and the data that the messages carry. In the Sequence Viewer, you can view event data related to Stateflow chart execution and the exchange of messages between Stateflow charts. The Sequence Viewer window shows messages as they are created, sent, forwarded, received, and destroyed at different times during model execution. The Sequence Viewer window also displays state activity, transitions, and function calls to Stateflow graphical functions, Simulink functions, and MATLAB functions.

With the Sequence Viewer, you can also visualize the movement of entities between blocks when simulating SimEvents models. All SimEvents blocks that can store entities appear as lifelines in the Sequence Viewer window. Entities moving between these blocks appear as lines with arrows. You can view calls to Simulink Function blocks and to MATLAB Function blocks.

You can add a Sequence Viewer block to the top level of a model or any subsystem. If you place a Sequence Viewer block in a subsystem that does not have messages, events, or state activity, the Sequence Viewer window informs you that there is nothing to display.

For instance, suppose that you simulate the Stateflow example `sf_msg_traffic_light`.




Copyright 2015, The MathWorks, Inc.

This model has three Simulink subsystems: Traffic Light 1, Traffic Light 2, and GUI. The Stateflow charts in these subsystems exchange data by sending messages. As messages pass through the

system, you can view them in the Sequence Viewer window. The Sequence Viewer window represents each block in the model as a vertical lifeline with simulation time progressing downward.

## Components of the Sequence Viewer Window

### Navigation Toolbar

At the top of the Sequence Viewer window, a navigation toolbar displays the model hierarchy path. Using the toolbar buttons, you can:

- Show or hide the Property Inspector.
- Select an automatic or manual layout.
- Show or hide inactive lifelines.
- Save Sequence Viewer settings.
- Restore Sequence Viewer settings.
- Configure Sequence Viewer parameters.
- Access the Sequence Viewer documentation.

### Property Inspector

In the Property Inspector, you can choose filters to show or hide:

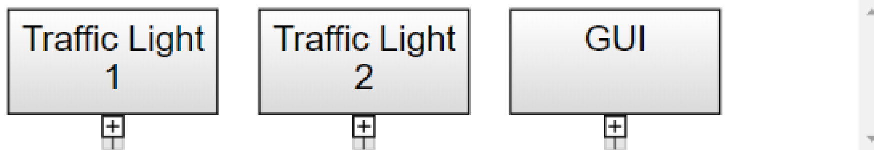
- Events
- Messages
- Function Calls
- State Changes and Transitions

**Header Pane**

The header pane below the Sequence Viewer toolbar shows lifeline headers containing the names of the corresponding blocks in a model.

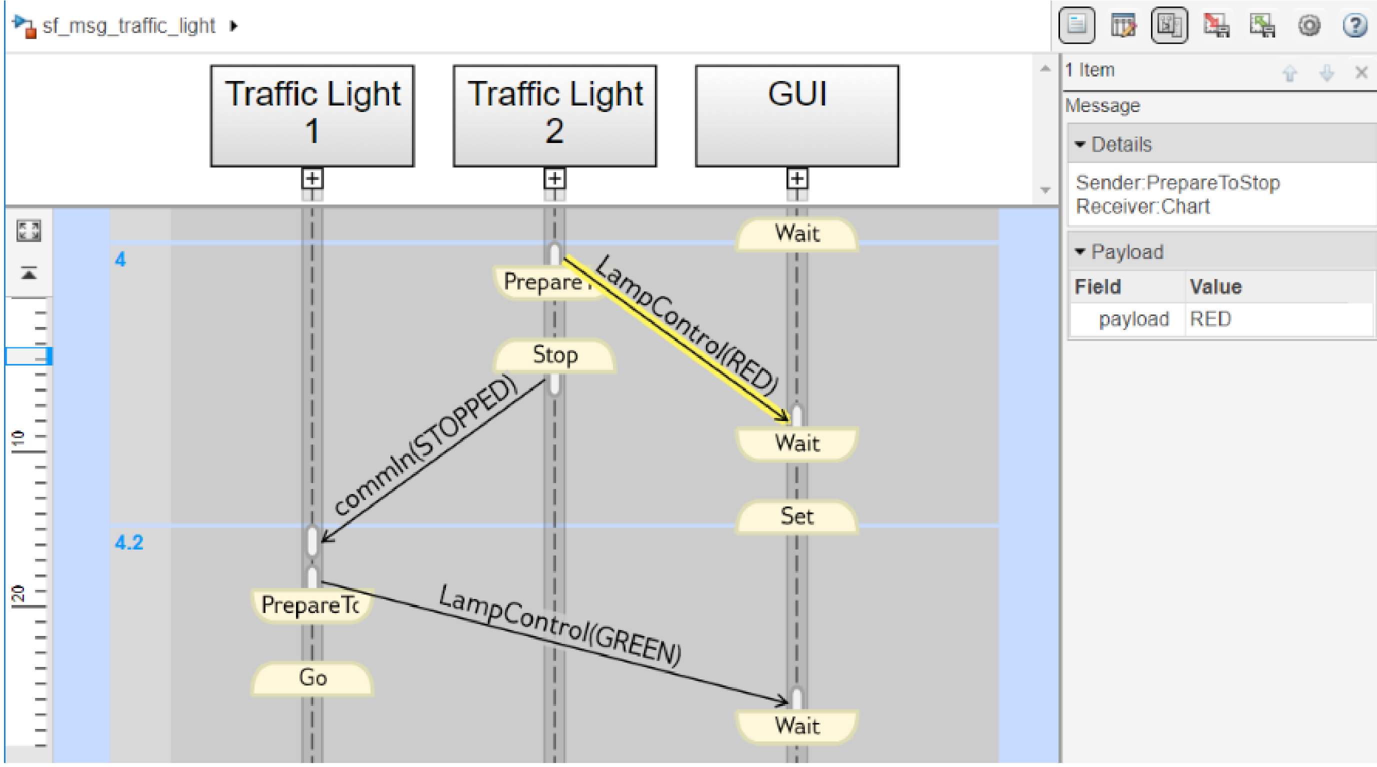
- Gray rectangular headers correspond to subsystems.
- White rectangular headers correspond to masked subsystems.
- Yellow headers with rounded corners correspond to Stateflow charts.

To open a block in the model, click the name in the corresponding lifeline header. To show or hide a lifeline, double-click the corresponding header. To resize a lifeline header, click and drag its right-hand side. To fit all lifeline headers in the Sequence Viewer window, press the space bar.




**Message Pane**

Below the header pane is the message pane. The message pane displays messages, events, and function calls between lifelines as arrows from the sender to the receiver. To display sender, receiver, and payload information in the Property Inspector, click the arrow corresponding to the message, event, or function call.



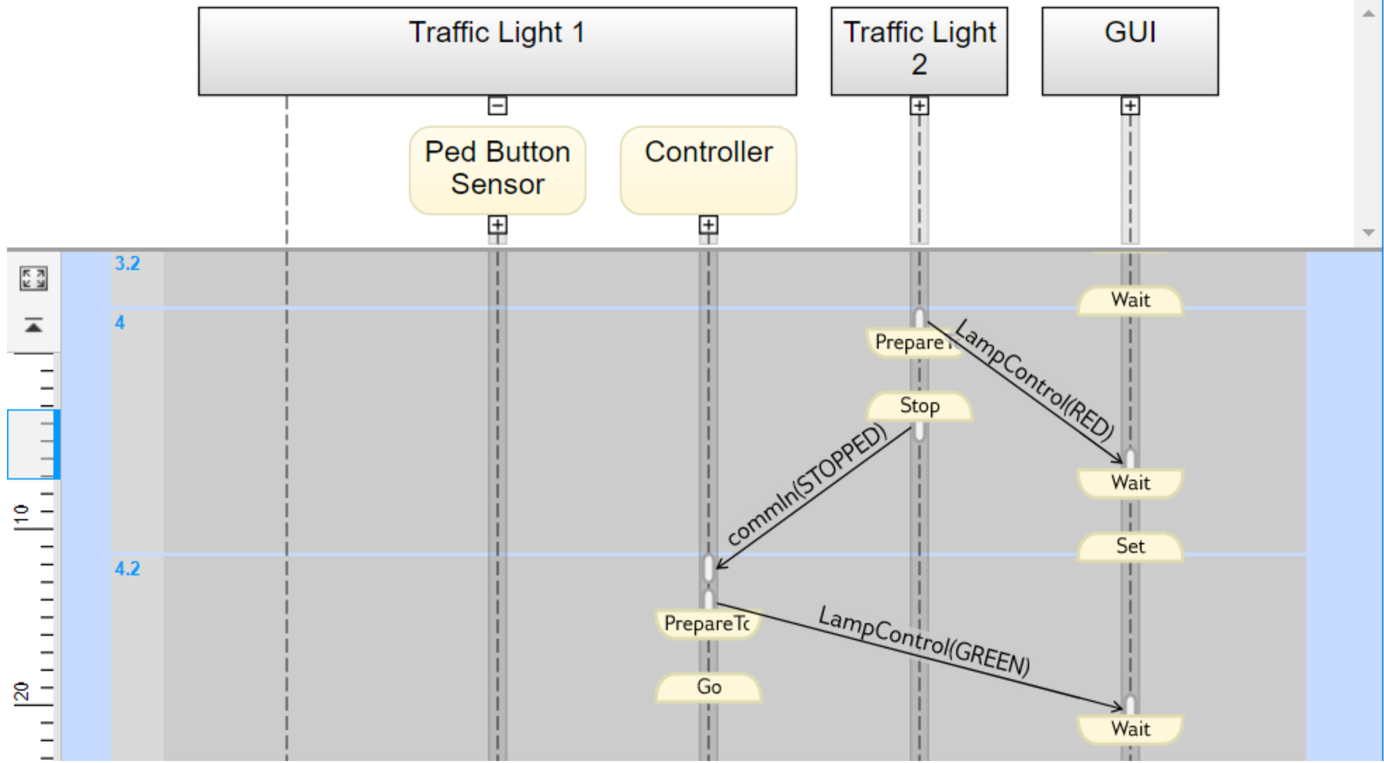

## Navigate the Lifeline Hierarchy

In the Sequence Viewer window, the hierarchy of lifelines corresponds to the model hierarchy. When you pause or stop the model, you can expand or contract lifelines and change the root of focus for the viewer.

### Expand a Parent Lifeline

In the message pane, a thick, gray lifeline indicates that you can expand the lifeline to see its children. To show the children of a lifeline, click the expander icon ⊞ below the header or double-click the parent lifeline.

For example, expanding the lifeline for the Traffic Light 1 block reveals two new lifelines corresponding to the Stateflow charts Ped Button Sensor and Controller.

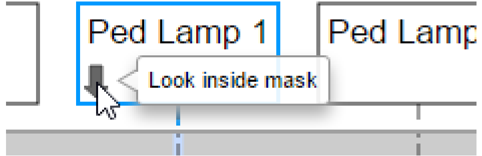



### Expand a Masked Subsystem Lifeline

The Sequence Viewer window displays masked subsystems as white blocks. To show the children of a masked subsystem, point over the bottom left corner of the lifeline header and click the arrow.

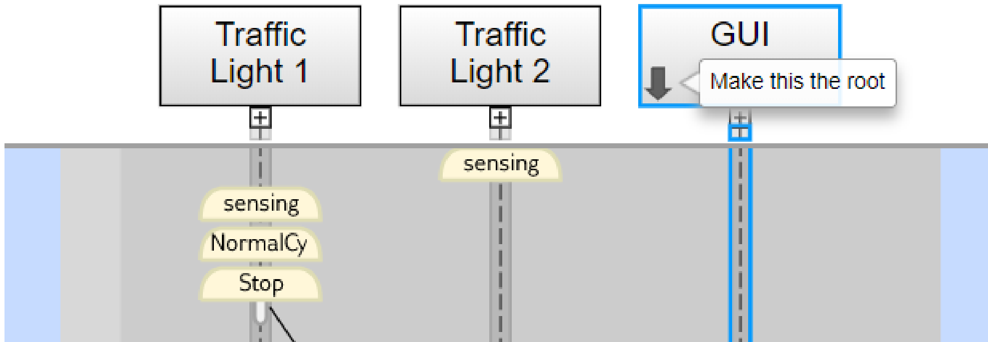
For example, the GUI subsystem contains four masked subsystems: Traffic Lamp 1,Traffic Lamp 2, Ped Lamp 1, and Ped Lamp 2.

You can display the child lifelines in these masked subsystems by clicking the arrow in the parent lifeline header.

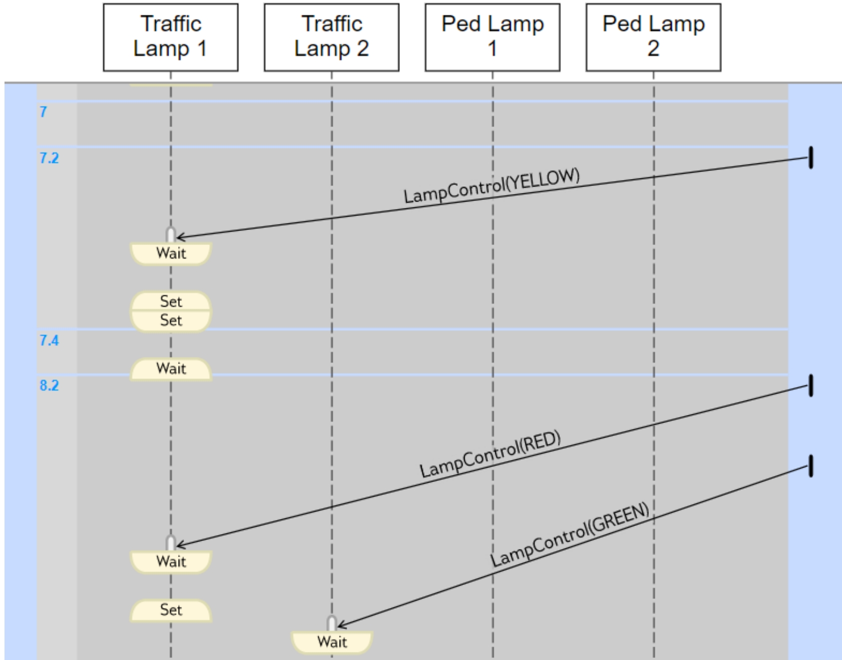


### Change Root of Focus

To make a lifeline the root of focus for the viewer, point over the bottom left corner of the lifeline header and click the arrow. Alternatively, you can use the navigation toolbar at the top of the Sequence Viewer window to move the current root up and down the lifeline hierarchy. To move the current root up one level, press the **Esc** key.




The Sequence Viewer window displays the current root lifeline path and shows its child lifelines. Any external events and messages are displayed as entering or exiting through vertical slots in the diagram gutter. When you point to a slot in the diagram gutter, a tooltip displays the name of the sending or receiving block.




## View State Activity and Transitions

To see state activity and transitions in the Sequence Viewer window, expand the state hierarchy until you have reached the lowest child state. Vertical yellow bars show which state is active. Blue horizontal arrows denote the transitions between states.

In this example, you can see a transition from `Go` to `PrepareToStop` followed, after 1 second, by a transition to `Stop`.

To display the start state, end state, and full transition label in the Property Inspector, click the arrow corresponding to the transition.

To display information about the interactions that occur while a state is active, click the yellow bar corresponding to the state. In the Property Inspector, use the **Search Up** and **Search Down** buttons to move through the transitions, messages, events, and function calls that take place while the state is active.

## View Function Calls

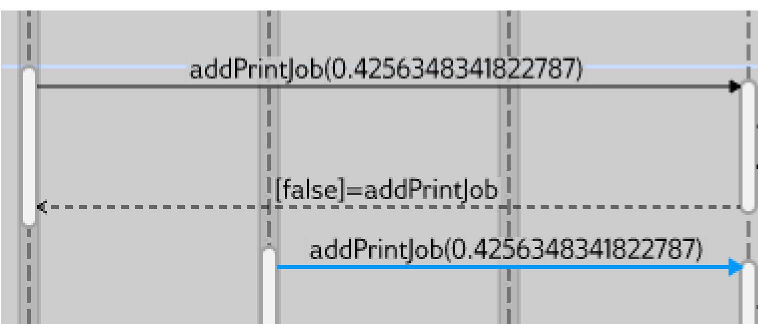
The Sequence Viewer displays function calls and replies. This table lists the type of support for each type of function call.

| Function Call Type | Support |
|---|---|
| Calls to Simulink Function blocks | Fully supported |

| Function Call Type | Support |
| --- | --- |
| Calls to Stateflow graphical or Stateflow MATLAB functions | • Scoped — Select the **Export chart level functions** chart option. Use the *chartName*.*functionName* dot notation.<br>• Global — Select the **Treat exported functions as globally visible** chart option. You do not need the dot notation. |
| Calls to function-call subsystems | Not displayed in the Sequence Viewer window |

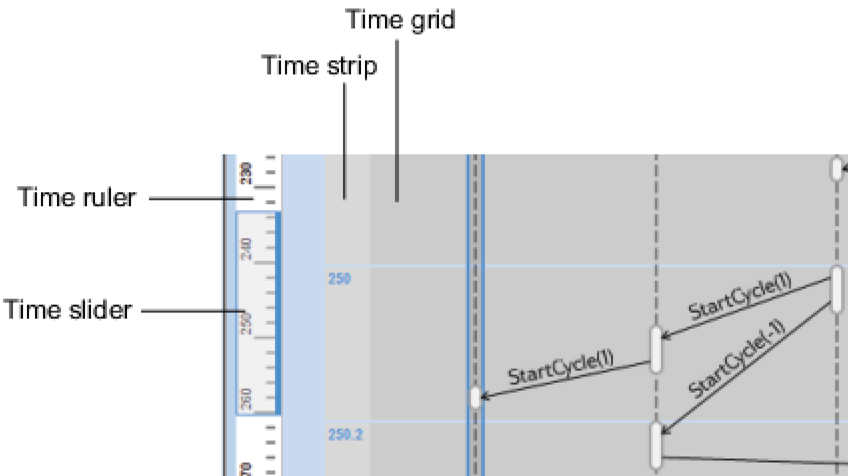
The Sequence Viewer window displays function calls as solid arrows labeled with the format *function_name*(*argument_list*). Replies to function calls are displayed as dashed arrows labeled with the format [*argument_list*]=*function_name*.

For example, in the model `slexPrinterExample`, a subsystem calls the Simulink Function block `addPrinterJob`. The function block replies with an output value of `false`.




## Simulation Time in the Sequence Viewer Window

The Sequence Viewer window shows events vertically, ordered in time. Multiple events in Simulink can happen at the same time. Conversely, there can be long periods of time during simulation with no events. As a consequence, the Sequence Viewer window shows time by using a combination of linear and nonlinear displays. The time ruler shows linear simulation time. The time grid shows time in a nonlinear fashion. Each time grid row, bordered by two blue lines, contains events that occur at the same simulation time. The time strip provides the times of the events in that grid row.



To show events in a specific simulation time range, use the scroll wheel or drag the time slider up and down the time ruler. To navigate to the beginning or end of the simulation, click the **Go to first event** or **Go to last event** buttons. To see the entire simulation duration on the time ruler, click the **Fit to view** button.

When using a variable step solver, you can adjust the precision of the time ruler. In the Model Explorer, on the **Main** tab of the Sequence Viewer Block Parameters pane, adjust the value of the **Time Precision for Variable Step** field.

## Redisplay of Information in the Sequence Viewer Window

The Sequence Viewer saves the order and states of lifelines between simulation runs. When you close and reopen the Sequence Viewer window, it preserves the last open lifeline state. To save a particular viewer state, click the **Save Settings** button in the toolbar. Saving the model then saves that state information across sessions. To load the saved settings, click the **Restore Settings** button.

You can modify the **Time Precision for Variable Step** and **History** parameters only between simulations. You can access the buttons in the toolbar before simulation or when the simulation is paused. During a simulation, the buttons in the toolbar are disabled.

## See Also

Sequence Viewer

## More About

- “Synchronize Model Components by Broadcasting Events” (Stateflow)
- “Communicate with Stateflow Charts by Sending Messages” (Stateflow)
- “Model a Distributed Traffic Control System by Using Messages” (Stateflow)

# Learning More About SimEvents Software


- “Event Calendar” on page 6-3
- “Save SimEvents Simulation Operating Point” on page 6-4
- “Example Model to Count Simultaneous Departures from a Server” on page 6-9
- “Example Model for Noncumulative Entity Count” on page 6-10
- “Adjust Entity Generation Times Through Feedback” on page 6-11
- “A Simple Example of Generating Multiple Entities” on page 6-14
- “A Simple Example of Event-Based Entity Generation” on page 6-15
- “Serve Preferred Customers First” on page 6-16
- “Find and Examine Entities” on page 6-17
- “Extract Found Entities” on page 6-20
- “Trigger Entity Find Block with Event Actions” on page 6-21
- “Build a Firewall and an Email Server” on page 6-22
- “Implement the Custom Entity Storage Block” on page 6-23
- “Implement the Custom Entity Storage Block with Iteration Event” on page 6-24
- “Implement the Custom Entity Storage Block with Two Timer Events” on page 6-25
- “Implement the Custom Entity Generator Block” on page 6-26
- “Implement the Custom Entity Storage Block with Two Storages” on page 6-27
- “Generating and Initializing Entities” on page 6-28
- “M/M/1 Queuing System” on page 6-37
- “M/D/1 Queuing System” on page 6-41
- “G/G/1 Queuing System and Little's Law” on page 6-44
- “Comparing Queuing Strategies” on page 6-47
- “Modeling Hybrid Systems - Tank Filling” on page 6-51
- “Resource Allocation from Multiple Pools” on page 6-55
- “Using Entity Priority to Sequence Departures” on page 6-60
- “Using Custom Visualization for Entities” on page 6-62
- “Selection Server - Select Specific Entities from Server” on page 6-65
- “Flush Entities from a Queue-Server” on page 6-67
- “Server with Pause/Continue” on page 6-70
- “Simulation of a Medical Device” on page 6-72
- “Dining Philosophers Problem” on page 6-77
- “Simulate Scheduler of a Multicore Control System” on page 6-81
- “Develop Custom Scheduler of a Multicore Control System” on page 6-86

- “Distributing Multi-Class Jobs to Service Stations” on page 6-94
- “Effects of Communication Delays on an ABS Control System” on page 6-96
- “Aircraft Boarding Process Flow” on page 6-100
- “Optimization of Shared Resources in a Batch Production Process” on page 6-102
- “Modeling a Kanban Production System” on page 6-112
- “Job Scheduling and Resource Estimation for a Manufacturing Plant” on page 6-122
- “Modeling Load Within a Dynamic Voltage Scaling Application” on page 6-137
- “Modeling Machine Failure” on page 6-140
- “Inventory Management” on page 6-145
- “Modeling Cyber-Physical Systems” on page 6-148
- “802.11 MAC and Application Throughput Measurement” on page 6-153
- “802.11ax System-Level Simulation with Physical Layer Abstraction” on page 6-169

# Event Calendar

During a simulation, the model maintains a list, called the event calendar, of upcoming events that are scheduled for the current simulation time or future times. The event calendar sorts multiple events that are scheduled for the same time by the priority of the entity for which they are scheduled. The model refers to the event calendar to execute events at the correct simulation time and in an appropriately prioritized sequence.

These are the events that the event calendar tracks.

| Event | For Blocks |
| --- | --- |
| Generate | Entity Generator, MATLAB Discrete-Event System |
| Forward | Entity Generator, Entity Queue, Multicast Receive Queue, Entity Server, Entity Terminator, Discrete Event Chart, MATLAB Discrete Event System, Entity Replicator, Resource Acquirer |
| ServiceComplete | Entity Server |
| Timer | MATLAB Discrete-Event System, Discrete Event Chart |
| Iterate | MATLAB Discrete-Event System |
| Destroy | MATLAB Discrete-Event System |

## See Also

Discrete Event Chart | Entity Generator | Entity Queue | Entity Server | MATLAB Discrete Event System | Resource Acquirer

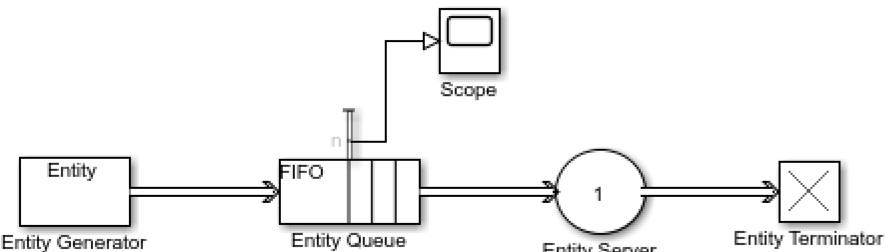
## More About


- “Debug SimEvents Models” on page 12-2
- “Visualization and Animation for Debugging” on page 5-10
- “Observe Entities Using simevents.SimulationObserver Class” on page 10-5
- “Use SimulationObserver Class to Monitor a SimEvents Model” on page 10-2

# Save SimEvents Simulation Operating Point

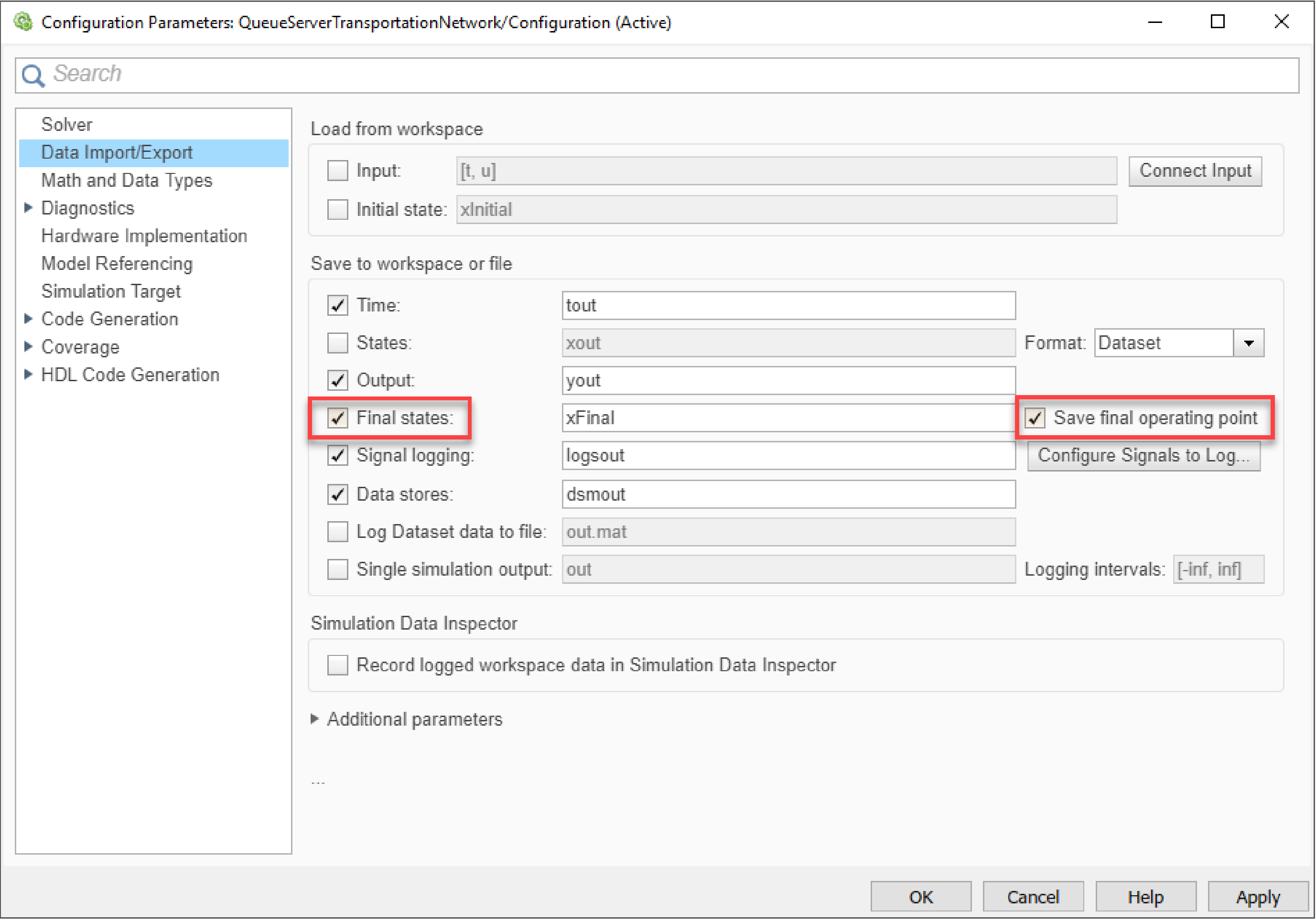
This example shows how to save and restore the simulation state of a SimEvents model by using **Save final operating point** check box and use it as an initial state for future simulations. For more information about using **Save final operating point**, see "Save and Restore Simulation Operating Point".

The **Save final operating point** check box is used to save the state of a simple queuing system with an Entity Generator block, an Entity Queue block, an Entity Server block, and an Entity Terminator block. The signal output port **n** displaying the number of entities departed the Entity Queue block is connected to a Scope block. For more information about performing basic tasks to create this model, see "Create a Discrete-Event Model". The only difference in the model is the placement of the scope.



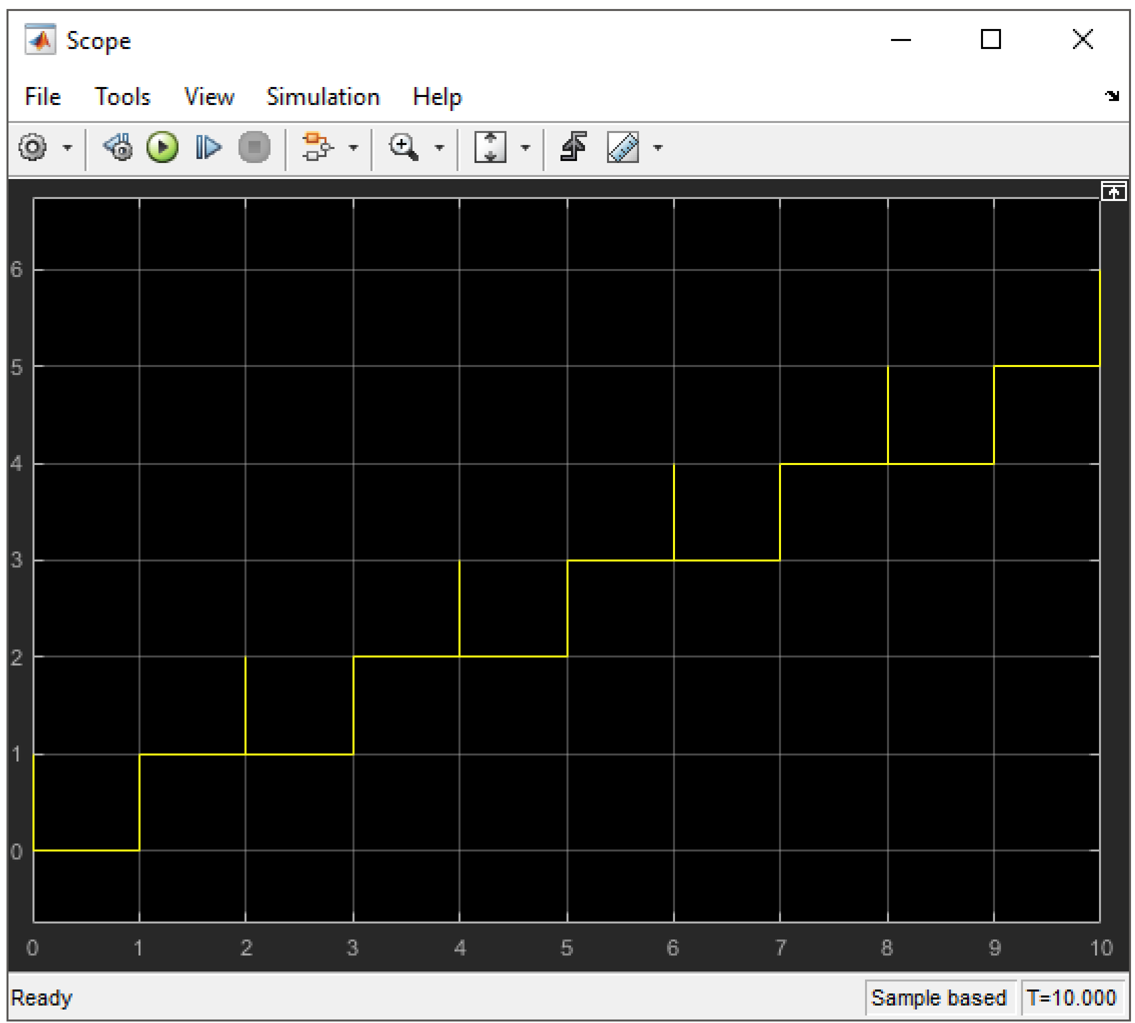

1. Open the Entity Server Block Parameters dialog box. Set the **Service time value** to 2.

   The queue length increases throughout the simulation because service time is larger than the entity intergeneration time.
2. From the Simulink Toolstrip, select **Modeling** tab and **Model Settings**. In the Configuration Parameters dialog box, in the **Data Import/Export** pane, select the **Final states** check box with the variable name `xFinal` and select the **Save final operating point** check box.



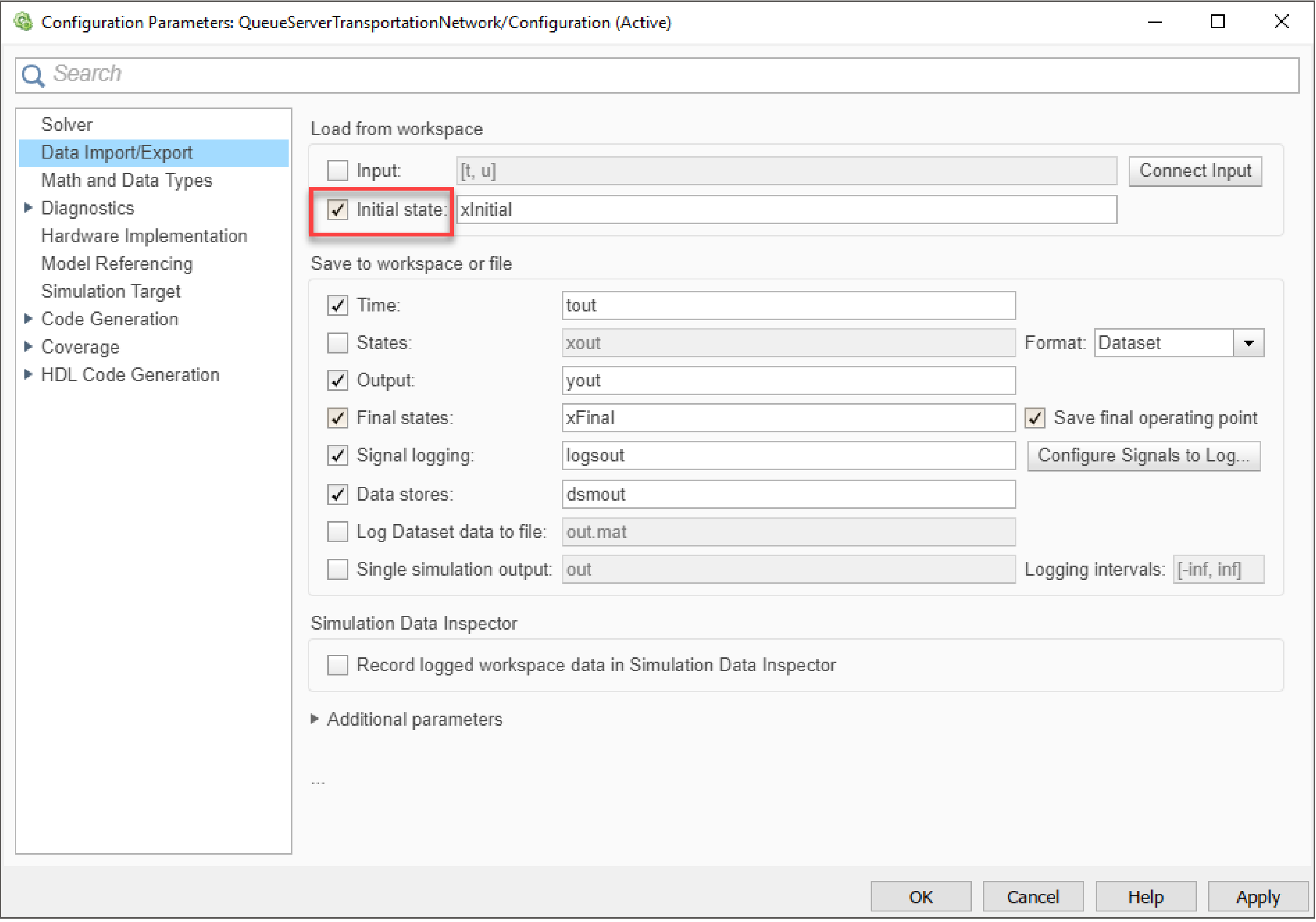

**3** Simulate the model and open the Scope block. Observe that the final queue length is 6.

The queue length increases, with spikes at times 2, 4, 6, 8, and 10 because the **Service time value** of the Entity Server block is 2. The entity in the Entity Server block departs, and the entity that arrives at the Entity Queue block immediately advances to the Entity Server block.




**4** In the Configuration Parameters dialog box, select the **Initial state** check box and specify the variable name as `xFinal`.

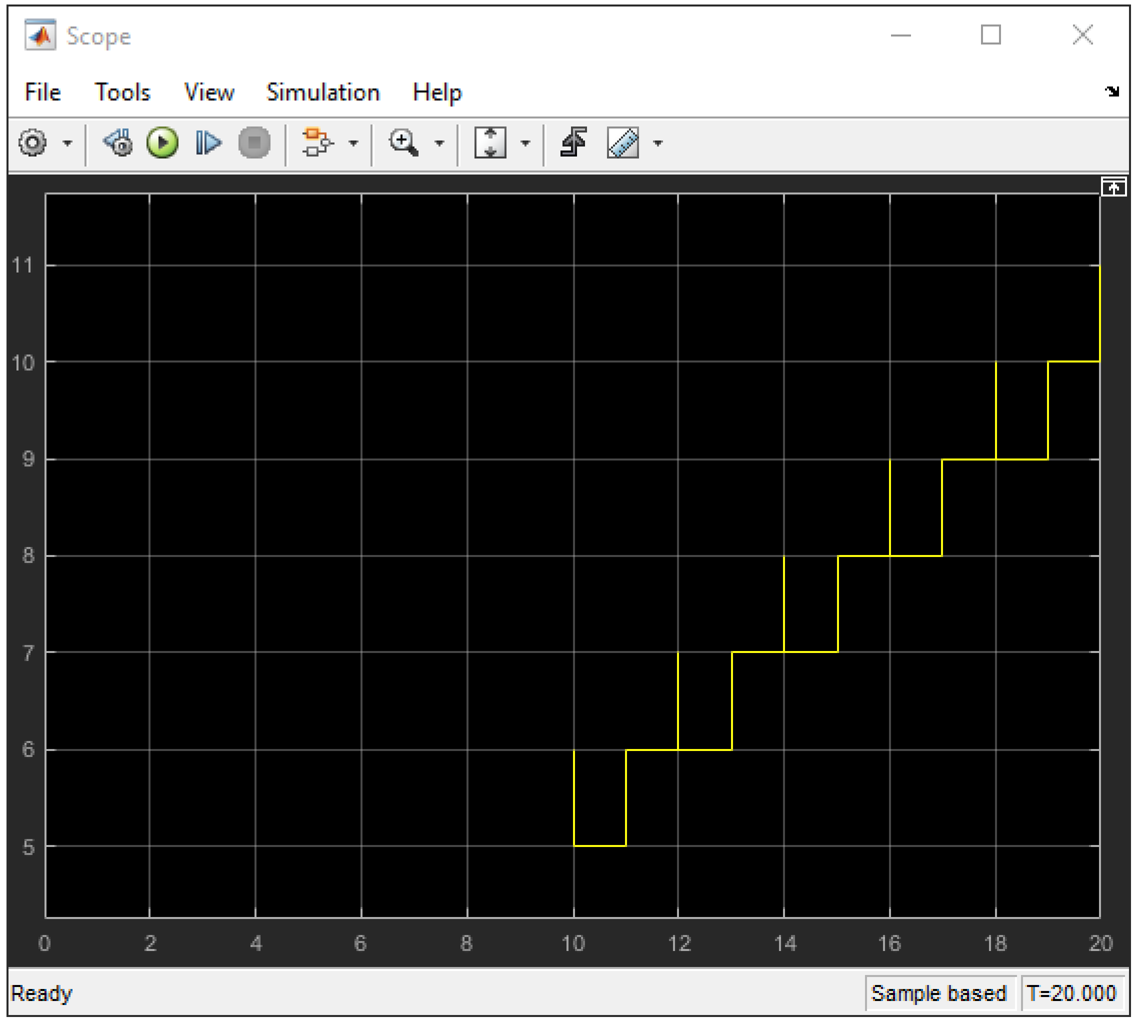
`xFinal` is used as an initial state for the next simulation.




**5** Increase the simulation time to `20`.

Set the simulation time larger than `10` to observe simulation with the saved initial simulation state.

**6** Simulate the model. Open the Scope block. Observe that the simulation starts from the queue length `6`, which is the final state of the previous simulation.




## See Also

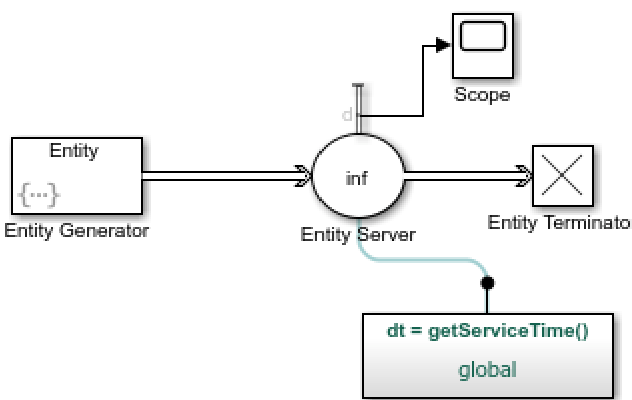
Entity Generator | Entity Queue | Entity Server | Entity Terminator

## Related Examples

- “Solvers for Discrete-Event Systems” on page 7-5
- “Debug SimEvents Models” on page 12-2
- “Manage Entities Using Event Actions”

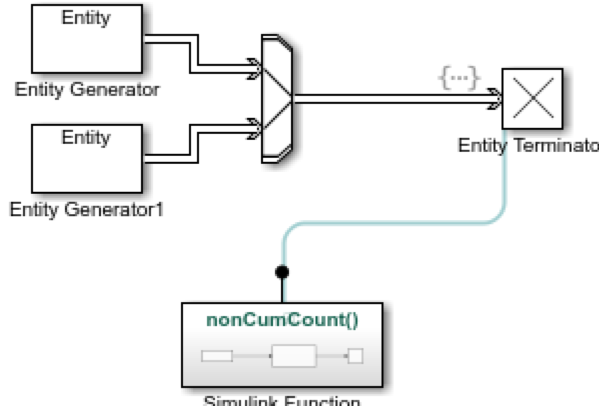
# Example Model to Count Simultaneous Departures from a Server

This example shows how to count the simultaneous departures of entities from a server. Use the **Number of entities departed, d** statistic from the Entity Server block to learn how many entities have departed the block. The output signal also indicates when departures occurred. This method of counting is cumulative throughout the simulation.



# Example Model for Noncumulative Entity Count

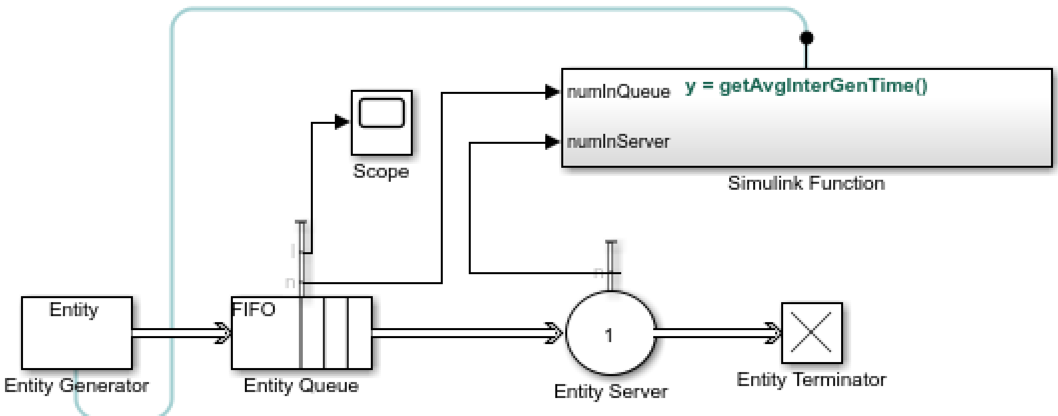
This example shows how to count entities, which arrive to an Entity Terminator block, in a noncumulative way by resetting the counter at each time instant.



# Adjust Entity Generation Times Through Feedback

This example shows a queuing system in which feedback influences the arrival rate. The goal of the feedback loop is to stabilize the entity queue by slowing the entity generation rate of the Entity Generator block as more entities accumulate in the Entity Queue block and the Entity Server block.

The diagram shows a simple queuing system with an Entity Generator, an Entity Queue, an Entity Server, and an Entity Terminator block. For more information about building this simple queuing system, see “Create a Discrete-Event Model”.




The capacity of the Entity Server block is `1`. This causes an increase in the queue length without feedback. The goal is to regulate entity intergeneration time based on the size of the queue and the number of entities waiting to be served.

- In the Entity Generator block, select `MATLAB action` as the **Time source**. Add this code to the **Intergeneration time action** field.

```
persistent rngInit;

if isempty(rngInit)
    seed = 12345;
    rng(seed);
    rngInit = true;
end

% Pattern: Exponential distribution
mu = getAvgInterGenTime();
dt = -mu*log(1-rand());
```

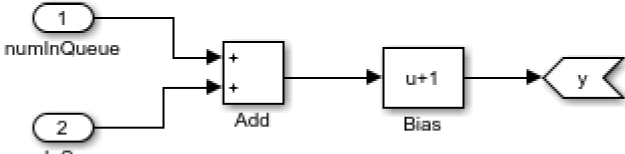
The entity intergeneration time `dt` is generated from an exponential distribution with mean `mu`, which is determined by the function `getAvgInterGenTime()`.

- In the Entity Queue block, in the **Statistics tab**, select the **Number of entities in block, n** and **Average queue length, l** as output statistics.

- In the Entity Server block, select `MATLAB action` as the **Service time source**. Add this code to the **Service time action** field.

```
persistent rngInit;
if isempty(rngInit)
    seed = 67868;
    rng(seed);
    rngInit = true;
end

% Pattern: Exponential distribution
mu = 3;
dt = -mu*log(1-rand());

The service time |dt| is drawn from an exponential distribution with
mean |3|.
```

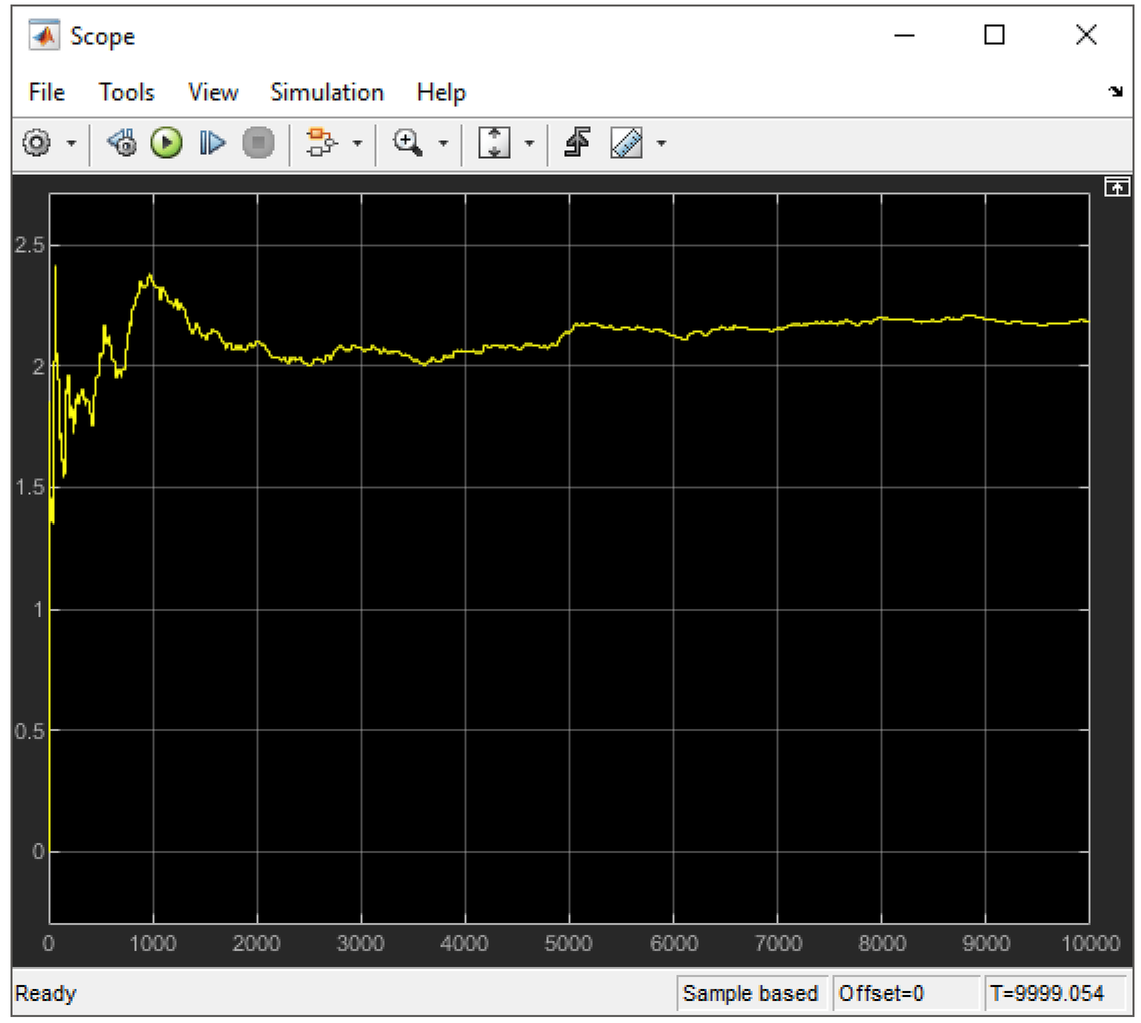
- In the Entity Server block, in the **Statistics tab**, select the **Number of entities in block, n** as output statistics.
- Add a Simulink Function block. On the Simulink Function block, double-click the function signature and enter `y = getAvgInterGenTime()`.
- In the Simulink Function block:




1. Add two In1 blocks and rename them as numInQueue and numInServer.
2. numInQueue represents the current number of entities accumulated in the queue and numInServer represents the current number of entities accumulated in the server.
3. Use Add block to add these two inputs.
4. Use a Bias block and set the `Bias` parameter as `1`. The constant bias `1` is to guarantee a nonzero intergeneration time.

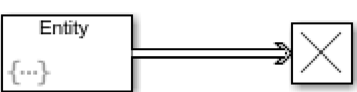
Optionally, select **Function Connections** from the **Information Overlays** under the **Debug** tab to display the feedback loop from the Simulink Function block to the Entity Generation block.

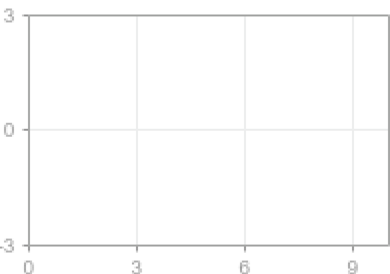
- In the parent model, connect the **Number of entities in block, n** statistics from the Entity Queue and Entity Server blocks to the Simulink Function block.
- Connect a Scope block to the **Average queue length, l** statistic from the Entity Queue block. The goal is to investigate the average queue length.
- Increase the simulation time to `10000` and simulate the model.
- Observe that the **Average queue length, l** in the scope is nonincreasing due to the effect of feedback for the discouraged entity generation rate.


Scope
File
Tools
View
Simulation
Help
2.5
2
1.5
1
0.5
0
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
Ready
Sample based
Offset=0
T=9999.054

# A Simple Example of Generating Multiple Entities

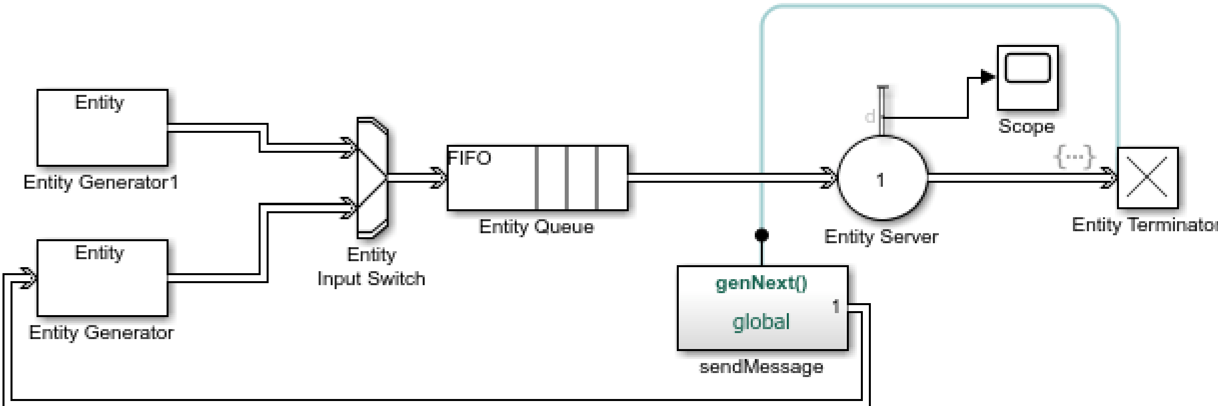
In this example, you can simultaneously generate multiple entities at the start of the simulation. You can then observe the behavior of the model from the output of the Dashboard Scope block.



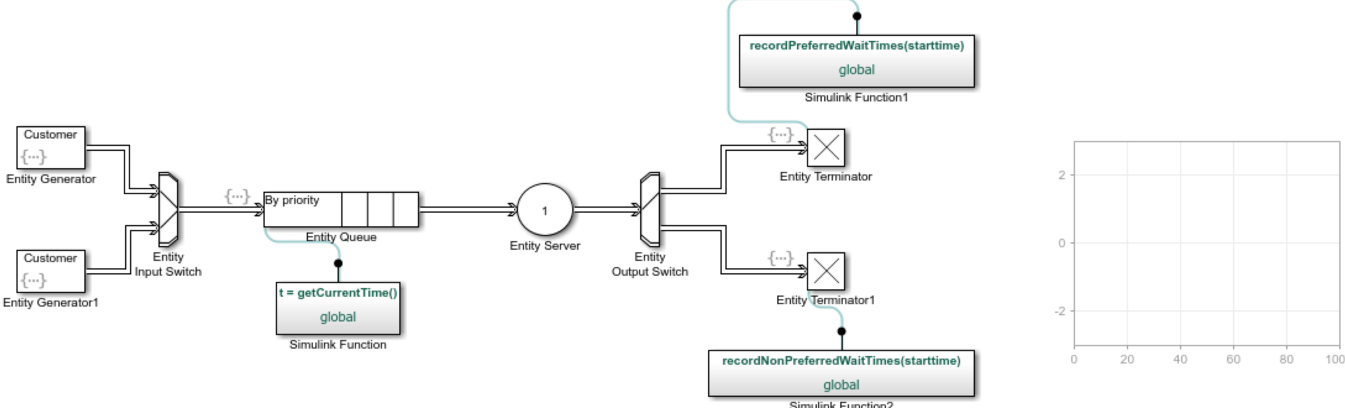
# A Simple Example of Event-Based Entity Generation

In this example, you generate entities based on the message arrival to the Entity Generator block.



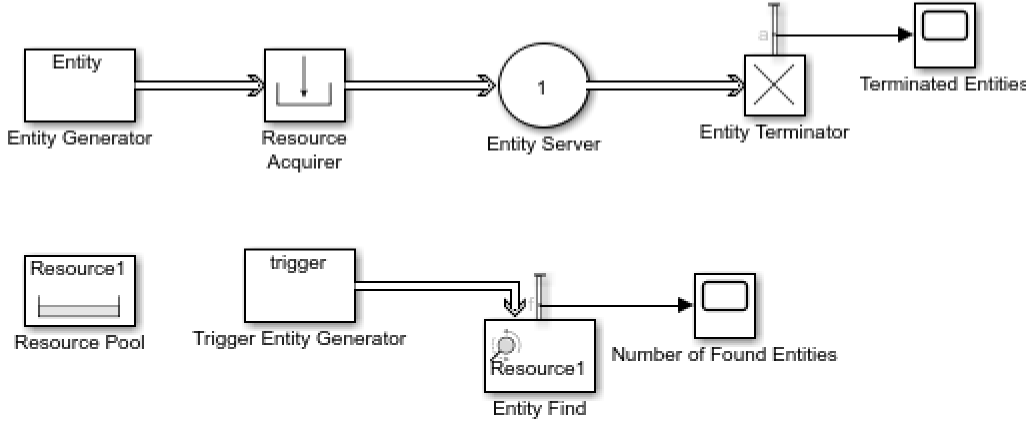
# Serve Preferred Customers First

In this example, two types of customers enter a queuing system. One type, considered to be preferred customers, are less common but require longer service. The priority queue places preferred customers ahead of nonpreferred customers. The model plots the average system time for the set of preferred customers and separately for the set of nonpreferred customers in a Dashboard Scope block.



# Find and Examine Entities

This example shows how to use the Entity Find block to find and examine entities at their location. In this example, the block finds entities that are tagged with a resource from the Resource Pool block.

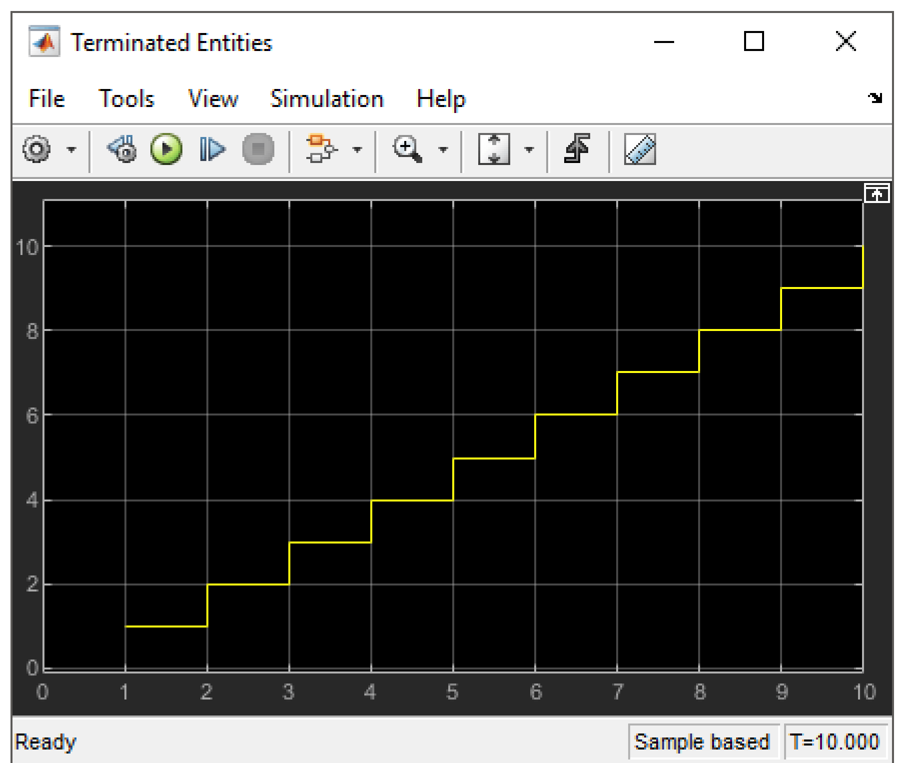



### Model Description

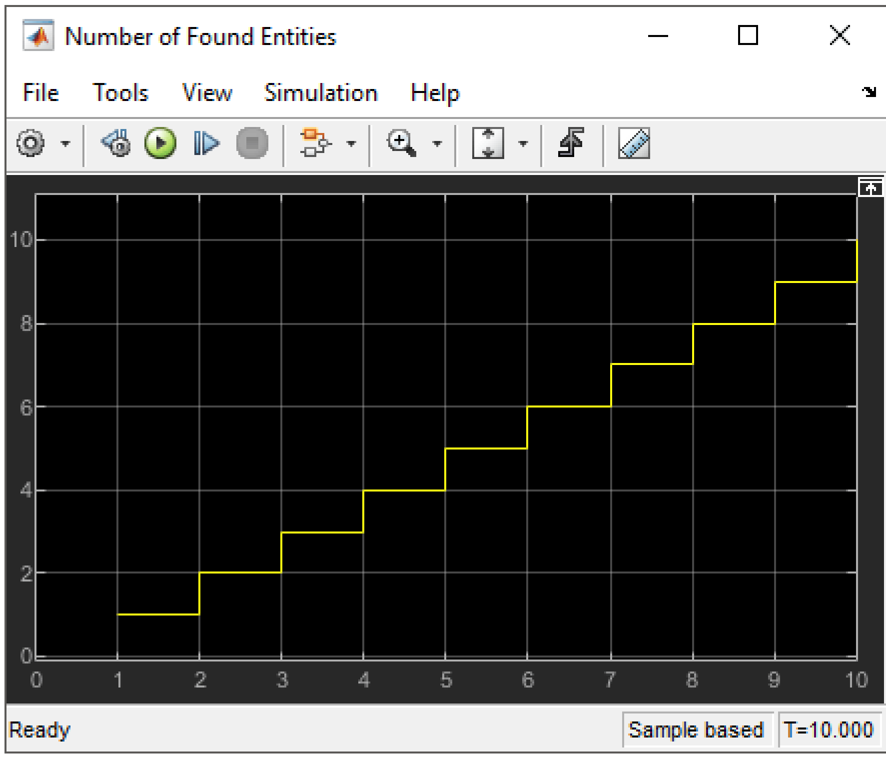
- The top model represents the flow of entities that acquire a `Resource1` resource.
- By default, the Entity Find block finds the entities having `Resource1` tag.
- Every time the Trigger Entity Generator generates an entity, the Entity Find block is triggered to find entities.

### Simulation Results

Simulate the model and observe the Scope blocks labeled as Terminated Entities and Number of Found Entities. The number of terminated entities is `10`.




The number of found entities by the Entity Find block is also `10`. This is because every generated entity acquires a `Resource1` tag and no entities are blocked in the model.



You can also modify and extract found entities. For more information, see “Find and Extract Entities in SimEvents Models” on page 4-10.

## See Also

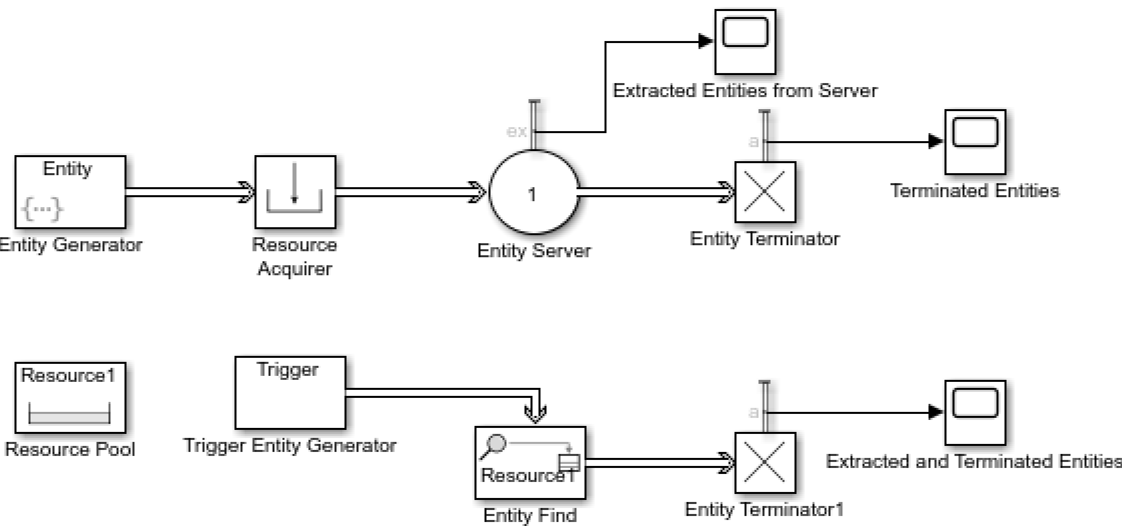
Entity Find | Resource Acquirer | Resource Pool

## More About

- “Find and Extract Entities in SimEvents Models” on page 4-10

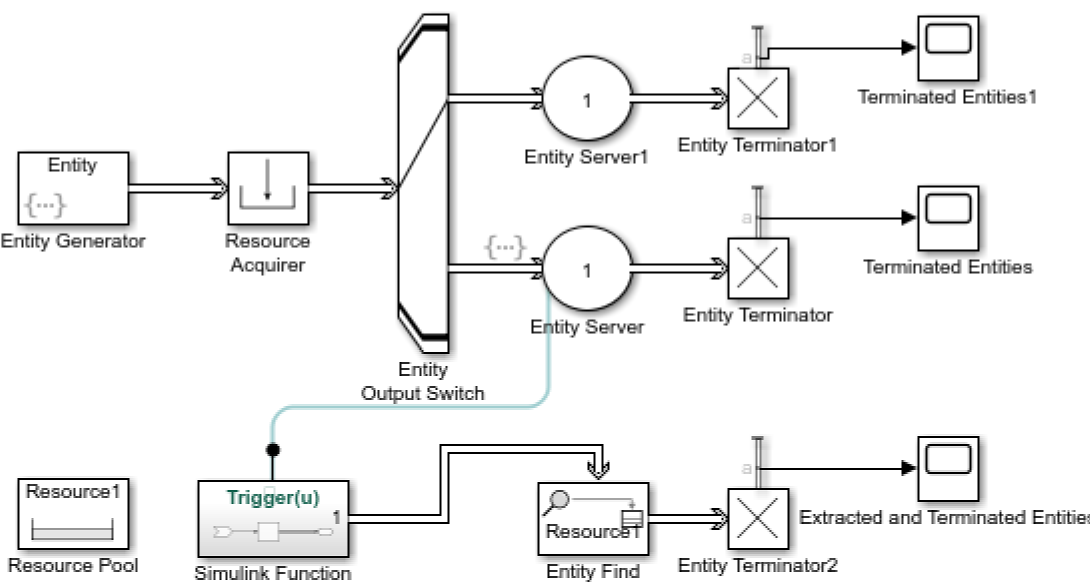
# Extract Found Entities

You can use the Entity Find block to find entities and extract them from their location to reroute. In this example, 3 entities found in the previous example are extracted from the system to be terminated.



# Trigger Entity Find Block with Event Actions

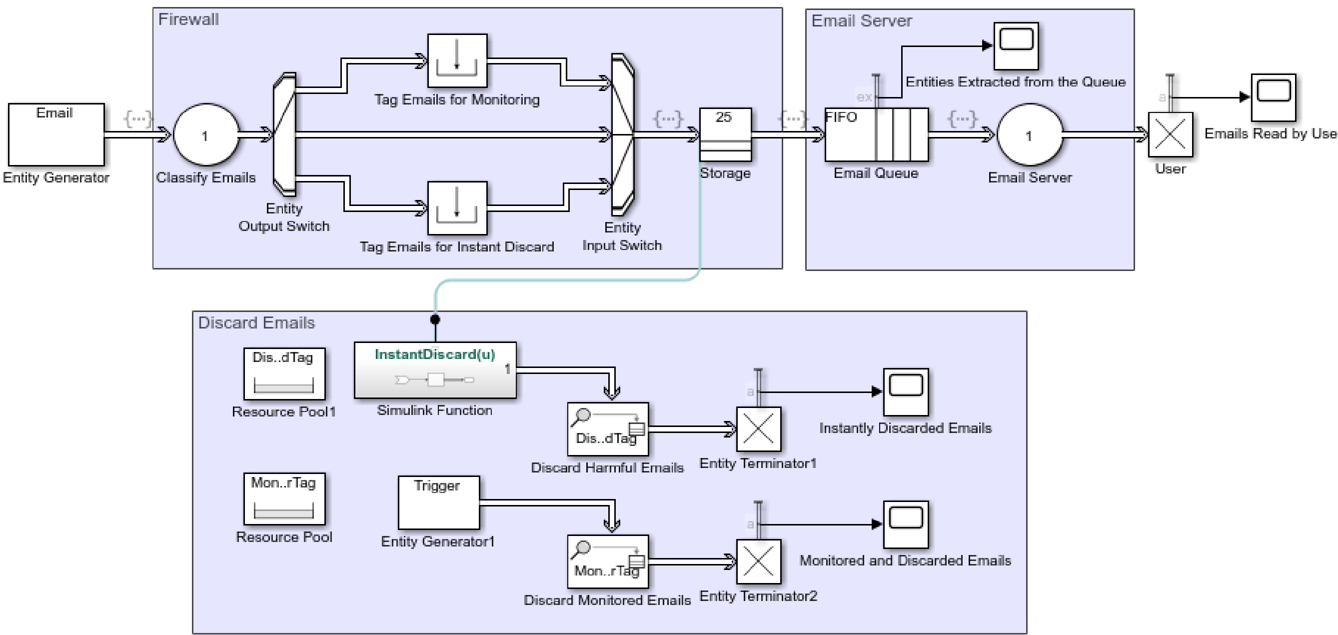
You can trigger the Entity Find block with event actions. In this example, the Entity Find block is triggered when an entity enters the Entity Server block. Modify the previous example by removing the Trigger Entity Generator and by adding the Entity Output Switch, Entity Server1, Entity Terminator2 and Scope blocks to the model and connect them as shown.



# Build a Firewall and an Email Server

You can use the Entity Find block to monitor multiple blocks in a model, to examine, and to extract entities.

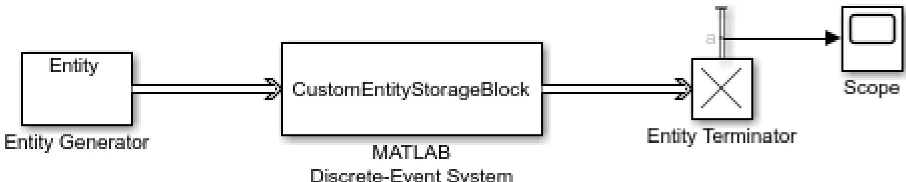
This example represents an email server with a firewall to track, monitor, and discard harmful emails before they reach the user. In the model, emails are generated using an Entity Generator block. In the Firewall component, all emails are classified as harmful for instant discarding, suspicious for monitoring, or safe based on their source. Harmful emails are tagged with a `DiscardTag` resource from the Resource Pool block and instantly discarded from the system. Suspicious emails are tagged with `MonitorTag` and tracked throughout the system for suspicious activity. If a suspicious activity is detected, the email is discarded before it reaches the user. Safe emails are not monitored or discarded.



Copyright 2019 The MathWorks, Inc.

# Implement the Custom Entity Storage Block

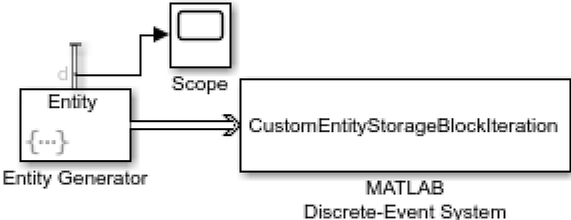
This example shows how to implement a discrete-event System Object™ using a MATLAB Discrete-Event system block. The model also includes an Entity Generator block and an Entity Terminator block. The custom block accepts entities and forwards them with a delay of 4.




Copyright 2019 The MathWorks, Inc.

# Implement the Custom Entity Storage Block with Iteration Event

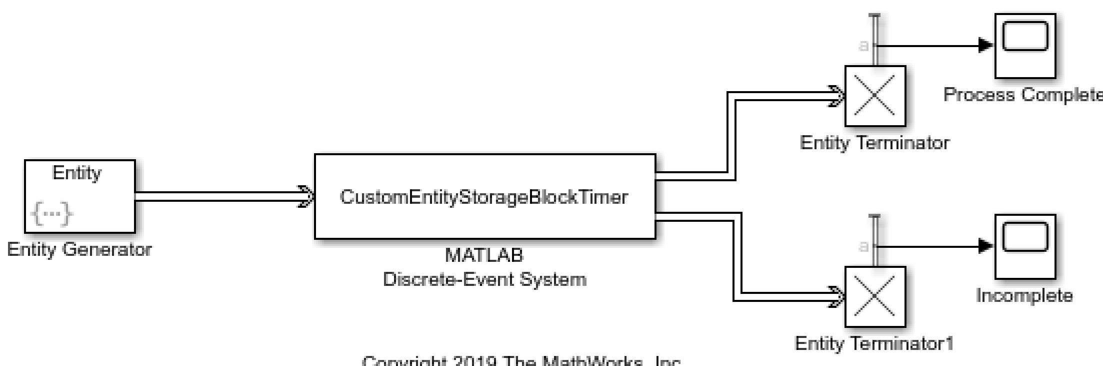
This example shows how to implement a discrete-event System Object™ which represents a custom entity storage block with an iteration event. The model also includes an Entity Generator block that generates Wheels with various Diameter values.




Copyright 2019 The MathWorks, Inc.

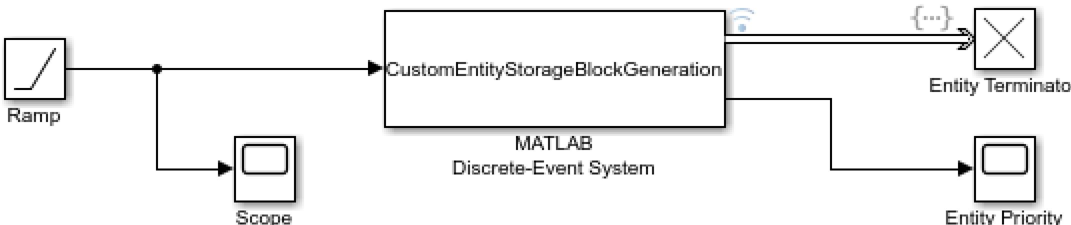
# Implement the Custom Entity Storage Block with Two Timer Events

This example shows how to implement a discrete-event System Object™ which represents a custom entity storage block with two timer events. The model also includes an Entity Generator block that generates entities and two Entity Terminator blocks.



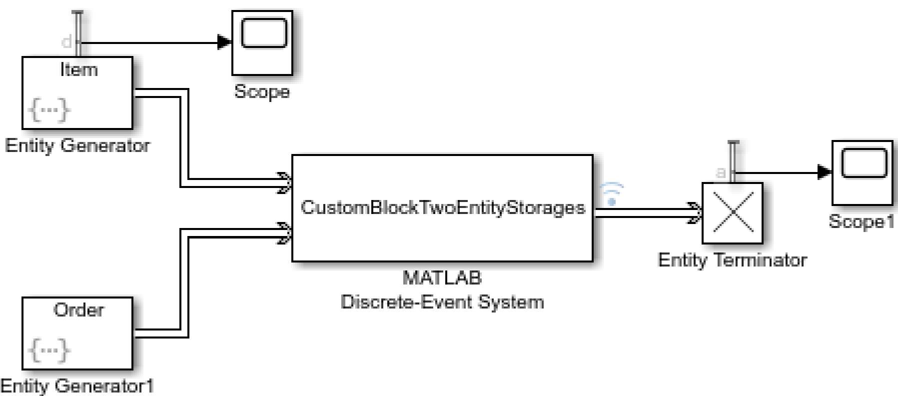
# Implement the Custom Entity Generator Block

This example shows how to implement a discrete-event System Object™ which represents a custom entity generator block. The custom generator block also assigns priority values and data to each generated entity. The priority values are acquired from the incoming signal from the Ramp block. The model also includes an Entity Terminator block.



# Implement the Custom Entity Storage Block with Two Storages

This example shows how to implement a discrete-event System Object™ which represents a custom entity storage block with two storages. The model also includes two Entity Generator blocks to generate two types of entities and an Entity Terminator block.



# Generating and Initializing Entities

### Description

This example shows different ways to generate and initialize entities and their attribute values.

**Generate and Initialize Entities** ?

Periodic Entity Generation
Initialize entities using MATLAB code

Burst Entity Generation
Preload a Queue at the Start of Simulation

Exponential Entity Arrival Times
Initialize entities using Simulink Functions

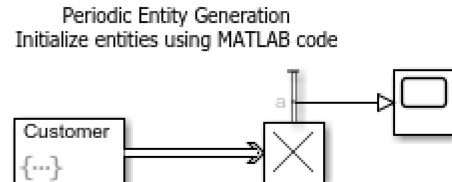
Event-Based Entity Generation

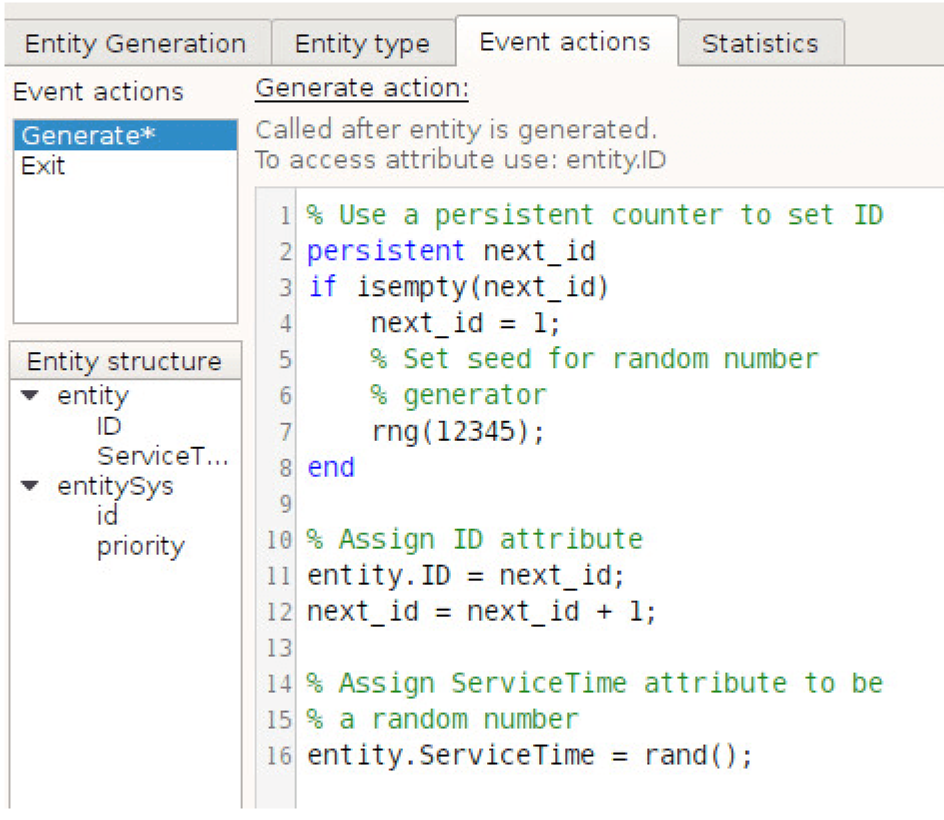
Inter-Arrival Times from Sequence
Initialize entities using Spreadsheet

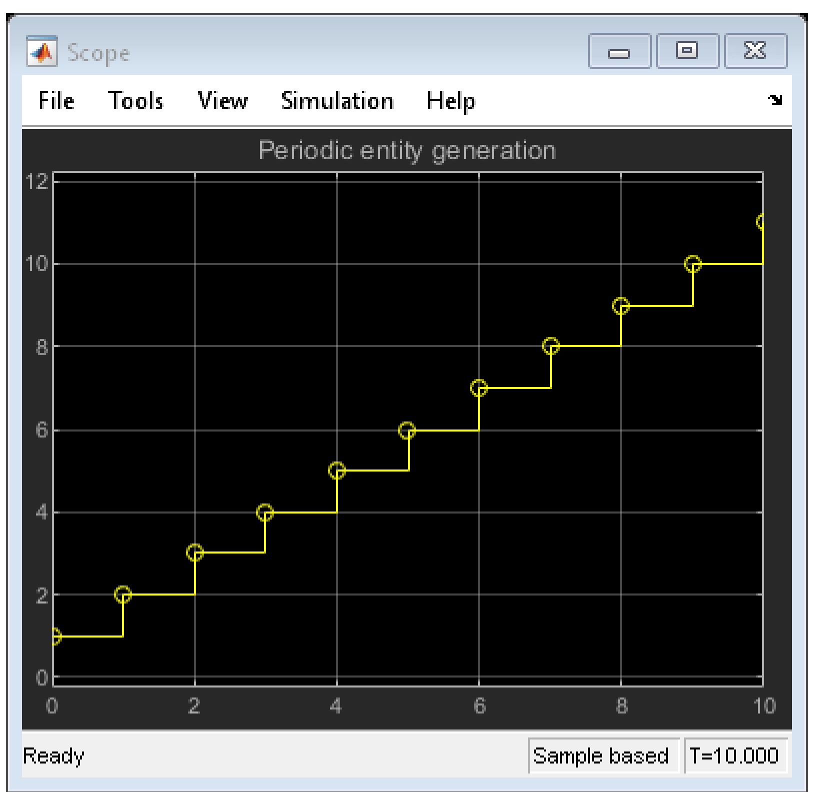
Copyright 2016 The MathWorks, Inc.

### Periodic Entity Generation and Initialize with MATLAB Code

Generate entities periodically by setting a constant value of intergeneration time in the Entity Generator. You can then initialize these entities in the Entity Generator using MATLAB code as shown below.

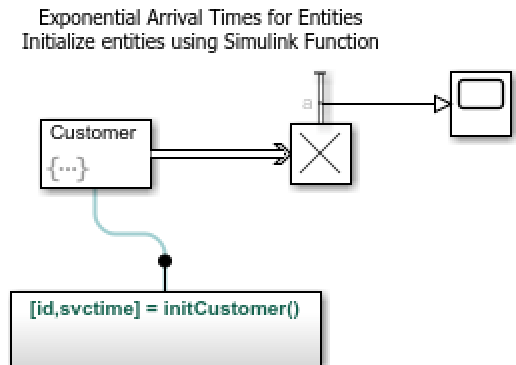


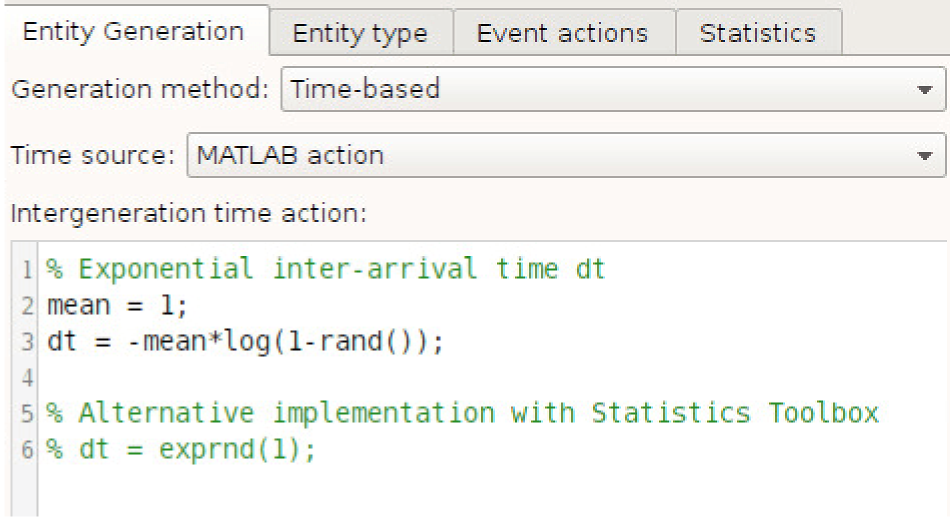

Entity Generation
Entity type
Event actions
Statistics
Event actions
Generate*
Exit
Entity structure
entity
ID
ServiceT...
entitySys
id
priority
Generate action:
Called after entity is generated.
To access attribute use: entity.ID
% Use a persistent counter to set ID
persistent next_id
if isempty(next_id)
next_id = 1;
% Set seed for random number
% generator
rng(12345);
end
% Assign ID attribute
entity.ID = next_id;
next_id = next_id + 1;
% Assign ServiceTime attribute to be
% a random number
entity.ServiceTime = rand();


Scope
File
Tools
View
Simulation
Help
Periodic entity generation
Ready
Sample based
T=10.000

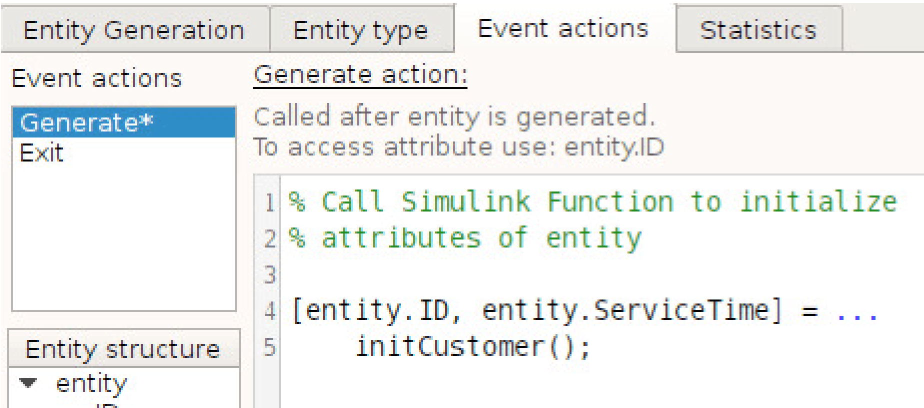
### Randomized Entity Generation and Initialize with Simulink Functions

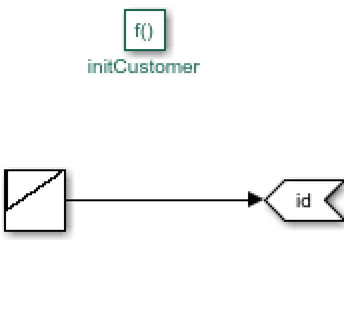
Generate entities using intergeneration time sampled from a random distribution by writing a custom intergeneration time action in the Entity Generator block, as shown below.

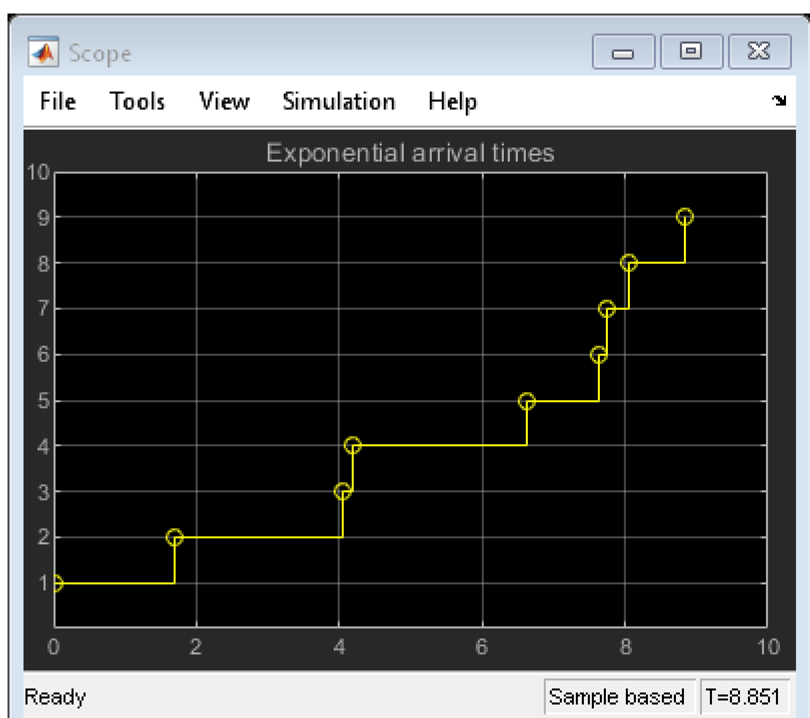



In this example, initialize the generated entities by calling into a Simulink Function that returns initial values for attributes as shown below.



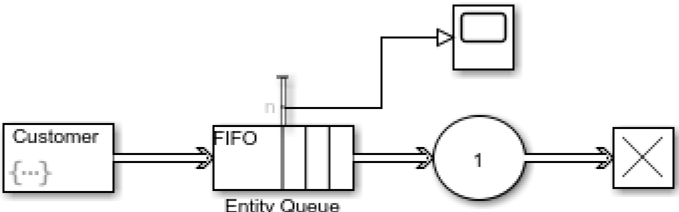



**Burst Entity Generation - Generating Multiple Entities Simultaneously**

This example shows how you can generate multiple entities simultaneously at the start of simulation to preload a queue. To generate N entities, the intergeneration time for these entities must be 0 (zero). To stop generation after N entities, set the intergeneration time to infinity (inf).

Burst Entity Generation
Generate multiple entities at the start of simulation

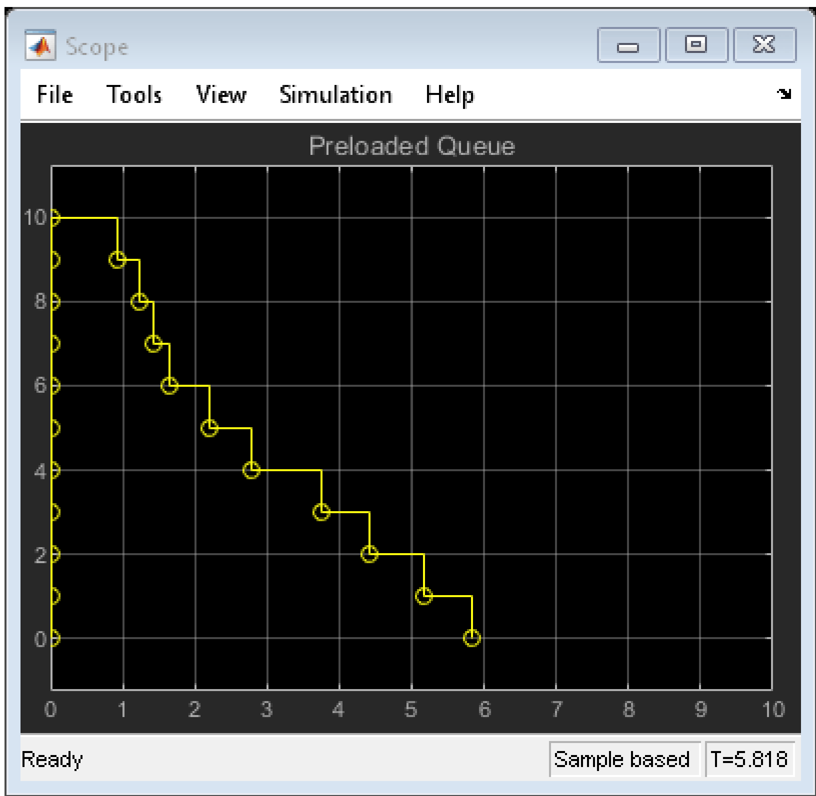



Generation method: Time-based

Time source: MATLAB action

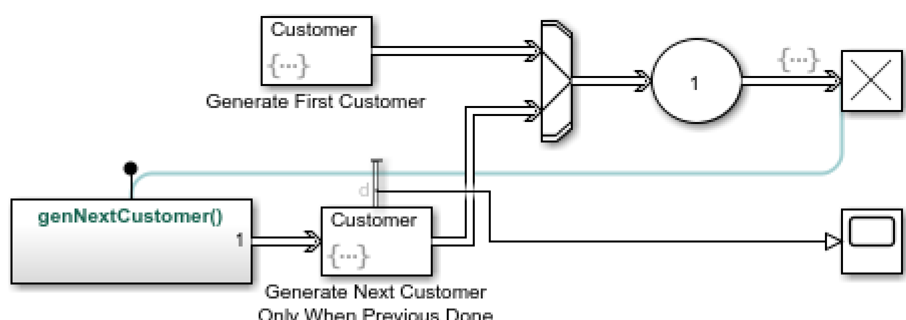
Intergeneration time action:

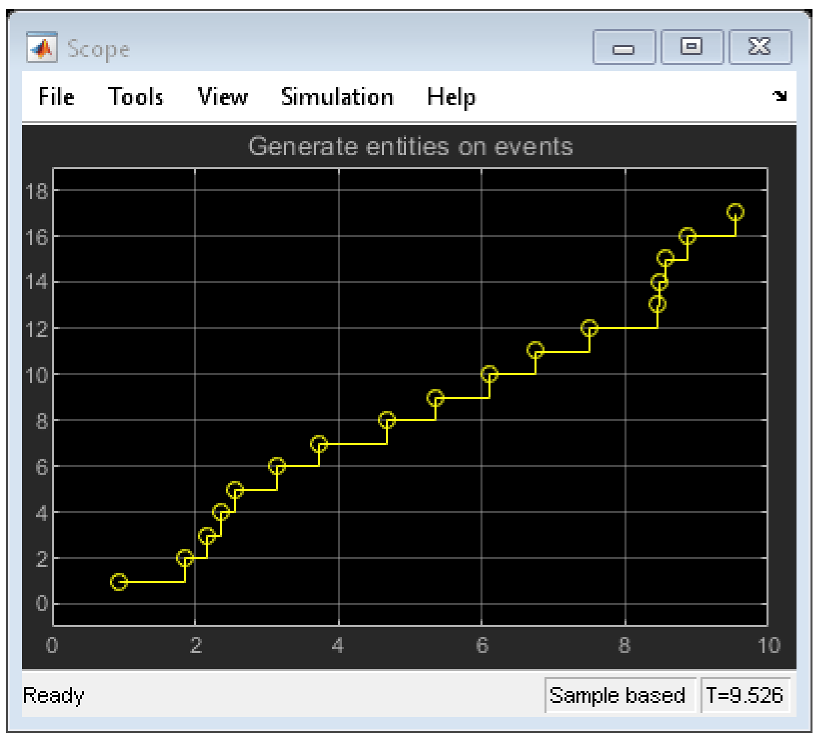
```
1  % Generate N entities at time 0
2  N = 10;
3
4  persistent dtArray index
5  if isempty(dtArray)
6      dtArray = [zeros(1,N) inf];
7      index = 1;
8  end
9  dt = dtArray(index);
10 index = index + 1;
```




### Event-Based Entity Generation

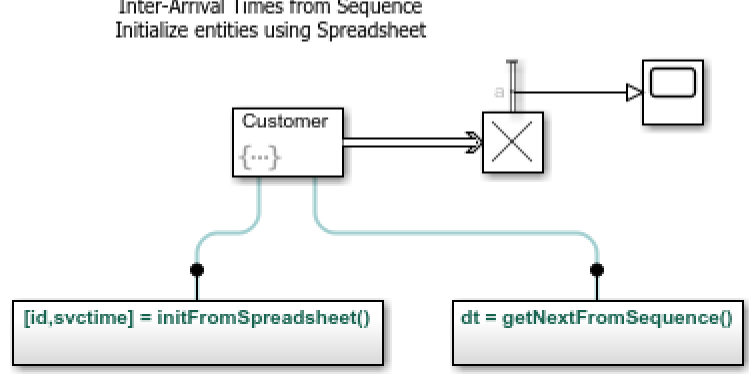
This example shows how you can generate entities when certain events occur in the model. Each such event can be translated into a message arrival at the event input port of the Entity Generator.



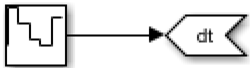



**Inter-Arrival Times from Sequence and Initialize from Spreadsheet**

This example shows how you can generate entities where the intergeneration times are specified from a sequence or an array. Use a Repeating Sequence Stair block inside the Simulink Function

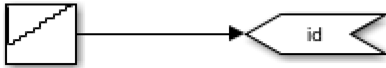


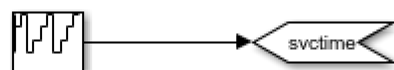

Simulink function: `getNextFromSequence`



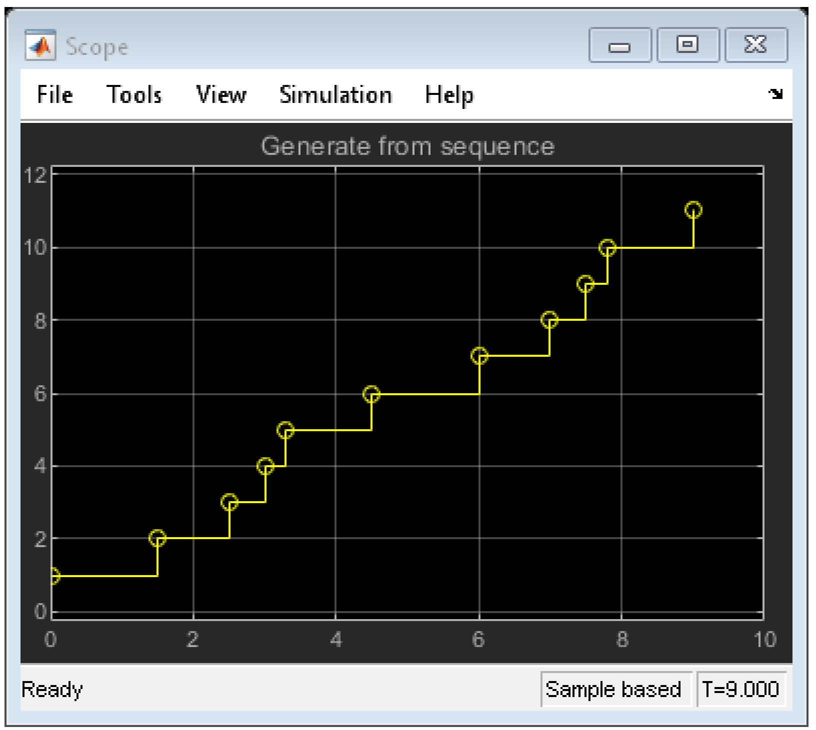

Initialize the generated entities using data from a spreadsheet by importing the data to a MATLAB table object. Individual columns of this table can then be read by the Repeating Sequence Stair block in the Simulink Function `initFromSpreadsheet`.

Reading data from a MATLAB table created from a spreadsheet




```
On model load we read from a spreadsheet as:
>> tableData = readtable('seExampleSpreadsheetData.xlsx')

To access the "id" column of the table as an array, use:
>> tableData{:, 'id'}
```




## See Also

Entity Server | Queue | Entity Terminator | Entity Generator

## Related Examples

- "Create a Discrete-Event Model"
- "Explore Statistics and Visualize Simulation Results"
- "Manage Entities Using Event Actions"

# M/M/1 Queuing System

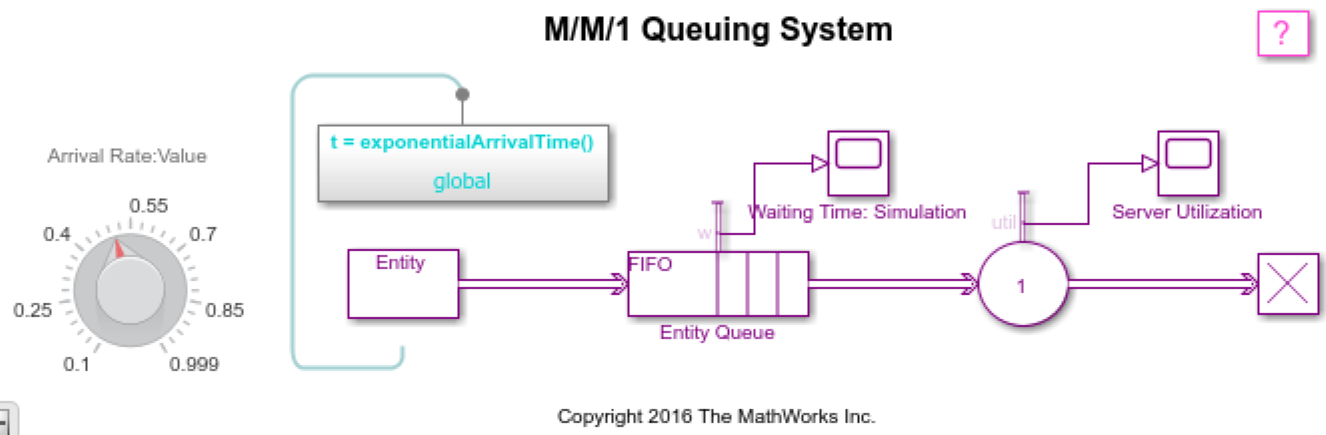
### Overview

This example shows how to model a single-queue single-server system with a single traffic source and an infinite storage capacity. In the notation, the M stands for Markovian; M/M/1 means that the system has a Poisson arrival process, an exponential service time distribution, and one server. Queuing theory provides exact theoretical results for some performance measures of an M/M/1 queuing system and this model makes it easy to compare empirical results with the corresponding theoretical results.

### Structure

The model includes the components listed below:

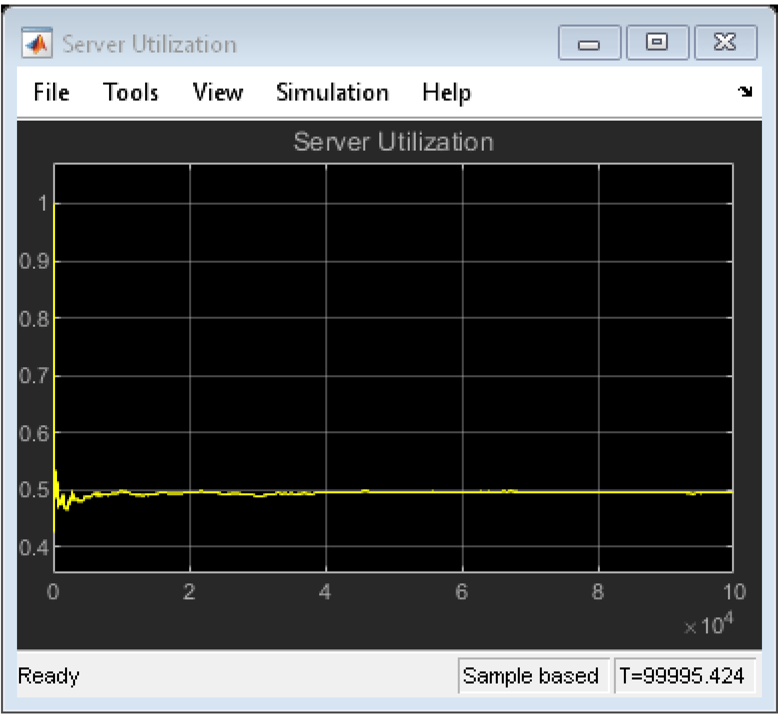
- **Entity Generator block:** Models a Poisson arrival process by generating entities (also known as "customers" in queuing theory).
- **Simulink Function exponentialArrivalTime():** Returns data representing the interarrival times for the generated entities. The interarrival time of a Poisson arrival process is an exponential random variable.
- **Entity Queue block:** Stores entities that have yet to be served in FIFO order
- **Entity Server block:** Models a server whose service time has an exponential distribution.

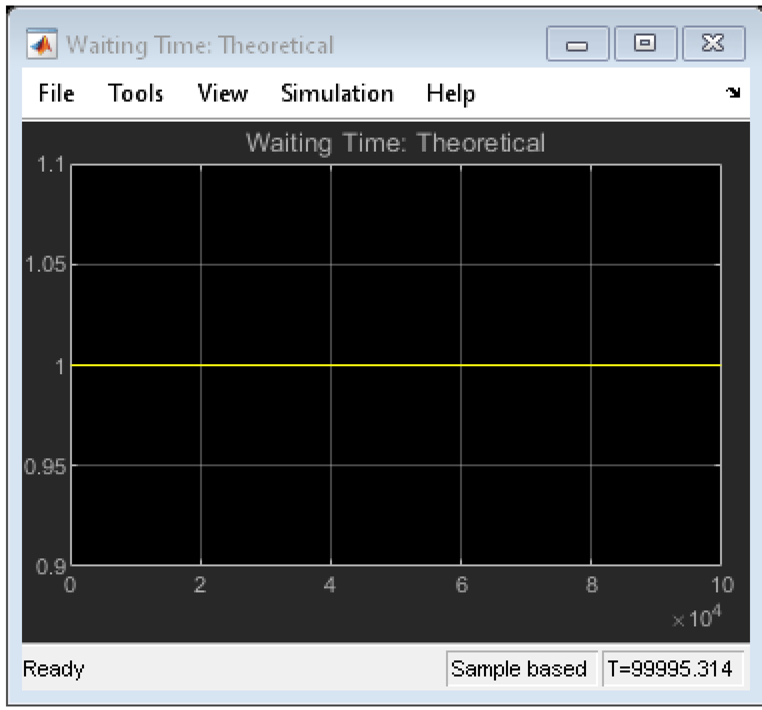


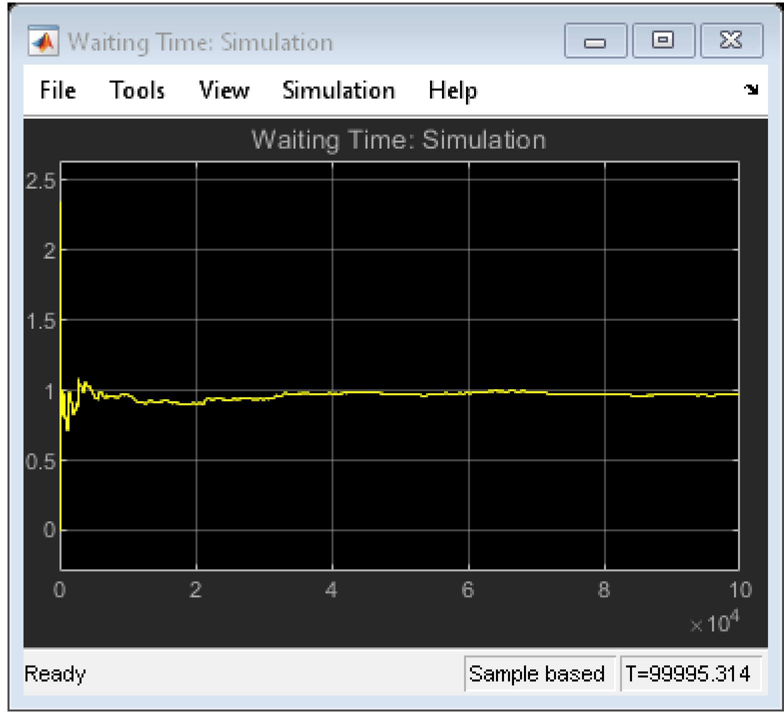

### Results and Displays

The model includes these visual ways to understand its performance:

- Scopes labeled "Waiting Time: Theoretical" and "Waiting Time: Simulation" showing the theoretical and empirical values of the waiting time in the queue, on a single set of axes. You can use this plot to see how the empirical values evolve during the simulation and compare them with the theoretical value.
- A scope labeled "Server Utilization" showing the utilization of the single server over the course of the simulation.


Server Utilization
File
Tools
View
Simulation
Help
Server Utilization
1
0.9
0.8
0.7
0.6
0.5
0.4
0
2
4
6
8
10
× 10^4
Ready
Sample based
T=99995.424


Waiting Time: Theoretical
File
Tools
View
Simulation
Help
Waiting Time: Theoretical
1.1
1.05
1
0.95
0.9
0
2
4
6
8
10
× 10^4
Ready
Sample based
T=99995.314




#### Theoretical Results

Queuing theory provides the following theoretical results for an M/M/1 queue with an arrival rate of $\lambda$ and a service rate of $\mu$:

- Mean waiting time in the queue = $1/(\mu - \lambda) - 1/\mu$

The first term is the mean total waiting time in the combined queue-server system and the second term is the mean service time.

- Utilization of the server = $\lambda/\mu$

#### Experimenting with the Model

Move the Arrival Rate knob during the simulation and observe the change in the simulation results

#### Related Examples

- M/D/1 Queuing System
- G/G/1 Queuing System and Little's Law

#### References

[1] Kleinrock, Leonard, Queueing Systems, Volume I: Theory, New York, Wiley, 1975.

## See Also

Entity Server | Queue | Entity Terminator | Entity Generator

## Related Examples

- “Create a Discrete-Event Model”
- “Manage Entities Using Event Actions”
- “Explore Statistics and Visualize Simulation Results”

# M/D/1 Queuing System

### Overview

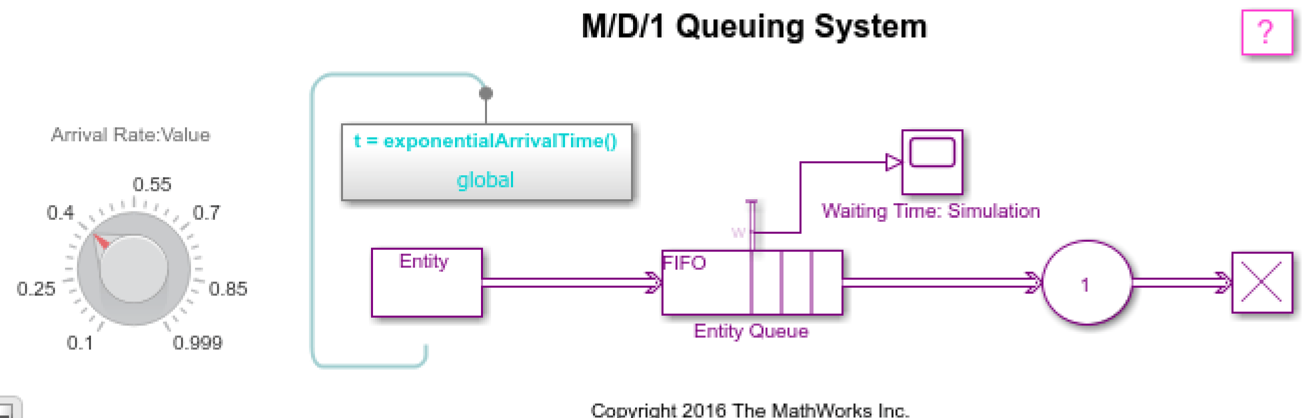
This example shows how to model a single-queue single-server system that has a Poisson arrival process and a server with constant service time. The queue has an infinite storage capacity. In the notation, the M stands for Markovian; M/D/1 means that the system has a Poisson arrival process, a deterministic service time distribution, and one server.

### Structure

The model includes the components listed below:

- **Entity Generator block:** Models a Poisson arrival process by generating entities (also known as "customers" in queuing theory).
- **Simulink Function exponentialArrivalTime():** Returns data representing the interarrival times for the generated entities. The interarrival time of a Poisson arrival process is an exponential random variable.
- **Entity Queue block:** Stores entities that have yet to be served in FIFO order
- **Entity Server block:** Models a server having a constant service time.

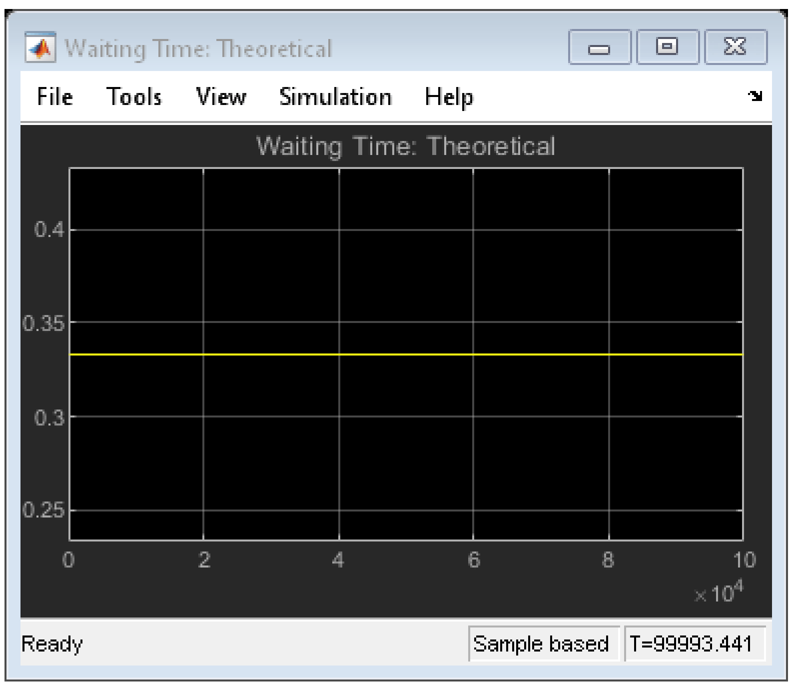
This model is similar to the M/M/1 Queuing System model, except that the service time in this model is constant.

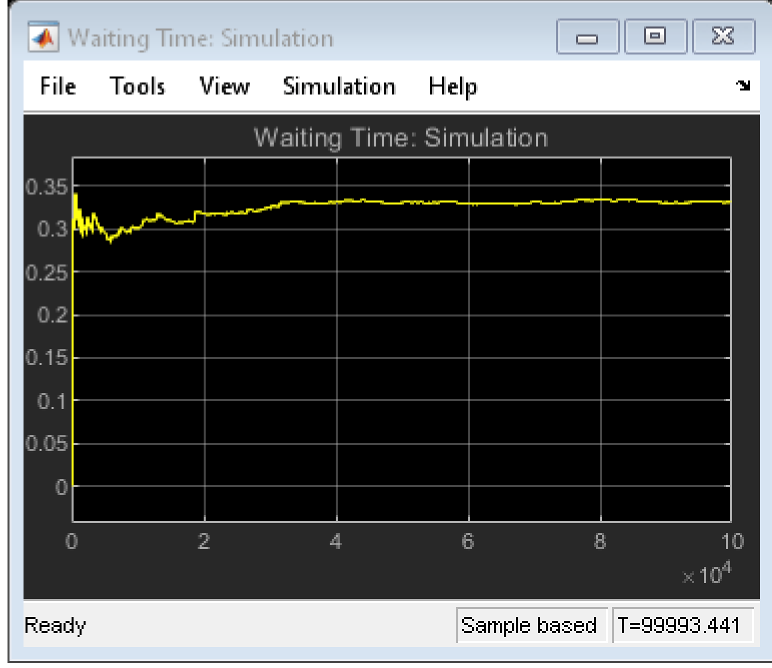



### Results and Displays

The model includes these visual ways to understand its performance:

- A scope showing the average waiting time of entities (customers) in the queue at over the course of the simulation.




### Theoretical Results

According to queuing theory, the mean waiting time in the queue equals $1/2(\mu - \lambda) - 1/2\mu$

where $\lambda$ is the arrival rate and $\mu$ is the service rate. This duration is half the theoretical mean waiting time in the queue for the M/M/1 queuing system with the same arrival rate and service rate.

#### Experimenting with the Model

Move the Arrival Rate Gain knob during the simulation and observe the change in the average waiting time.

#### Related Examples

- M/M/1 Queuing System
- G/G/1 Queuing System and Little's Law

#### References

[1] Kleinrock, Leonard, Queueing Systems, Volume I: Theory, New York, Wiley, 1975.

## See Also

Entity Server | Queue | Entity Terminator | Entity Generator

## Related Examples

- “Create a Discrete-Event Model”
- “Manage Entities Using Event Actions”
- “Explore Statistics and Visualize Simulation Results”

# G/G/1 Queuing System and Little's Law

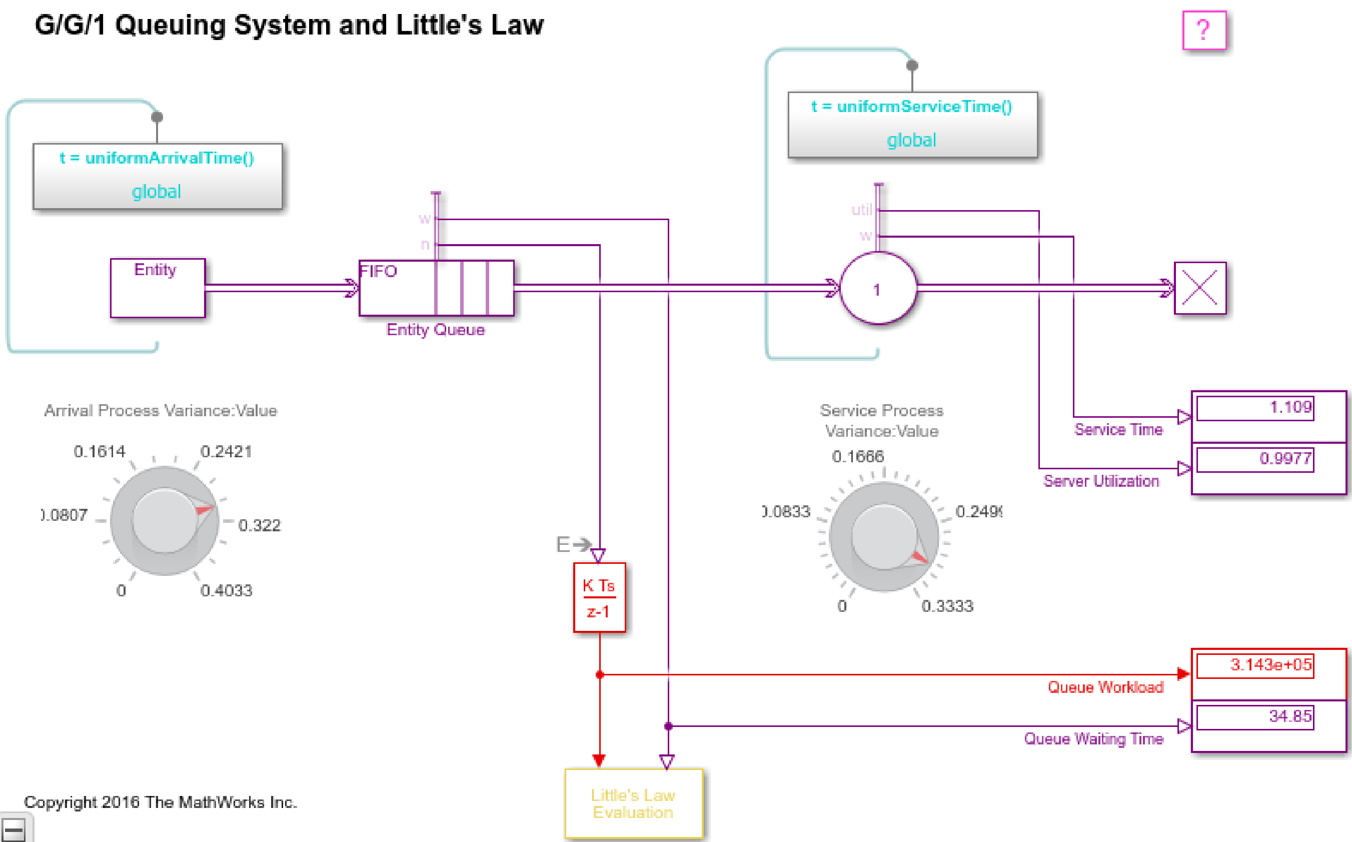
## Overview

This example shows how to model a single-queue single-server system in which the interarrival time and the service time are uniformly distributed with fixed means of 1.1 and 1, respectively. The queue has an infinite storage capacity. In the notation, the G stands for a general distribution with a known mean and variance; G/G/1 means that the system's interarrival and service times are governed by such a general distribution, and that the system has one server. You can change the variances of the uniform distributions. You can use this model to examine Little's law.

## Structure of the Model

The model includes the components listed below:

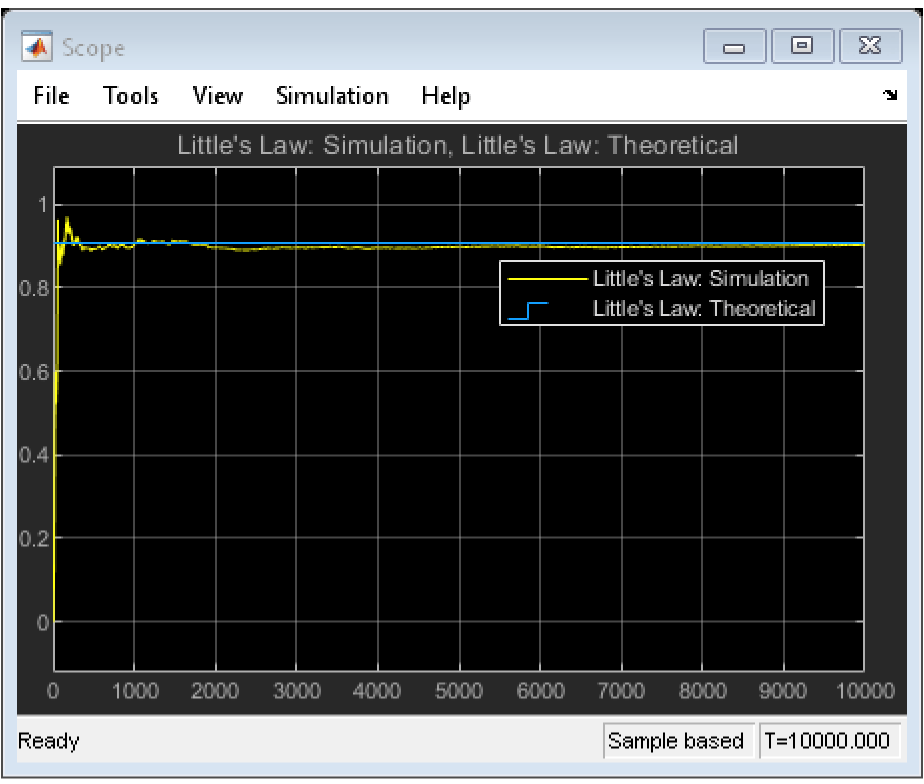
- **Entity Generator block:** Generates entities (also known as "customers" in queuing theory).
- **Simulink Function uniformArrivalTime():** Returns data representing the interarrival times for the generated entities. After you set the distribution's variance using the Arrival Process Variance knob, the function computes a uniform random variate with the chosen variance and mean 1.1. To see the computation details, double-click the Simulink Function and open the block labeled Uniform Distribution.
- **Entity Queue block:** Stores entities that are to be served in FIFO order
- **Entity Server block:** Models a server whose service time has a uniform distribution.



### Results and Displays

The model includes these visual ways to understand its performance:

- Display blocks that show the queue workload, average waiting time in the queue, average service time, and server utilization.
- A scope comparing empirical and theoretical ratios. See the discussion of Little's law below.




### Little's Law

You can use this model to verify Little's law, which states the linear relationship between average queue length and average waiting time in the queue. In particular, the expected relationship is as follows:

Average queue length = (Mean arrival rate)(Average waiting time in queue)

The Entity Queue block computes the current queue length and average waiting time in the queue. The subsystem called Little's Law Evaluation computes the ratio of average queue length (derived from the instantaneous queue length via integration) to average waiting time, as well as the ratio of mean service time to mean arrival time. The two ratios appear on the plot labeled Little's Law.

Another way to interpret the equation above is that, given a normalized mean service time of 1, you can use the average waiting time and average queue length to derive the system's arrival rate.

### Little's Law Applied to the Server

You can also use this model to verify the linear relationship that Little's law predicts between the server utilization and the average service time. The Entity Server block computes the server

utilization and average waiting time in the server. Because each entity can depart from the server immediately upon completing service, waiting time is equivalent to service time for the server in this model.

#### Experimenting with the Model

Move the Arrival Process Variance knob or the Service Process Variance knob during the simulation and observe how the queue content changes. When traffic intensity is high, the average waiting time in the queue is approximately linear in the variances of the interarrival time and service time. The larger the variances are, the longer an entity has to wait, and the more entities are waiting in the system.

#### Related Examples

- M/D/1 Queuing System
- M/M/1 Queuing System

#### References

[1] Kleinrock, Leonard, Queueing Systems, Volume I: Theory, New York, Wiley, 1975.

## See Also

Entity Server | Queue | Entity Terminator | Entity Generator

## Related Examples

- “Create a Discrete-Event Model”
- “Manage Entities Using Event Actions”
- “Explore Statistics and Visualize Simulation Results”

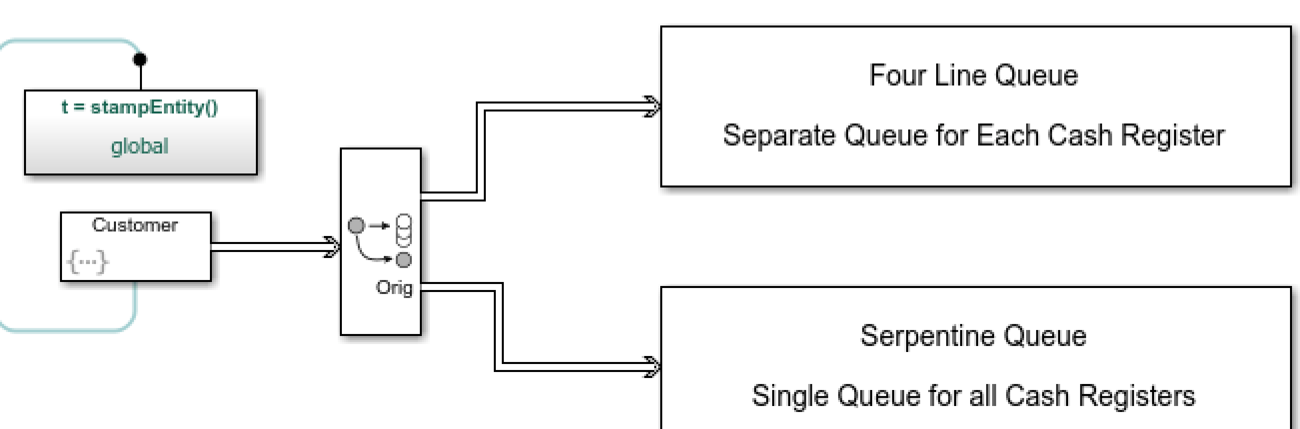
# Comparing Queuing Strategies

### Overview of Example

Have you ever been in a supermarket checkout and wondered why you are in the slowest line? This example shows how queuing systems can be modeled in SimEvents for this type of application. Two parallel versions of a simple model of a four register supermarket counter are presented - one that uses four separate queues and one with a single "serpentine" queue that feeds all registers.

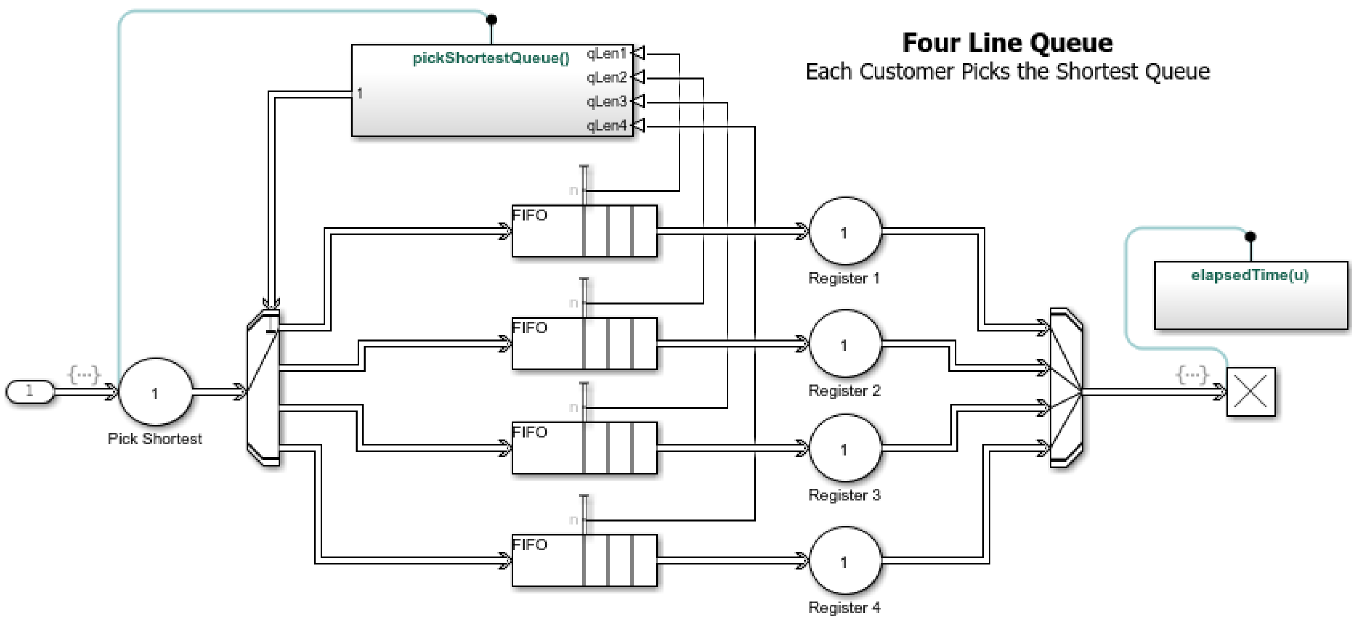
### Setup

To begin, we model random customers entering the checkout area using entities in SimEvents to represent customers that can be generated at random time intervals following an exponential distribution. During generation we specify a random duration (also exponentially distributed) that a customer will take to be served at a register by assigning a special attribute to the corresponding entity. The average service time is set at 2 mins and the average arrival time is set at 1 min. Each customer is cloned after generation so that the two different line configurations can be exercised identically.



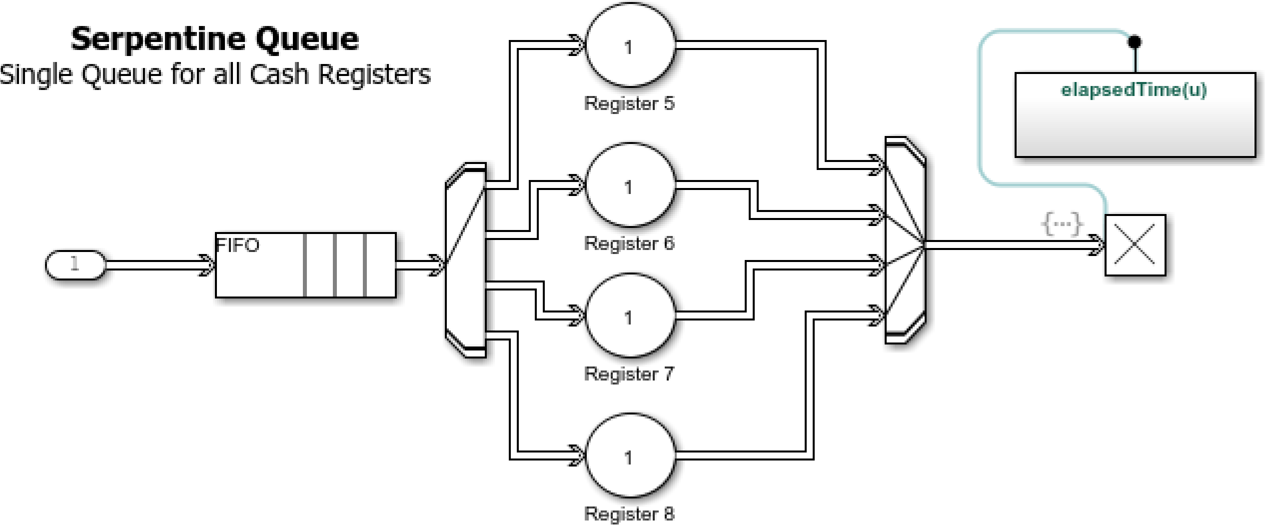

### Four Separate Queues

To model the case where four separate queues feed the four cash registers, we use a Switch that routes customers to the shortest of the four Queues. Each Queue then feeds a Server representing a checkout register. This Server holds the customer for the amount of time that was setup during generation.



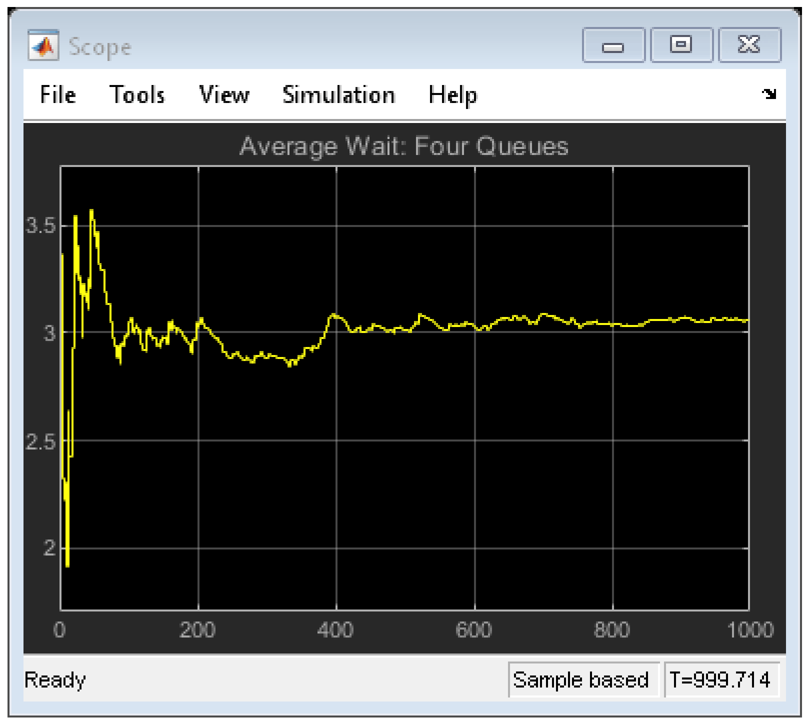

### "Serpentine" Queue

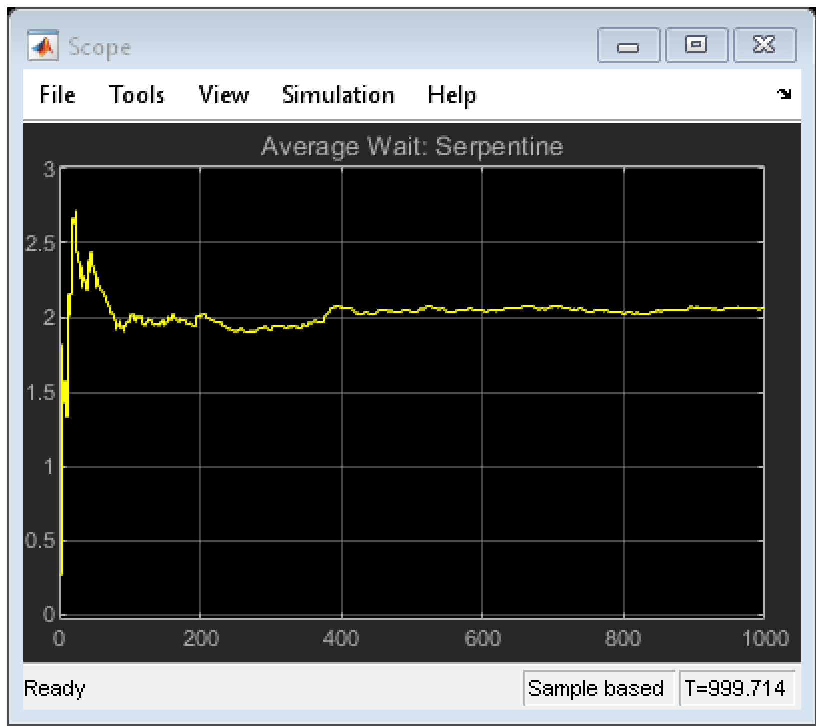
To model the "serpentine" queue, we use a single Queue that feeds the four registers via a Switch that routes customers to a free register when one becomes available.




### Conclusion

The configuration with the four queues on average results in longer wait times. This example shows the modeling of queueing systems in SimEvents for evaluating applications such as shortest lines.




## See Also

Queue | Entity Terminator | Entity Generator

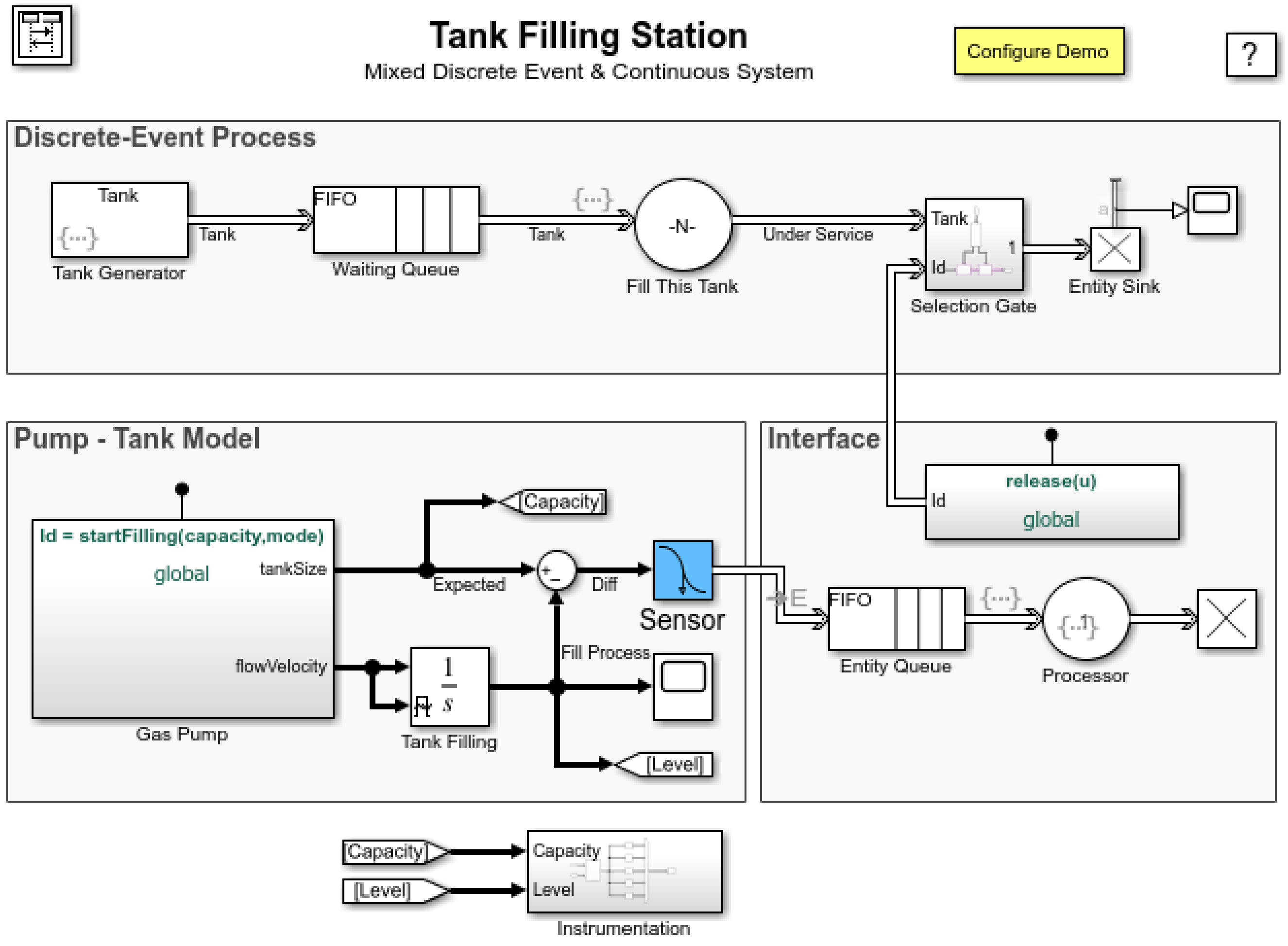
## Related Examples

- "M/M/1 Queuing System" on page 6-37
- "Create a Discrete-Event Model"
- "Explore Statistics and Visualize Simulation Results"
- "Manage Entities Using Event Actions"

# Modeling Hybrid Systems - Tank Filling

### Description

This example shows a hybrid system with both continuous time and discrete event sections. The discrete event part models tanks, represented by entities, which are being queued and need to be filled up. Each tank has a "Capacity" attribute. The continuous time part models the process of filling up a tank, modeled by an Integrator. When a tank is filled to capacity, this event can be detected by a Hit Crossing block, which will generate a message corresponding to this event. The generated message will trigger the server to release the tank.




### Structure of the Model

The model includes the following components:

- **Tank Generator**: Generates tanks periodically with each tank having an arbitrarily assigned Capacity attribute.

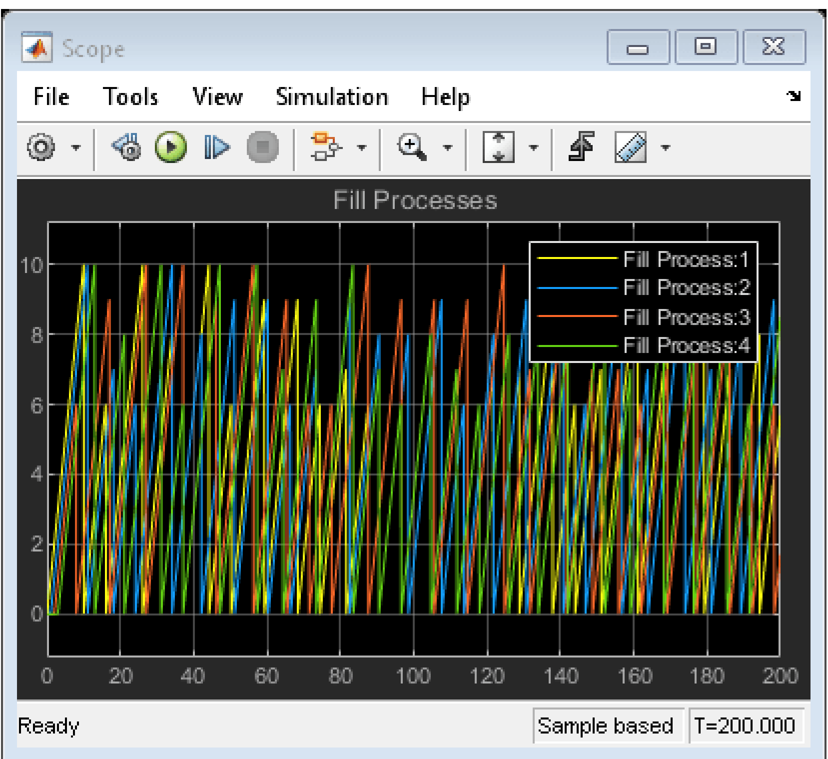
- **Waiting Queue**: Queues tanks waiting to be filled
- **Fill This Tank**: Serves tanks and calls into the Simulink Function `startFilling` to pass the tank's capacity attribute to the time-based section of the model.
- **Tank Filling**: Models the process of filling each tank up to capacity
- **Sensor**: Detects when the amount filled in the tank has reached capacity and when this happens, sends a message to the discrete-event section of the model. Sensor serves as a bridge between the time-based section and even-based section.
- **Processor**: Receives message generated from the Sensor and decides which tank to be released from the Server. It then calls the Simulink Function named `release` to generate a release message for a specific tank.
- **Selection Gate**: Receives a release message, and in response, opens the gate to let the specific tank through.
- **Configure Demo**: Sets the number of gas pumps in the gas station and turns on/off of the animation. To show the animation, please use a gas pump number between 1 and 20.

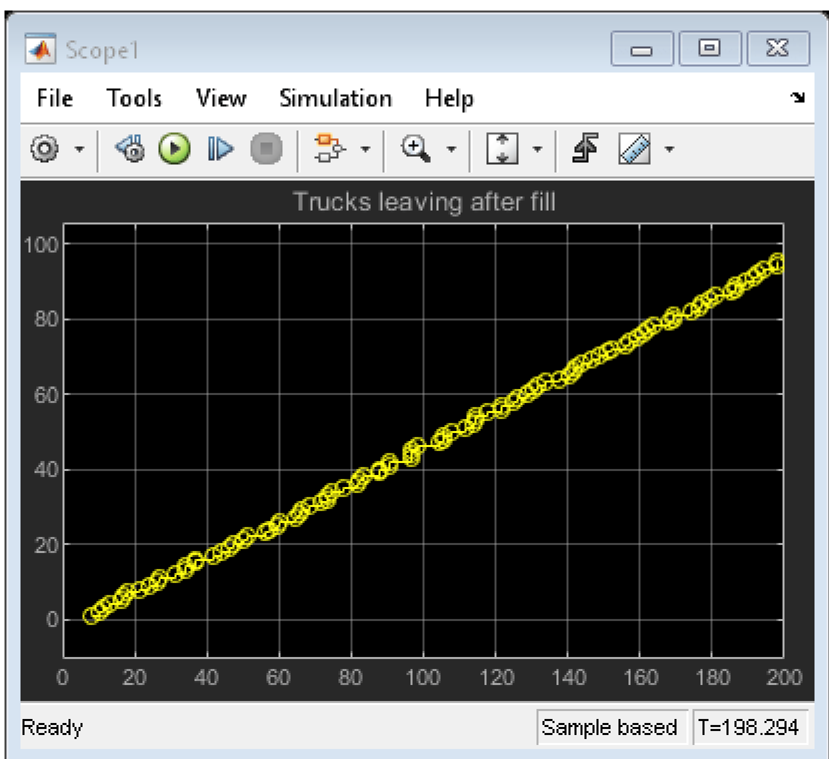
#### Domain Crossings Between Time Domain and Event Domain

SimEvents automatically handles any exchange of data across the time and event domains by automatically inserting gateways where needed. These positions are annotated in the model using `E`. In this model, a gateway has been inserted at the input port of the Entity Queue block that is connected to the Hit Crossing block since it receives a message from the time domain section of the model.

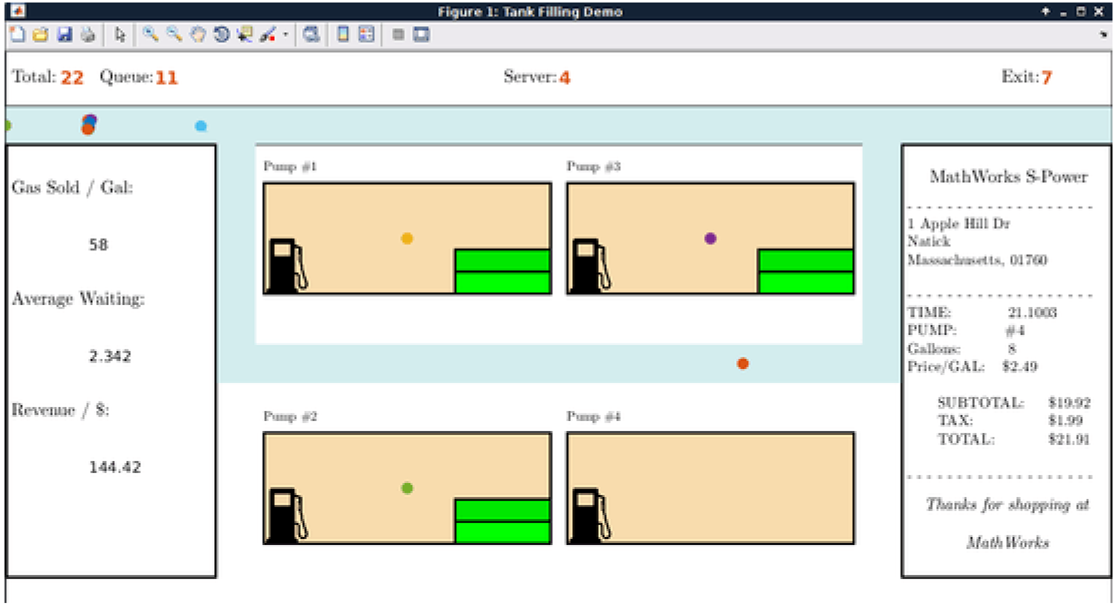
#### Results

The Scope block labeled "Fill Process" and "Trucks leaving after fill" shows the results of the simulation.




If `Show Animation` check box is selected in **Configure Demo**, an animation window appears for visualizing the demo. A screenshot of the animation with four gas pumps is shown below:




## See Also

Queue | Entity Terminator | Entity Generator

## Related Examples

- "Route Vehicles Using an Entity Output Switch Block" on page 3-2
- "M/M/1 Queuing System" on page 6-37
- "Create a Discrete-Event Model"
- "Explore Statistics and Visualize Simulation Results"
- "Manage Entities Using Event Actions"

# Resource Allocation from Multiple Pools

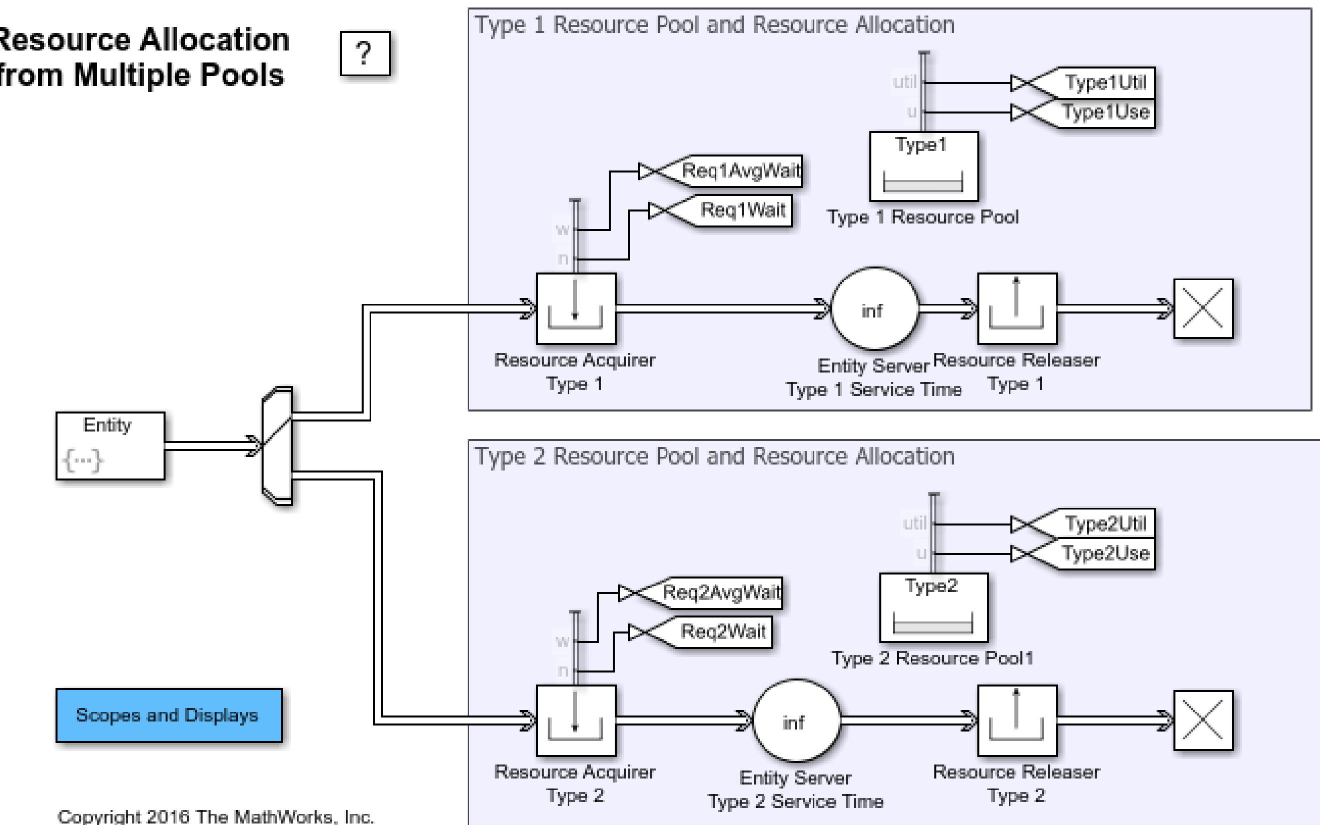
## Overview

This example shows a technique for allocating resources from multiple resource pools. It shows how to choose a pool from which to draw a resource, based on given criteria.

## Structure of the Model

There are two main components of the model.

- Request Generation and Queuing
- Resource Pools and Resource Allocation




## Request Generation and Queuing

The Entity Generator block generates requests using a Uniform distribution. In order for these requests to be acted upon, they require a resource from one of the two resource pools. Each of the requests has an attribute that specifies the kind of resource it requires. The requests move to one of the queues dedicated for each type of resource pool.

### Resource Pools and Resource Allocation

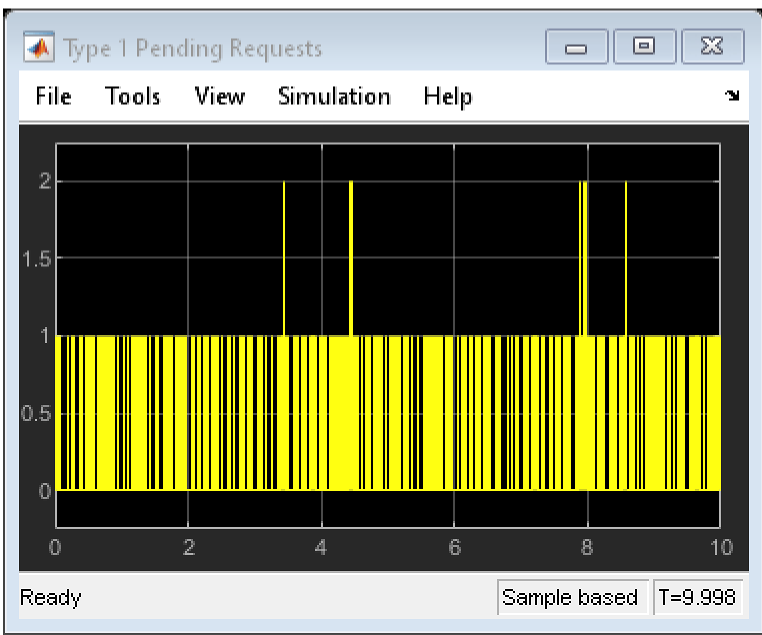
The model has two resource pools, Type 1 and Type 2. The Type 1 and Type 2 Resource Pool blocks model the pools. These pools hold the resources before and after their use. The size of each pool is defined as parameter of the corresponding block.

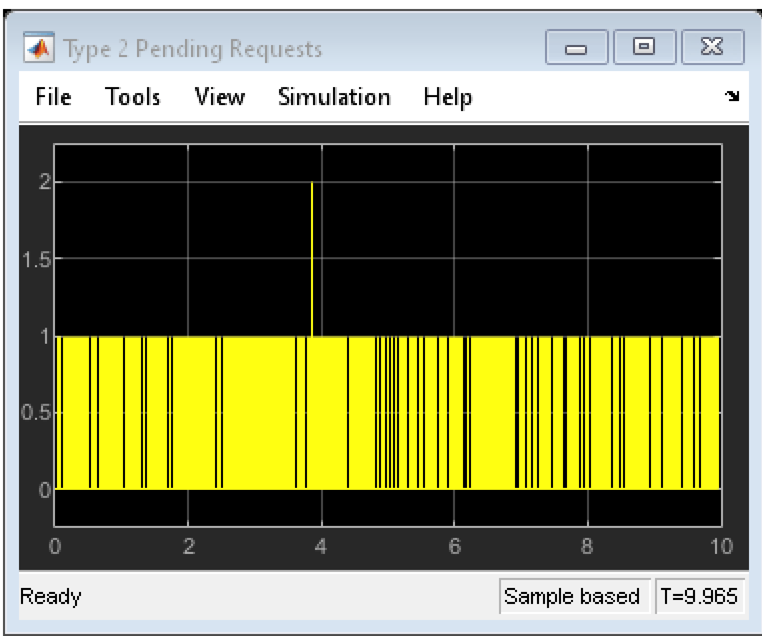
The Resource Acquirer and Resource Releaser blocks manage the acquisition and the return of the resource. The Entity Server block in that region models the duration for which the resources are used.

### Results and Displays

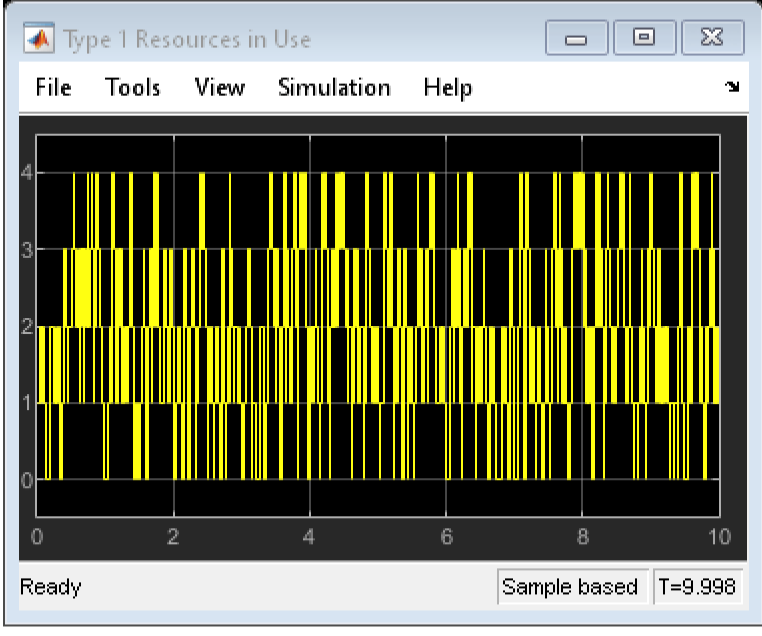
This model includes the following plots.

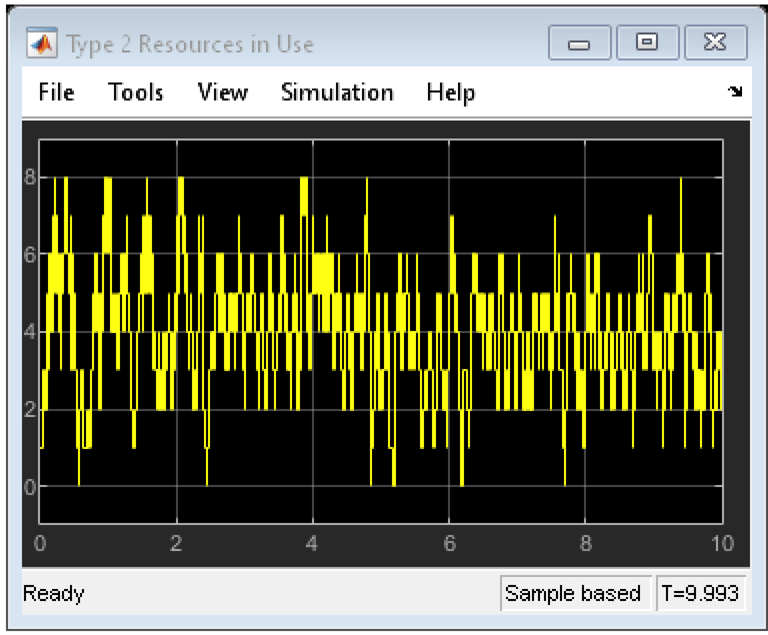
- The Pending Type 1 Requests and Pending Type 2 Requests plots show the number of requests waiting for resources from the corresponding pools. You can see that more requests for Type 1 resources wait compared to requests for Type 2 resources.






- The Type 1 Resources in Use and the Type 2 Resources in Use plots show the instantaneous values of the number of resources available for use in the corresponding resource pools.






- The average wait time for acquiring each type of resources is reported by the Resource Acquire blocks for Type 1 Resource and Type 2 Resource.

```
Average Wait Time for Resource Type 1 = 0.00
Average Wait Time for Resource Type 2 = 0.00
```

- The average amount of resources in use at each resource pool is reported by Resource Pool blocks for Type 1 Resource and Type 2 Resource.

```
Average Use of Resource Type 1 = 0.47
Average Use of Resource Type 2 = 0.50
```

The model has the following configuration:

- Resource request distribution: Type 1 = 0.4, Type 2 = 0.6
- Duration of Type 1 resource use : 0.05
- Duration of Type 2 resource use : 0.07
- Type 1 pool size : 4
- Type 2 pool size : 8

From the above results, you can see that the larger pool size of Type 2 resources results in a lower average wait time even with a higher request rate and longer duration of resource use.

#### Experimenting with the Model

To vary system behaviors, like the number of resources available and wait times for resources, change the following settings:

- The probabilities for changing the generation rate of the resource requests in the `Intergeneration time action` parameter of Entity Generator.
- The `Service time` parameter of the Entity Server blocks in the Resource Pool and Resource Allocation regions of the model for changing the duration of the resource usage.

- The `Resource amount` parameter of the Type 1 Resource Pool and Type 2 Resource Pool blocks to change the number of resources in the pool.

## See Also

Resource Pool | Resource Acquirer | Resource Releaser | Entity Generator

## Related Examples

- "Resource Allocation Modeling"

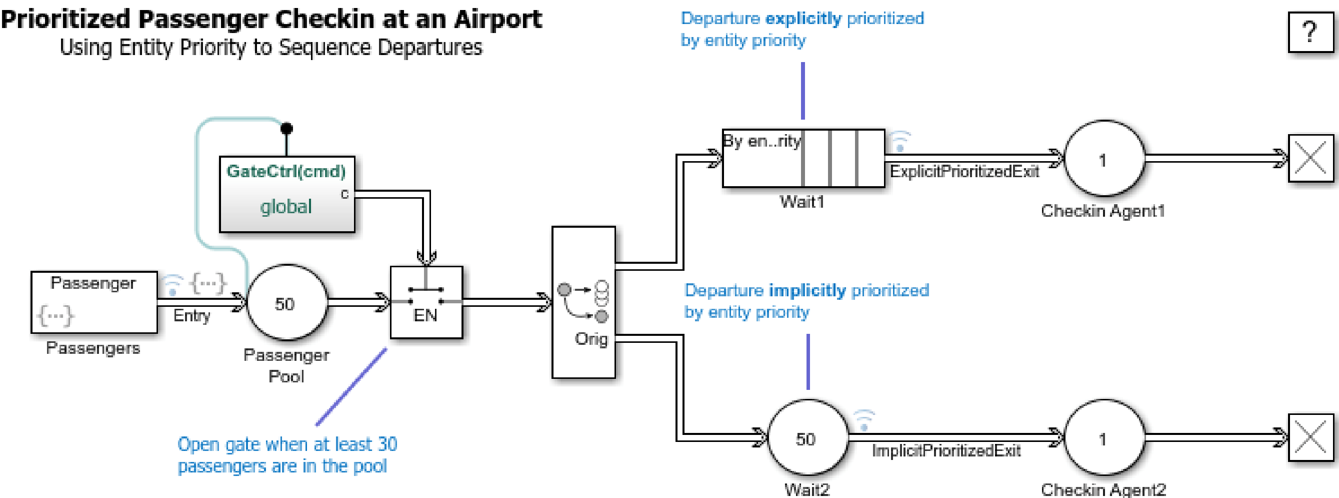
# Using Entity Priority to Sequence Departures

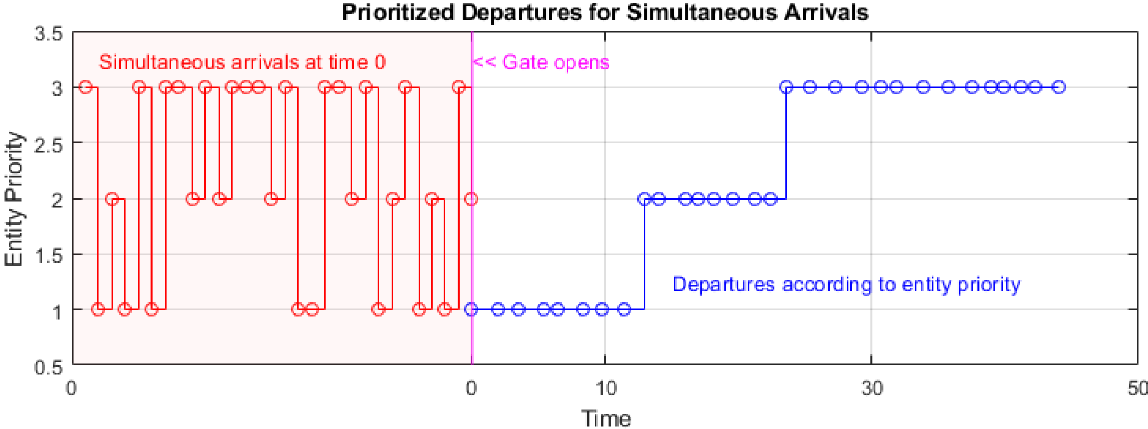
### Description

This example shows how to use entity priority to sequence entity departures when multiple entities are available to depart. The example models an airport check-in counter where passengers arrive to be checked in. Passengers can have either First-Class, Business Class, or Economy Class reservations, modeled using entity priority values 1, 2 and 3 respectively.

The example models two scenarios:

- The first involves a Priority Queue in which passengers are explicitly sorted by their entity priority. This ensures that all First Class passengers are sorted before Business Class, which are in turn sorted before Economy Class. This is called "Explicit Prioritization" in the model.
- The second involves an Entity Server in which all passengers are waiting in an unordered fashion. When the check-in agent is available, all passengers schedule departure events. These simultaneous events are ordered by entity priority, ensuring that the entity with the highest priority will depart first, and all the other departures will fail. This is called "Implicit Prioritization" in the model.






### Results

In this simulation, the Entity Generator simultaneously generates 30 passengers at time 0. Once 30 are available in the Pool, the Entity Gate opens and all passengers can depart. The results show the simultaneous arrivals of all passengers at time 0 with their entity priorities. When the Gate opens, it is seen that all passengers depart in order of their entity priority.

## See Also

Queue | Entity Terminator | Entity Generator

## Related Examples

- “Create a Discrete-Event Model”
- “Manage Entities Using Event Actions”
- “Entity Priorities” on page 1-36

# Using Custom Visualization for Entities

### Overview

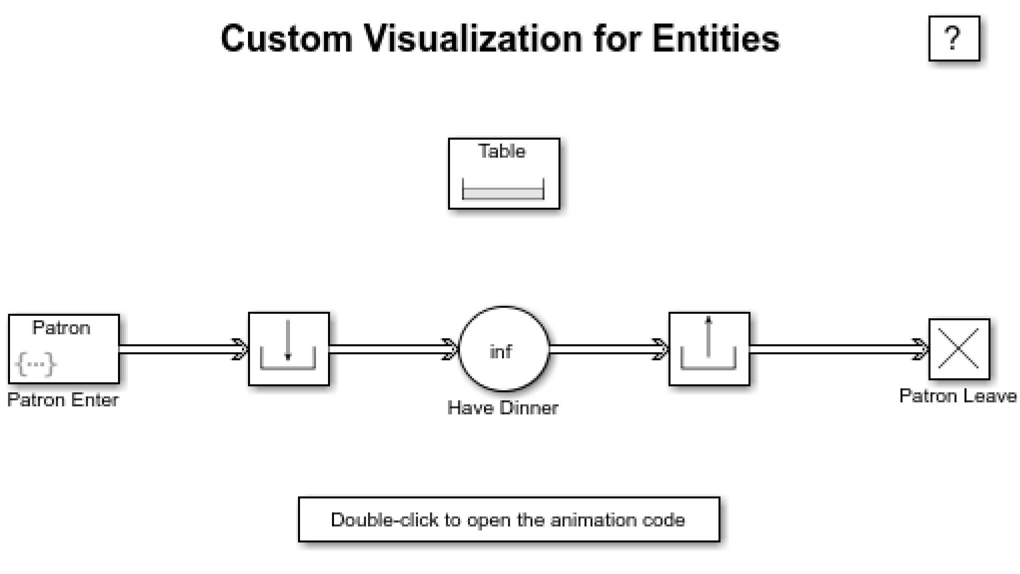
These examples show how you can create MATLAB based custom visualization for entities. The example illustrates the visualization of a restaurant layout with customer entities entering, dining, and leaving.

### Structure of Model

The model contains the following major components:

- The Resource Pool block models the tables in the restaurant. Since there are 10 tables in the restaurant, the resource amount is 10.
- The Entity Generator block (Patron Enter) generates entities representing customers. They enter a waiting area, represented by a Resource Acquire. Here they wait for a free table.
- When a table is available for a customer, he can move to the Entity Server block which models the duration of eating.
- When the customer is done eating, he releases the table back to the pool and exits the restaurant.

```
modelname = 'seCustomVisualization';
open_system(modelname);
```




### Visualizing the Restaurant

seRestaurantAnimator visualizes the restaurant as follows:

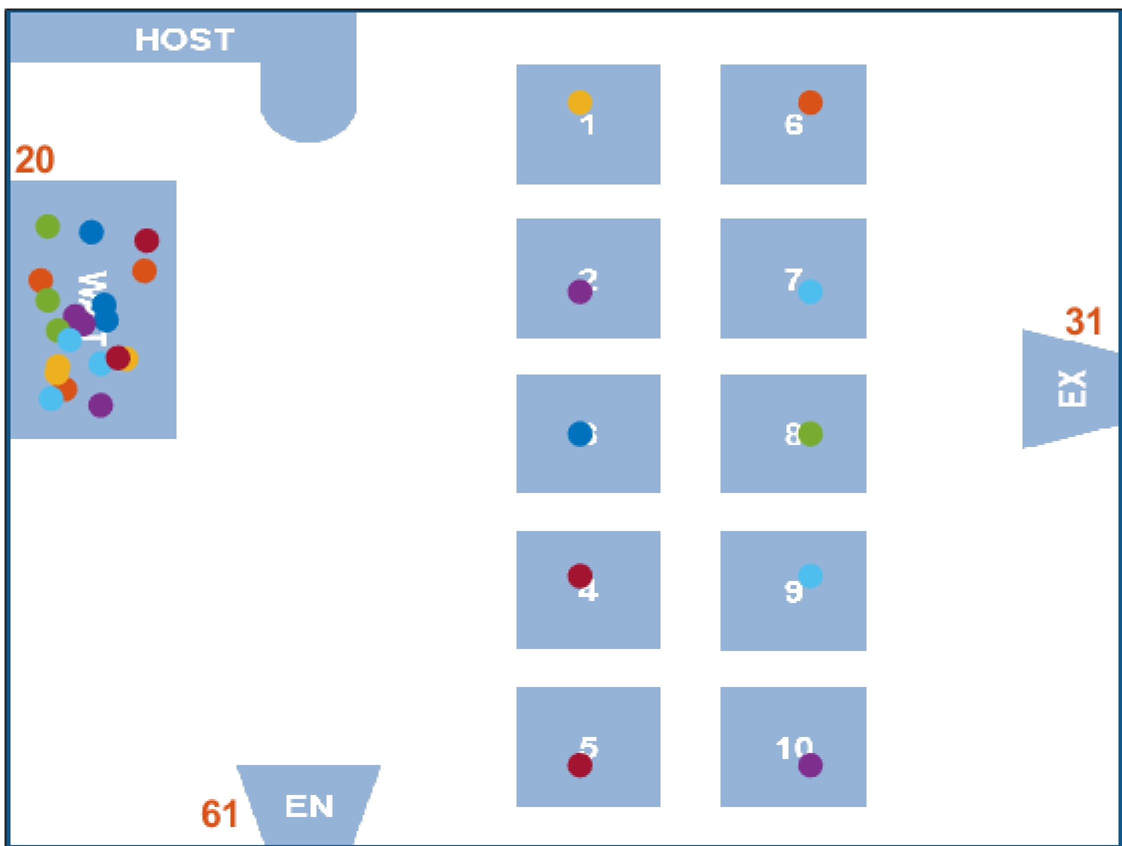
- seRestaurantAnimator provides the visualization of the restaurant layout for the model.
- It generates the figure containing the layout of a restaurant with an entrance, a waiting area, 10 dining tables, and an exit.

- As entities move during the simulation, it creates a marker (glyph) for each entity in the figure and programs motion for the marker so that it appears to move from one point to another.
- The animator assigns a table ID for each waiting customer and shows the customer moving to the table.
- To inspect the attributes of the customer entity, pause the model and click on a customer entity glyph. The figure displays the TimeToDine attribute.
- To make the motion appear continuous, it uses a MATLAB timer to periodically execute a function that incrementally moves each entity towards its destination.
- It uses MATLAB graphics to display statistics on the figure about the number of entities entering, waiting, and leaving.
- Clicking an entity in the visualization displays the attributes that it contains. It uses a MATLAB graphics callback to program a ButtonDownFcn on each entity marker.

```
animator = seRestaurantAnimator;
```

To simulate the model, enter:

```
sim(modelname);
```



```
close(animator.getFigureHandle);
bdclose all;
clear modelname animator
```

## See Also

Queue | Entity Terminator | Entity Generator

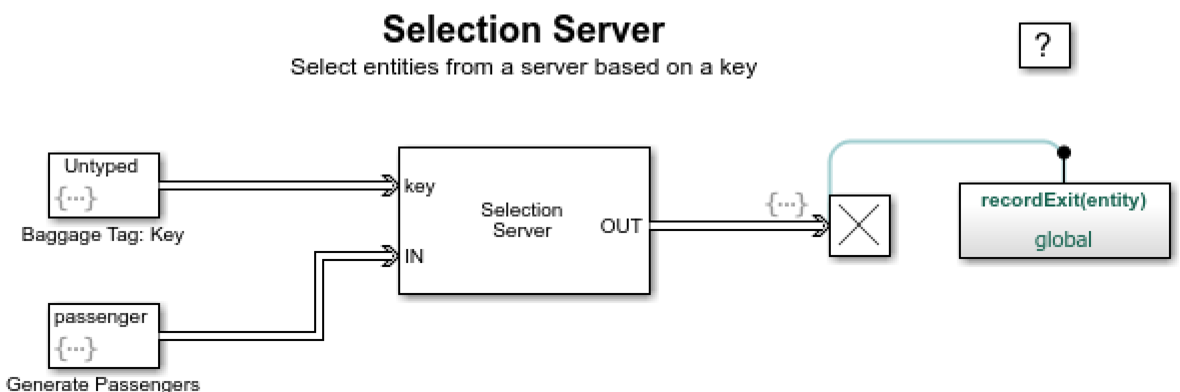
## Related Examples

- “Observe Entities Using simevents.SimulationObserver Class” on page 10-5
- “Create a Discrete-Event Model”
- “Manage Entities Using Event Actions”

# Selection Server - Select Specific Entities from Server

### Description

This example shows how you can use the MATLAB Discrete Event System block to write a custom N-Server from which specific entities can be selected using a key lookup. Passengers enter from the IN port of the block and are stored in the block until a message arrives at the KEY port carrying a lookup key. Upon receiving this message, the system schedules an "Iterate" event during which it can visit every entity stored in it and output the one that matches the key.




### Results

The simulation prints information about entities entering the MATLAB Discrete Event System block and selection commands.

```
Passenger entry: key = 2.000000
Passenger entry: key = 10.000000
Baggage entry: key = 10.000000
Passenger exit: key = 10.000000
Passenger entry: key = 4.000000
Passenger entry: key = 5.000000
Passenger entry: key = 3.000000
Baggage entry: key = 3.000000
Passenger exit: key = 3.000000
Passenger entry: key = 8.000000
Passenger entry: key = 7.000000
Passenger entry: key = 1.000000
Baggage entry: key = 8.000000
Passenger exit: key = 8.000000
Passenger entry: key = 6.000000
Passenger entry: key = 9.000000
```

## See Also

MATLAB Discrete-Event System

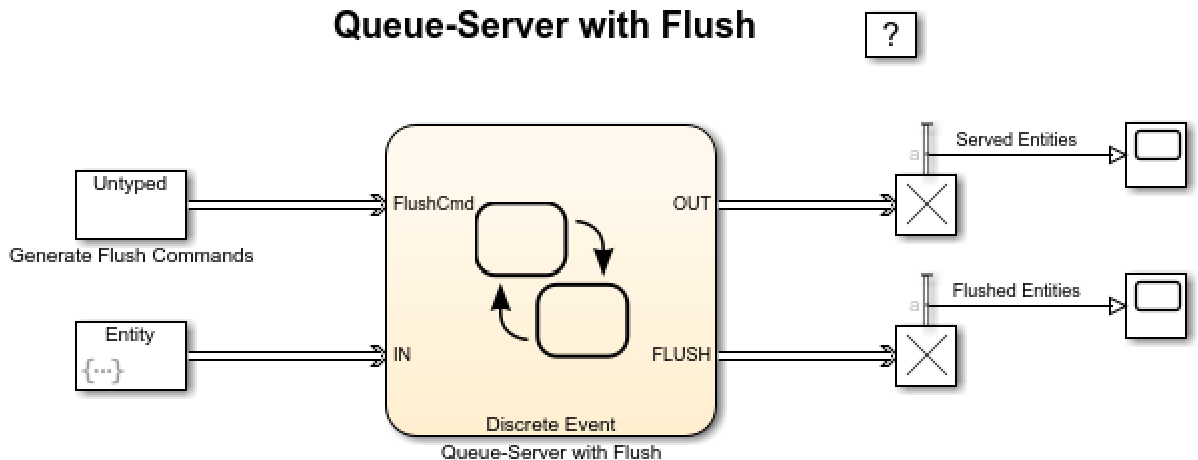
## Related Examples


- “Delay Entities with a Custom Entity Storage Block” on page 9-9
- “Create a Custom Entity Storage Block with Iteration Event” on page 9-14
- “Custom Entity Storage Block with Multiple Timer Events” on page 9-19
- “Custom Entity Generator Block with Signal Input and Signal Output” on page 9-24
- “Build a Custom Block with Multiple Storages” on page 9-31
- “Create a Custom Resource Acquirer Block” on page 9-38

# Flush Entities from a Queue-Server

### Description

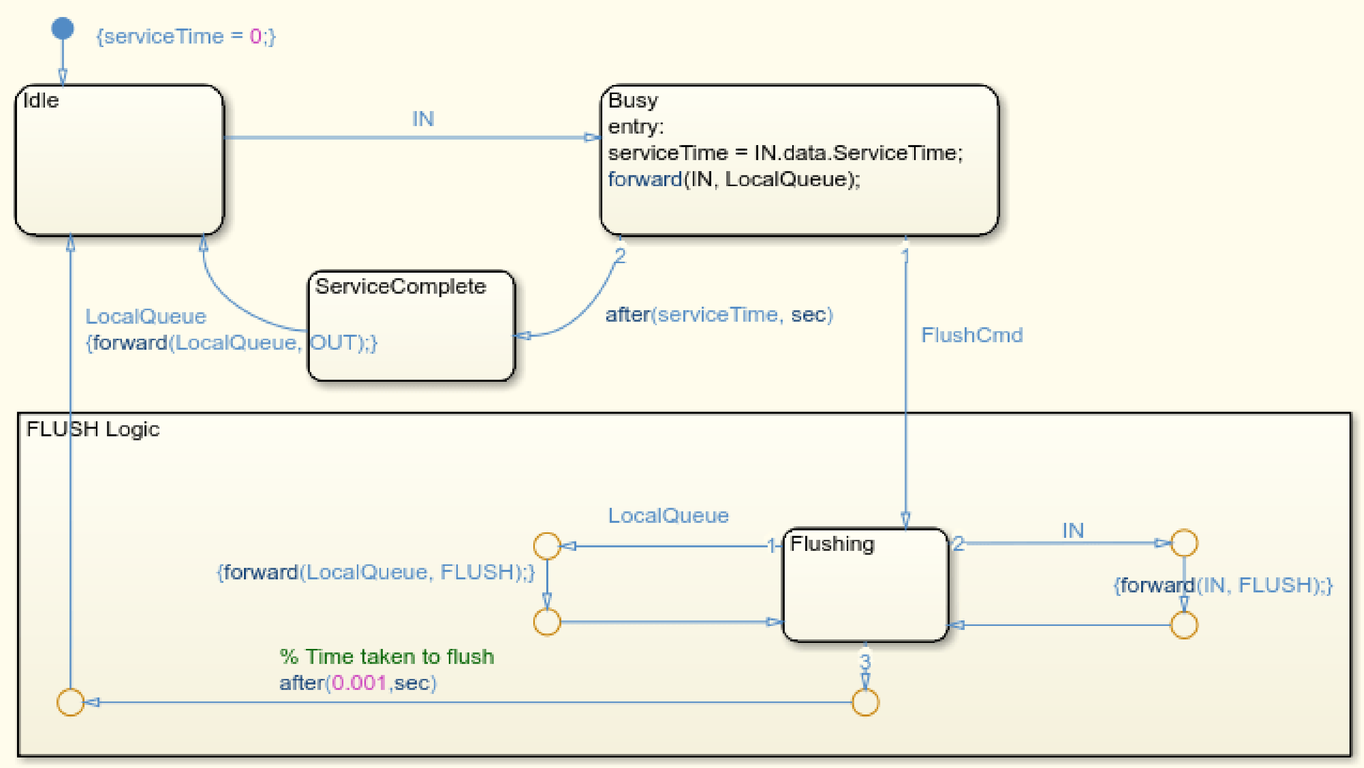
This example shows how you can use a Discrete-Event Chart block to model a queue-server that can flush entities when it receives a message on the "FlushCmd" port.




### Discrete-Event Chart

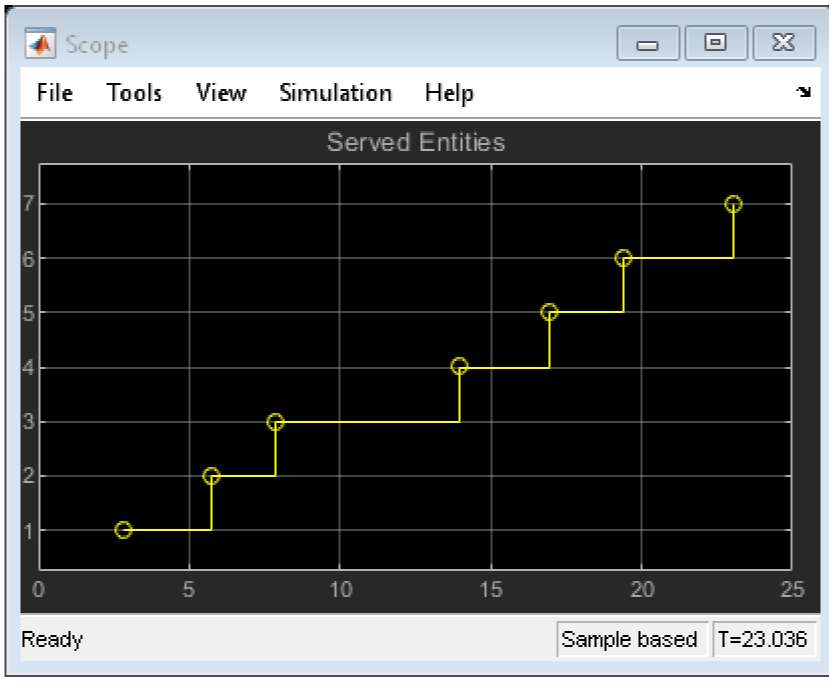
The Discrete-Event Chart implements a single server with two states "Idle" and "Busy". The server is busy when an entity arrives at the IN port. It holds the entity in a local queue named "LocalQueue" until its service time expires. After this time the entity is forwarded out.

While serving an entity, if a "FlushCmd" command is received, it transitions to the "Flushing" state in which it iterates over its input queue and forwards each of its waiting entities out from the FLUSH output port. Additionally, it also forwards the entity that is currently being served in the LocalQueue.




### Results

The results show that a flush command was received at times 10 and 20 during the simulation. At these instants, all entities in the queue-server were flushed out.



## See Also

Queue | Discrete-Event Chart | Entity Generator

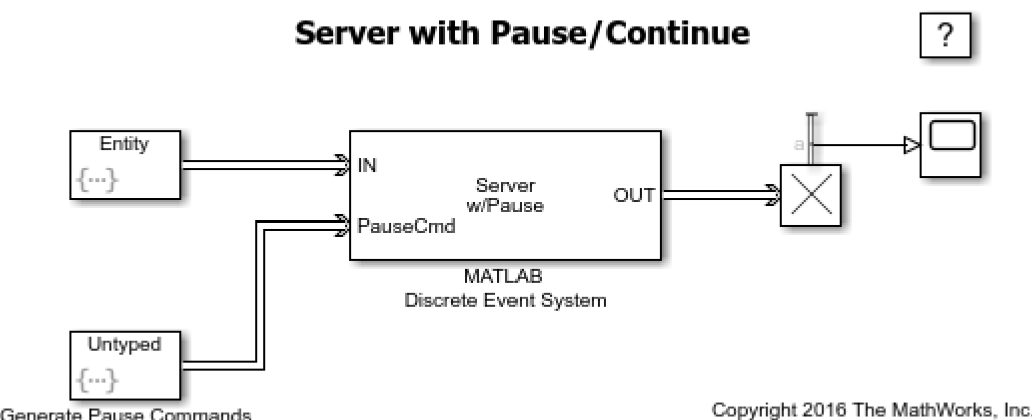
## Related Examples

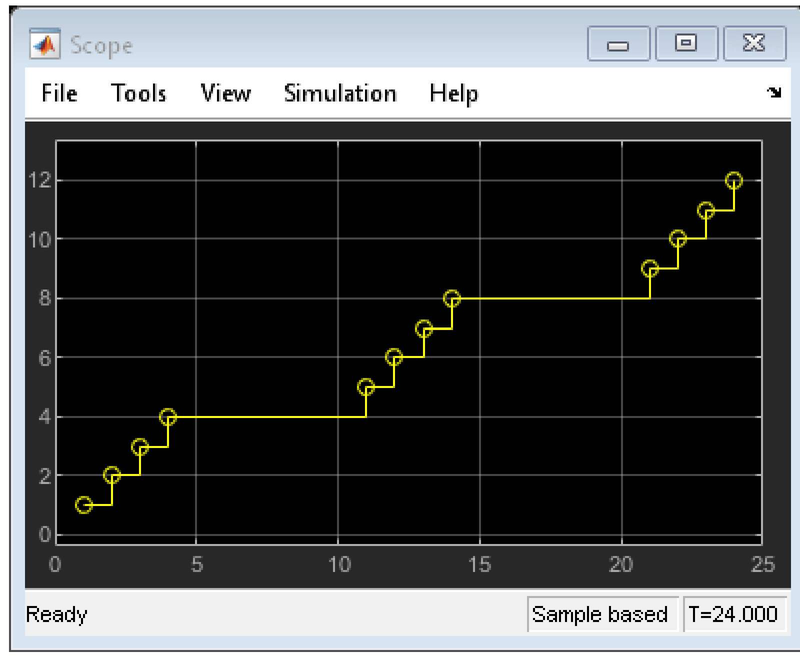
- "Discrete-Event Chart Precise Timing" on page 8-6
- "Trigger a Discrete-Event Chart Block on Message Arrival" on page 8-9
- "Dynamic Scheduling of Discrete-Event Chart Block" on page 8-18
- "Create a Discrete-Event Model"
- "Manage Entities Using Event Actions"

# Server with Pause/Continue

### Description

This example shows how you can use a MATLAB Discrete Event System block to model a single server that can pause service. The input port IN receives entities to be served. Additionally, the system may receive sporadic pause commands on port PauseCmd. If the message received on the PauseCmd port carries data=1, the system pauses. The system reschedules service for the current entity when it receives a continue message on this port, i.e. a message that carries data=0.




## See Also

MATLAB Discrete-Event System

## Related Examples


- "Delay Entities with a Custom Entity Storage Block" on page 9-9
- "Create a Custom Entity Storage Block with Iteration Event" on page 9-14
- "Custom Entity Storage Block with Multiple Timer Events" on page 9-19
- "Custom Entity Generator Block with Signal Input and Signal Output" on page 9-24
- "Build a Custom Block with Multiple Storages" on page 9-31
- "Create a Custom Resource Acquirer Block" on page 9-38

# Simulation of a Medical Device

This example shows how to conduct automated tests to model a medical device that analyzes biology samples. This example also requires a Stateflow license.

### Objective

The objective of medical device modeling is to assess the optimal dimensions of the sample area to maximize the number of samples analyzed by the device per hour.

### Overview of System to Be Modeled

A medical device contains:

- Samples to be analyzed
- Reagent bottles

The vials that hold the samples to be analyzed are loaded on the left side of the device. The reagent bottles are loaded on the right side of the device.

### Process of Sampling for a Specific Test

1. The sample is mixed with the corresponding reagent in a cuvette.
2. The mixture sits for a certain duration for the reagent to act on the sample.
3. To take readings of the mix, the device shines a laser on the mixture.

See ''Automation Workflow Using Three Robot Arms'' to see how a medical device uses three robot arms to implement this process as an automated workflow.

### Automation Workflow Using Three Robot Arms

1. Robot arm 1 picks up a cuvette and places it in the testing area at the top.
2. Robot arm 2 draws a sample and puts it into the cuvette.
3. To create a mixture to be sampled, robot arm 3 draws the required amount of reagent corresponding to the test and puts it into the cuvette.
4. The mixture sits for a short duration to allow the reagent to act on the sample.
5. To take readings, the device shines a laser light on the mixture.
6. The device discards the cuvette.
7. This process is repeated until there are no more samples in the device.

**Model of Medical Device:** This is the SimEvents model for the medical device:

To modify the number of samples before starting to simulate, turn the knobs in the 'Samples' block.

- A sample bay is a device that contains holders to hold samples. To specify the number of sample bays to use, turn the knob that changes the variable 'nSampleBays'.
- To specify the number of samples that a sample bay can take, turn the knob that changes the variable 'samplesPerBay'.

#### The Model Has Three Primary Elements:

- Model samples
- Test data
- Model animation The model runs the length of the specified samples and dimensions. The model animation visualizes the simulation and allows you to interact with the simulation.

#### Model Samples

The block labeled 'Samples' models the sample holding area. At simulation start, the reagents area is loaded with all the reagents. The cuvette area is loaded with cuvettes. The sample area is loaded with patient samples.

#### Test Data

'BioSampleAnalyzerData.xlsx' contains tests requested by patients. It contains these worksheets:

- 'PatientTests' - Patient IDs and test IDs of the tests to be conducted.
- 'TestData' - Details of each test. For each test ID it contains information about the amount of sample to be used (sampleAmount), the reagent to be used (reagentId), the amount of reagent to

be used (reagentAmount), the priority for the test, and the amount of time the mixture must stay together (testTime) before taking a reading.

- 'TestNames' - List of names for the reagents.

#### Model Animation

To view and interact with the model using animation, click the switch on the 'Animation Switch' block. Clicking the switch 'On' opens the 'Hematology Diagnostic Instrument' window. If you do not use model animation, the example runs until the end.

#### The 'Hematology Diagnostic Instrument' Window Contains:

- Three robot arms at the top.
- Time - which displays elapsed time.
- Throughput - which displays the samples/hr of the device.
- Cuvette area - where cuvettes are placed. The number of remaining cuvettes is shown at the top of the cuvette area.
- Reagents area - where reagents are kept. Reagents display as blue circles. The top of each reagent circle displays an abbreviation of the reagent. The bottom of each circle displays the amount of remaining reagent. When the amount of remaining reagent falls below 3 units, the amount of remaining reagent displays in red. To refill a reagent ball, click it. When a test is skipped due to an insufficient amount of reagent, the corresponding reagent is highlighted in yellow.
- Samples area - where samples are kept. Samples display as pink balls. The top of each sample circle displays the testID. The bottom right of each sample circle displays the sample priority; the lower the number, the higher the priority. Samples are tested in order, highest priority to lowest priority. If a sample is waiting for a reagent to refill, the ball turns yellow. The model skips that sample and proceeds to the next sample until it can no longer continue. When a sample is completed, the ball turns orange. At the bottom of each sample column is a number indicating the sample bay. To test all the samples in a bay, click the corresponding sample bay number.

#### Things to Try

- Change the number of samples.
- Configure the parameters for the bioTester block.
- Sample - Change the 'Number of sample bays' and the 'Number of samples per bay' values for the samples area.
- Timing - Change timing related value, such as velocities for the robots and other timing related values.
- Toggle the 'Animation Switch'.
- Toggling the switch to 'Off' runs the simulation until all the samples are exhausted.
- Toggling the 'Animation Switch' 'On' shows the animation window. In this mode, you can interact with the simulation. To load samples, click one of the buttons at the bottom of a column that corresponds to a sample bay. When the button is clicked, sample data is read from the workspace variable 'patientTests' and the balls in the bay corresponding to the button are populated with samples. You can continue clicking the buttons.

#### Evaluating the Best Sample Area Dimensions

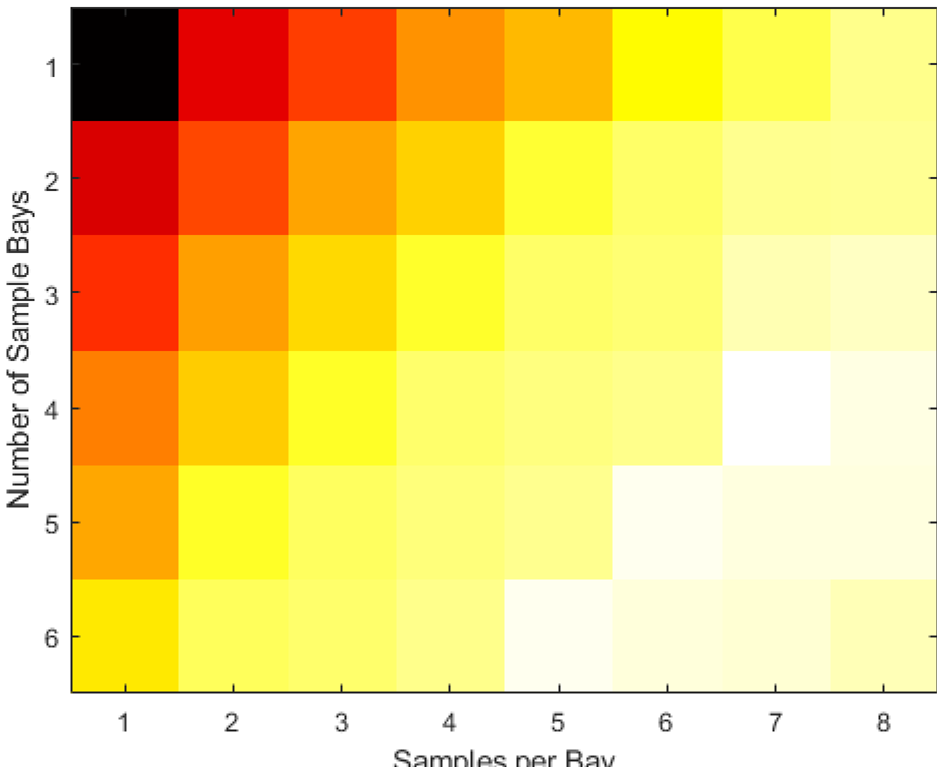
One of the objectives of a Medical Device builder might be to determine the sample area size that gives the best throughput for the device. One consideration is that increasing the size of the sample

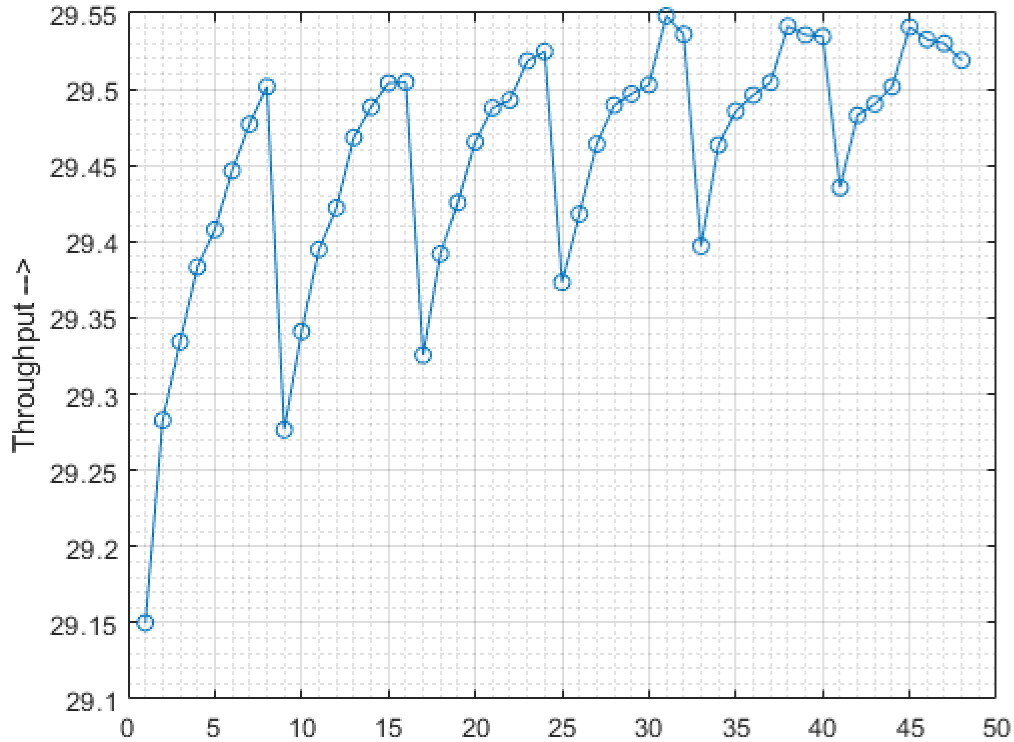
area adds to the amount of time it takes the robot arm to reach the samples that are furthest away. Reducing the size of the sample area reduces the travel time of the robot arm. There is however a fixed setup time required to load all the sample bays, follow the device initialization procedures, and turn on the device. This setup time is amortized across all samples. If the number of samples is low, the setup time adds to the overall throughput.

To find the best sample area dimensions, you can simulate the device with different sample area size configurations. The script searchDim.m performs searches across all the possible sample area dimensions and plots the throughput for each sample area dimension. The script calculates the throughput as:

**Throughput = (Number of samples)/(time to finish samples * 3600)**

The following plots show the results of running this script:






The first plot shows a Heat Map of throughput with 'samples per bay' along the horizontal and 'number of sample bays' along the vertical axis. The second plot shows a line plot of 'sample throughput' vs. 'number of samples'. As seen from these figures, the throughput for the dimensions corresponding to 6 bays and 9 samples per bay gives the highest throughput.

## See Also

MATLAB Discrete-Event System

## Related Examples

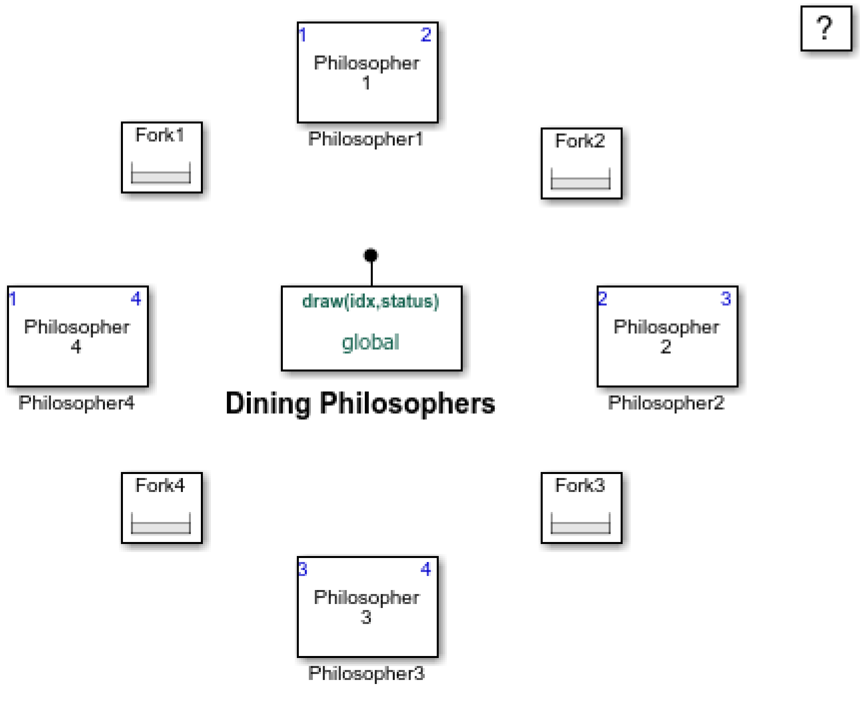
- “Delay Entities with a Custom Entity Storage Block” on page 9-9
- “Create a Custom Entity Storage Block with Iteration Event” on page 9-14
- “Custom Entity Storage Block with Multiple Timer Events” on page 9-19
- “Custom Entity Generator Block with Signal Input and Signal Output” on page 9-24
- “Build a Custom Block with Multiple Storages” on page 9-31
- “Create a Custom Resource Acquirer Block” on page 9-38

# Dining Philosophers Problem

### Problem Description

The Dining Philosophers problem is a classical problem, originally formulated by E.W. Dijkstra, to demonstrate classical problems in computer science and the programming of concurrent or parallel processes.

Four philosophers are seated at a table, spending their lives in an infinite cycle of thinking and eating. A philosopher must pick up both forks before he can eat. You can think of the philosophers as concurrent processes and the forks as shared resources. The problem is to determine the policy or algorithm so that each philosopher gets to eat and does not starve. For example, one algorithm is for each philosopher to pick up first the fork to his right, then the fork to his left, before he eats. That this will eventually lead to a deadlock situation where all of the philosophers are holding one fork, waiting for each other to put down their forks.

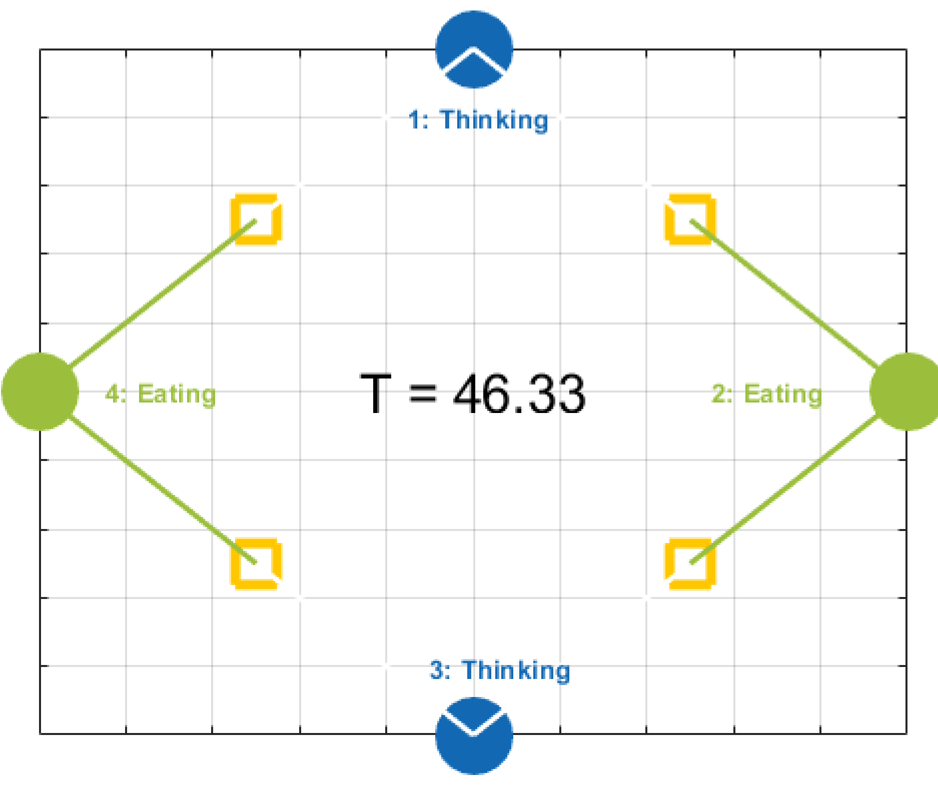


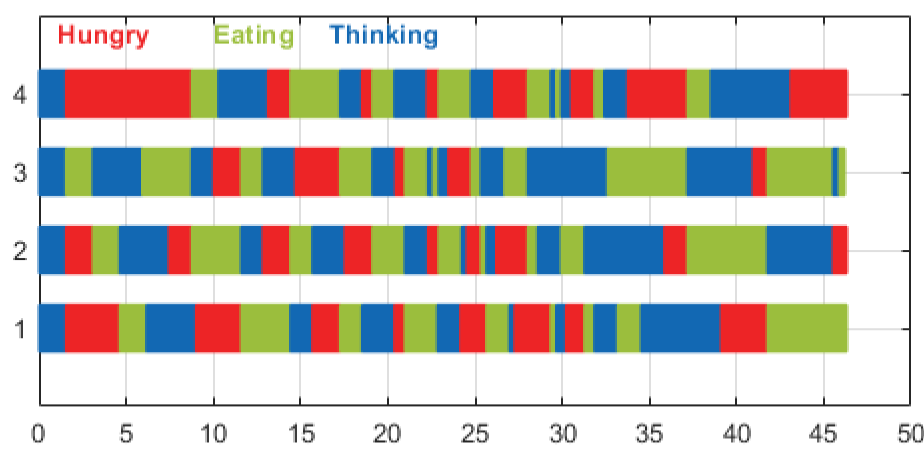

### Philosopher Model

This example models each philosopher as a discrete event system that generates a single entity at the start of the simulation. The position of the entity within the system reflects the state of the philosopher. Each state of the philosopher is an Entity Server that can hold the entity for a randomized period of time.

### Resource Hierarchy Solution

The algorithm illustrated here is a variation of the original algorithm described by Dijkstra. Each fork is numbered and philosophers first pick up the smaller numbered fork and then the larger numbered fork. This algorithm is sufficient to avoid deadlocks because only one philosopher can ever hold the highest numbered fork and consequently that philosopher can proceed to eat.



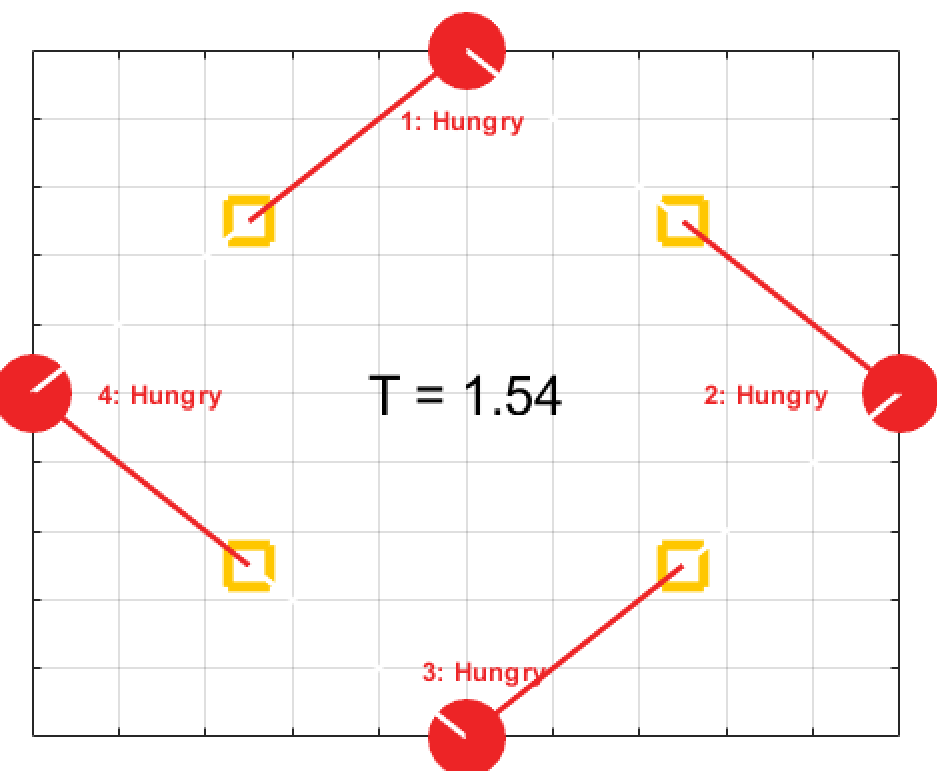
### Results

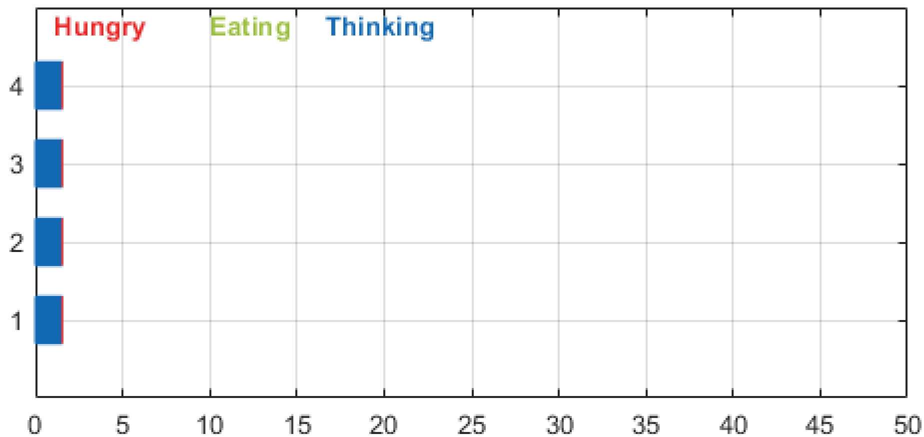
The first figure shows a Gantt chart of all four philosophers as they cycle between thinking, starving, and eating.

The second figure shows the instantaneous states of all four philosophers during the simulation. A line drawn from a philosopher (filled circle) to a fork (rounded rectangle) indicates that the philosopher has picked up that fork and hence the fork is unavailable for its neighbor.

### Deadlock

In this model, a deadlock can be reached if Philosopher 4 reverses his order of fork preference so that he picks up Fork 4 before Fork 1. This violates the above resource hierarchy constraint and simulating the model with this change will result in a deadlock as shown below.






The result above shows that each philosopher has picked up the right fork and everyone is waiting for the other fork to become available, causing a deadlock.

**References**

[1] Dining Philosophers Problem - Wikipedia (https://en.wikipedia.org/wiki/Dining_philosophers_problem)

## See Also

Resource Pool | Resource Acquirer | Resource Releaser | Entity Generator

## Related Examples

- "Resource Allocation Modeling"

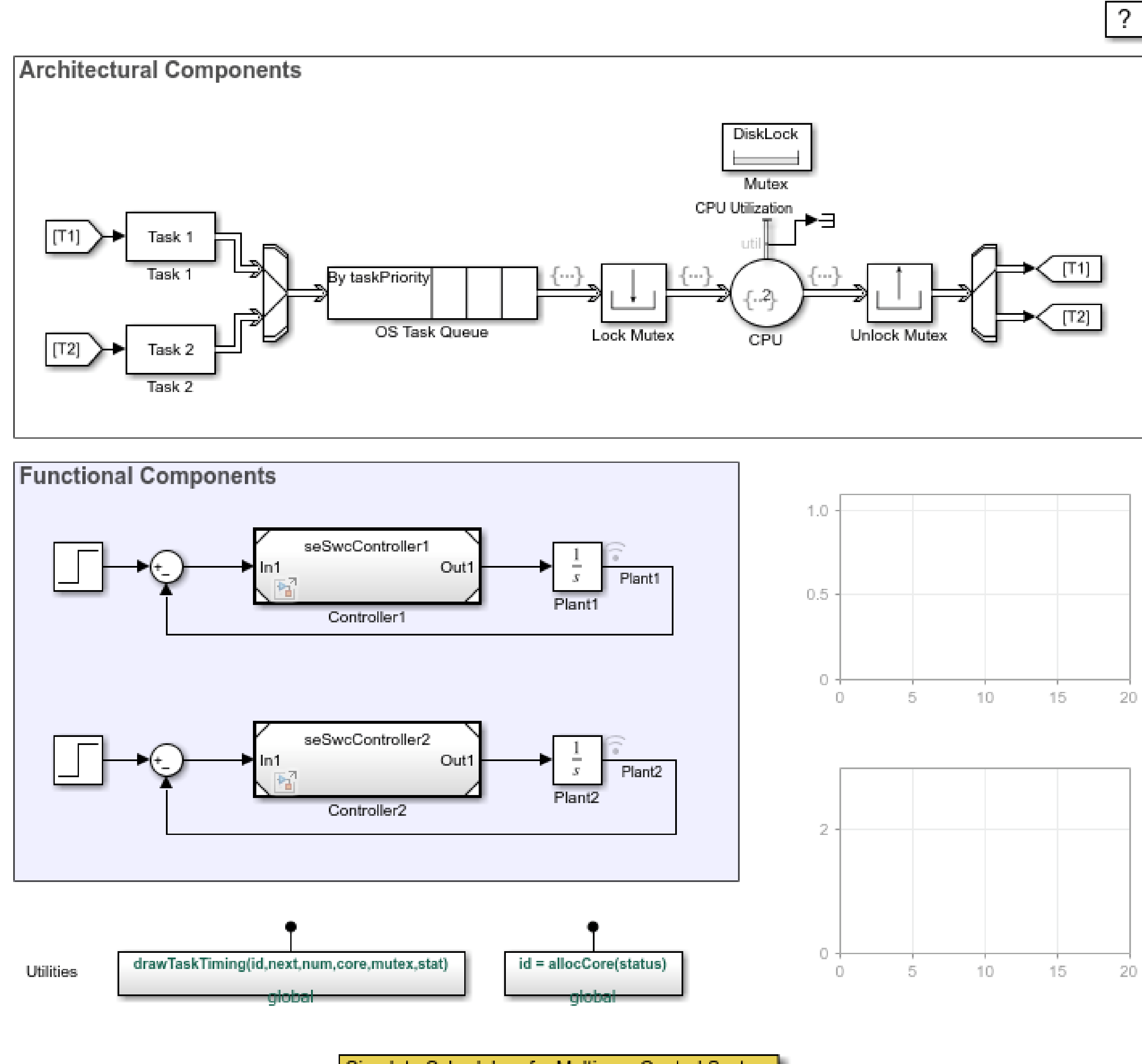
# Simulate Scheduler of a Multicore Control System

### Overview

This example shows how to model the task scheduling of a control application using SimEvents blocks. SimEvents expands Simulink with the capability to model and simulate architectural components of a real-time system.

The top model includes two areas of blocks:

- **Functional Components** includes two closed-loop systems. Each has a proportional controller operating a plant.
- **Architectural Components** includes SimEvents blocks modeling the tasks and scheduler of this control system.




### Modeling Tasks and Scheduler

This example models a controller as a Simulink exported function model. It maps execution of a controller to a software task that an operating system periodically schedules and executes. A task can be divided into multiple segments (or subtasks). Due to data dependencies, these segments must be executed in sequential order.

A task is specified with the following parameters:

- **ID:** Unique identifier of a task.
- **Period:** How frequently a task is instantiated for execution.
- **Priority:** Priority of the task (smaller value indicates higher priority).
- **List of runnable segments (functions):** Executables associated with each segment of the task. These executables are represented by Simulink functions of an exported function model.
- **Segment execution duration:** Time for a task segment to complete, if it is executed on a processor without interruption.
- **Needs disk i/o resource for each segment:** Whether a segment of the task requires the use of a mutex-protected shared resource (a hard disk).

For example, block Task 2 specifies a task for the second controller (block Controller2). The task includes two segments, "t2_run" and "t2_write", both modeled as Simulink functions in model seSwcController2. In these segments, "t2_write" requires the use of the mutex-protected shared resource.

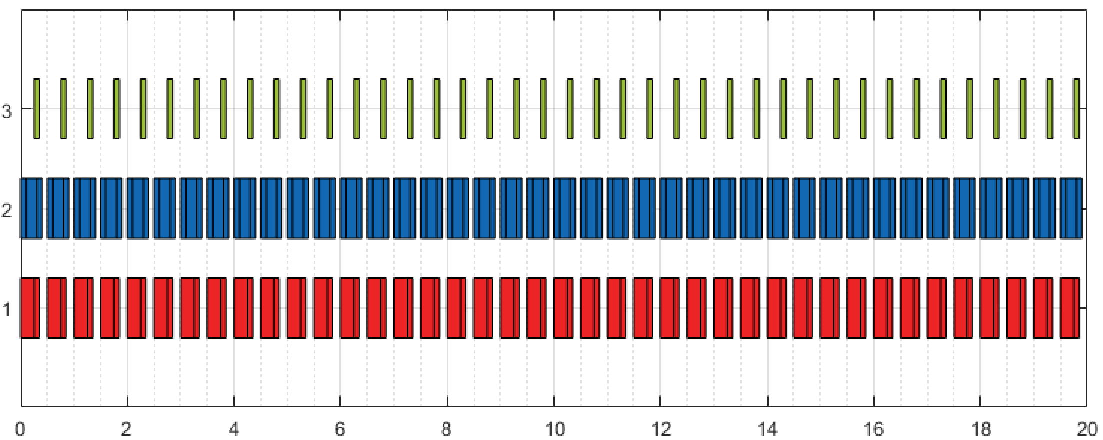
Scheduler of the operating system is modelled with the following components:

- **Task creation:** Block Task 1 and Task 2 create tasks and manage task states. A SimEvents entity represents an instance of a task. Properties of a task (such as its priority) are modelled as entity attributes.
- **Task queue:** Upon instantiation, a task joins a ready task queue, which is modelled by the Entity Queue block OS Task Queue. To simulate a non-preemptive priority-based scheduling policy, the queue block is configured to sort tasks by the taskPriority attribute.
- **CPU:** The processor of the system is modelled as an Entity Server block CPU. It accepts entities from the OS Task Queue and processes the entity for a duration as specified by the task's **Segment execution duration** parameter. At the end of this delay, the corresponding Simulink function of this task segment is called, as a part of the block's Service complete action.
- **Lock/unlock Mutex:** Before a task segment enters the block CPU, it must acquire the required resource at the preceding Lock Mutex block. After the task segment completes and exits the block CPU, the resource is released at the Unlock Mutex block.
- **Managing task states:** Blocks under the mask of Task 1 and Task 2 manage the run-time state of tasks. Upon completion of a task segment, if the task has subsequent segments to execute, the task is routed back to the OS Task Queue. Otherwise, this task instance is completed and discarded.

### Results and Displays

The block CPU is configured with two cores. Simulating the model generates the following Gantt chart.

- The higher priority task, Task 2 (red bars), is scheduled to core 1 (y = 1).
- The lower priority task, Task 1 (blue bars), is scheduled to core 2 (y = 2).
- The second segment of Task 2 uses the mutex DiskLock. Green bars indicate the usage (y = 3).



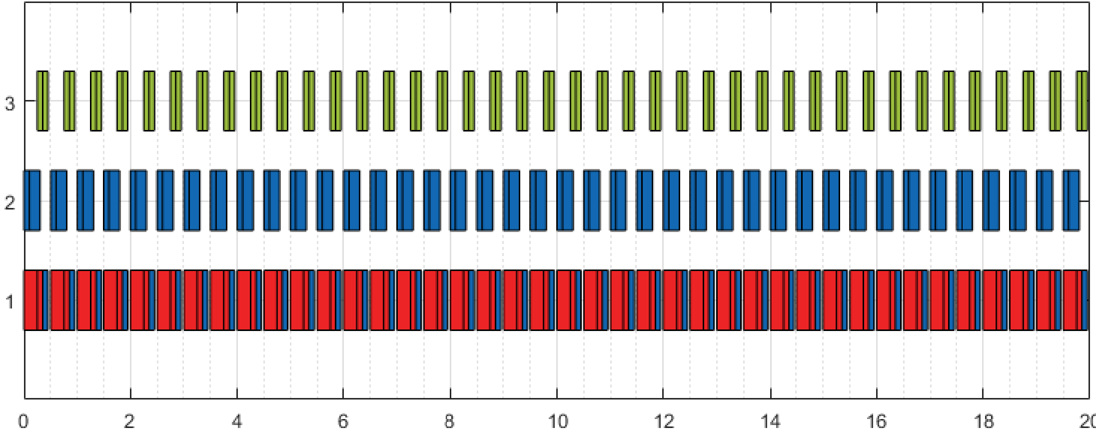

#### Explore the Model

Change the following parameters and explore how task schedules and controller performance changes with re-configured architectural parameters.

- In block CPU, configure parameter Capacity to change the number of cores.
- In blocks Task 1 and Task 2, configure parameters such as Period and Priority to change task specifications.

For example, if we change **Need disk i/o resource for each segment** parameter of the Task 1 block to [0 0 1], the t1_write segment of Controller 1 must acquire mutex DiskLock before it can start to run. Simulation generates a Gantt chart that illustrates that change.

- Both tasks have segments that use mutex DiskLock, as indicated by green bars (y = 3).
- The third segment of Task 1 now must execute in a serial fashion with the second segment of Task 2 (see y = 1), because both segments share the mutex DiskLock.



#### Related Examples

- Develop Custom Scheduler for a Multicore Control System

## See Also

Entity Server | Queue | Resource Pool | Resource Acquirer | Resource Releaser | Entity Generator

## Related Examples

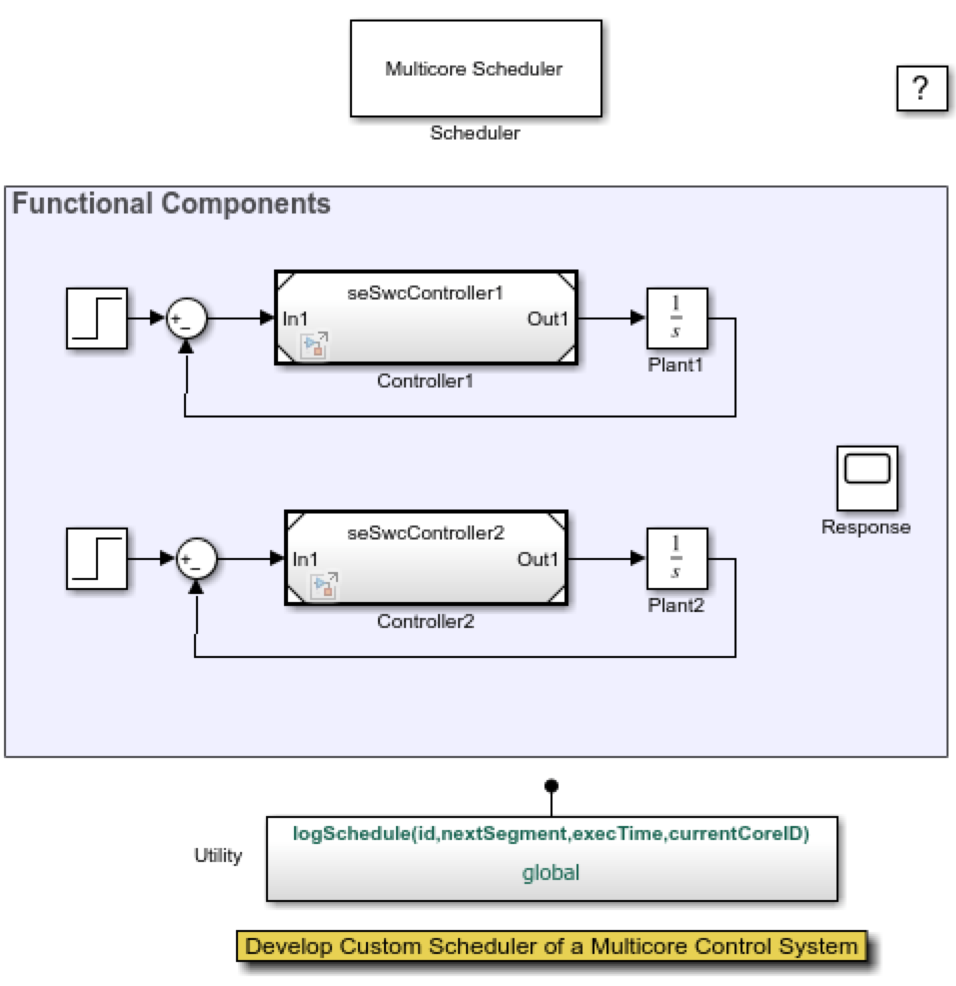
- “Create a Discrete-Event Model”
- “Manage Entities Using Event Actions”
- “Entity Priorities” on page 1-36
- “Resource Allocation Modeling”

# Develop Custom Scheduler of a Multicore Control System

### Overview

This example shows how to model a customer scheduler using the SimEvents MATLAB Discrete-Event System block. The model includes a Scheduler block that can simulate a multicore system with an arbitrary number of cores, tasks, and mutually exclusive resources.

The model configures the Scheduler block to process tasks of closed-loop control systems. The simulation measures the performance of these control systems, and provides metrics of the run-time environment, such as latencies and resource contingencies. These results can help designers of control systems develop architectural specifications for their functional components.




Copyright 2015 The MathWorks, Inc.

### Creating Custom Scheduler in MATLAB

The Scheduler block of the root model is developed primarily as a MATLAB discrete-event system. MATLAB file seSchedulerClass contains the implementation of the corresponding discrete-event System object.

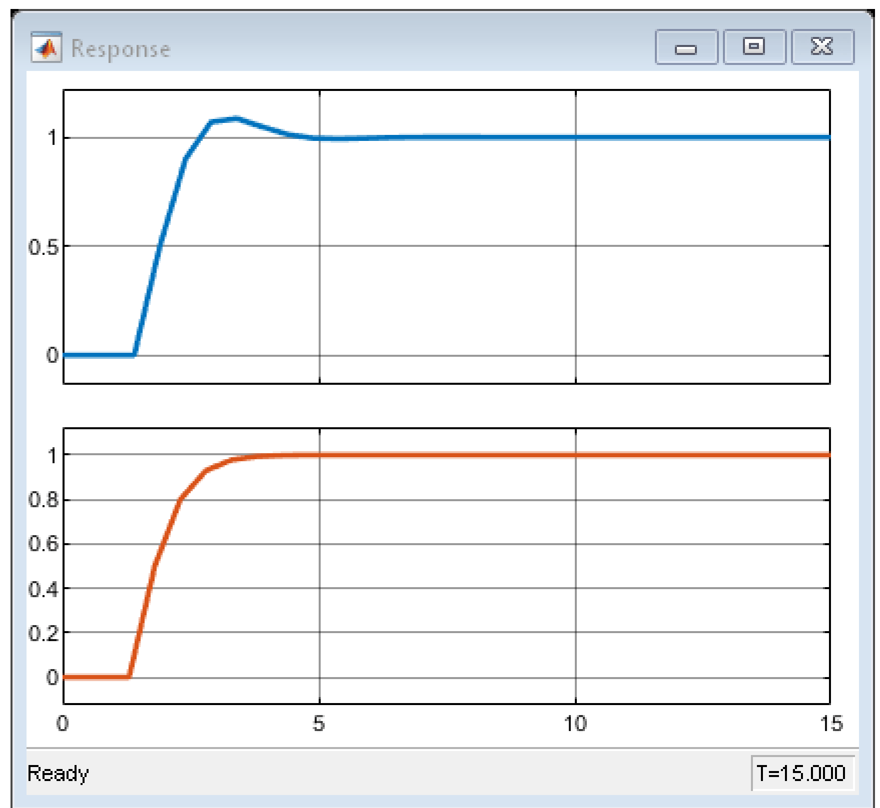
The example models a controller as a Simulink exported function model. Execution of a controller is mapped to a software task that is scheduled to execute periodically. A task can be divided into multiple segments (or subtasks). Due to data dependencies, these segments must be executed in sequential order.

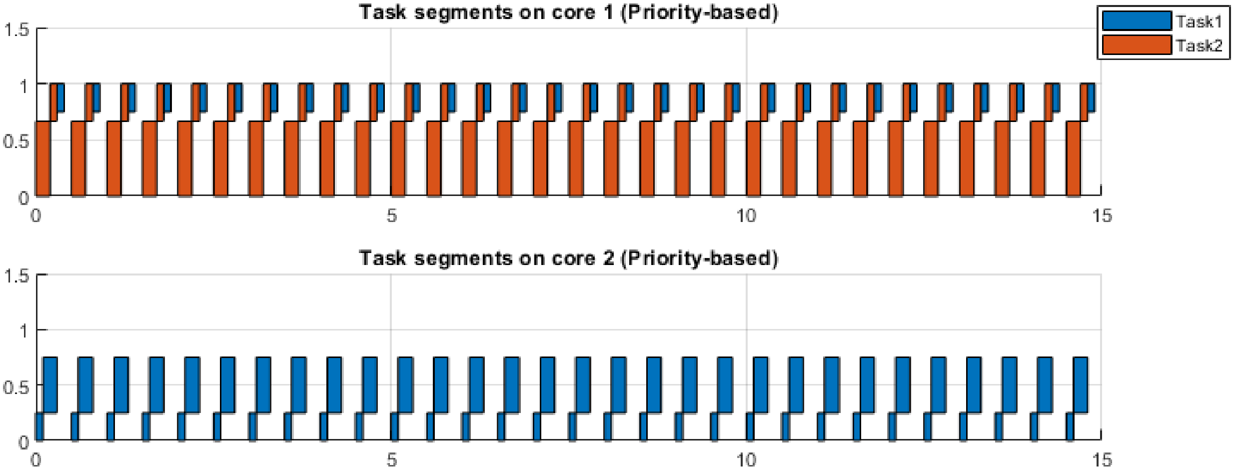
The Scheduler block includes the following parameters:

- **Number of cores:** Number of cores available for the operating system to use.
- **Scheduling policy:** Select either "Priority-based" or "Round robin" as the scheduling policy of the operating system. Priority-based scheduling sorts and executes tasks in a prioritized order. Round robin policy allows tasks to equally take turns.
- **Number of tasks:** Number of tasks in this operating system.
- **Task periods:** How frequently each task is instantiated for execution.
- **Task priorities:** Priority of each task (smaller value indicates higher priority).
- **Number of segments in each task:** Number of segments (subtasks) a task has.
- **Simulink function for each segment:** Executables associated with each segment of a task. These executables are represented by Simulink functions of an exported function model.
- **Execution durations of each segment:** Time for a task segment to complete, if it is executed on a processor without interruption.
- **Number of mutually exclusive resources:** Number of mutually exclusive resources of the operating system. One task at a time can acquire and use the resource. Operating system uses mechanisms such as mutexes to manage these resources.
- **Use of resources by each task:** Cell vector. Each element of the vector indicates the use of resources by a task.

### Comparing Different Core Allocations

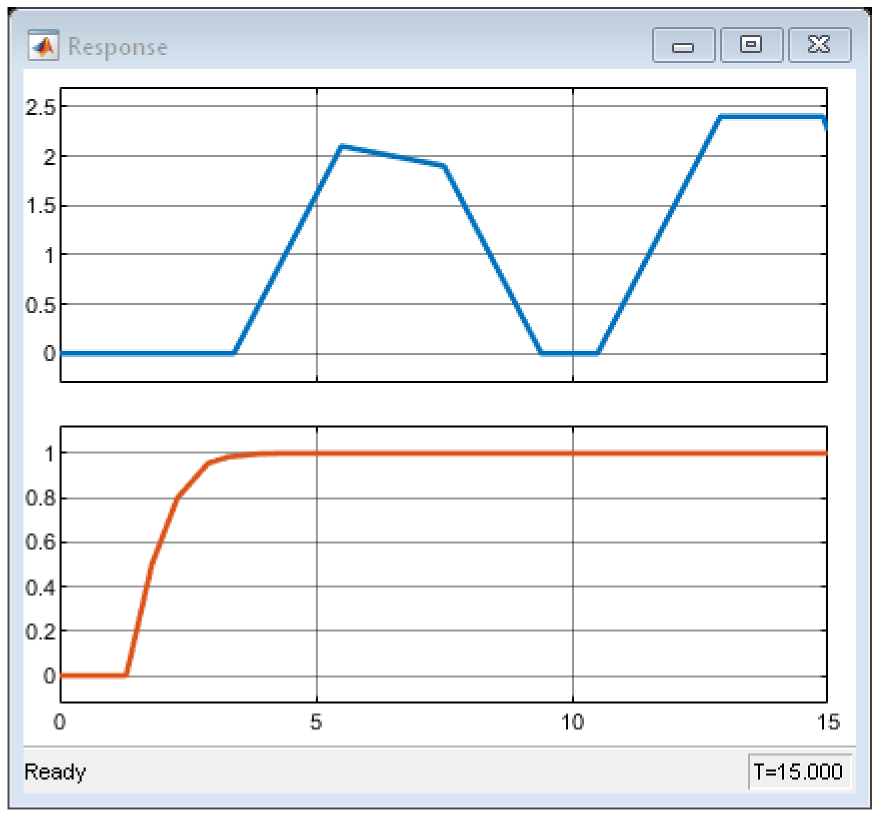
The Scheduler block allows you to assign an arbitrary number of cores and explore how that impacts system performance. We begin with a scenario where two cores have been assigned to execute the two control tasks. With sufficient processing capacity, both closed-loop control systems perform well in response to set point changes.

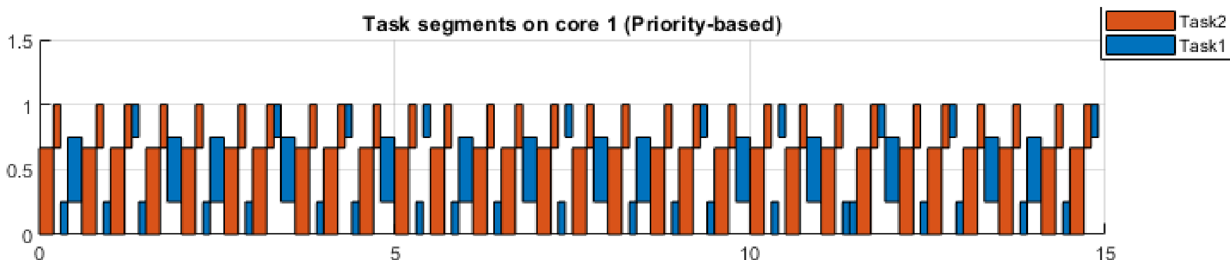



The timing diagram of the scheduler indicates that control tasks are concurrently processed by both cores, with cores having medium and balanced utilizations.

In comparison, when only one core is assigned, the performance of Controller1 degrades due to task overruns (see Plant1). The timing diagram clearly indicates such task overruns, and the significantly increased latencies.

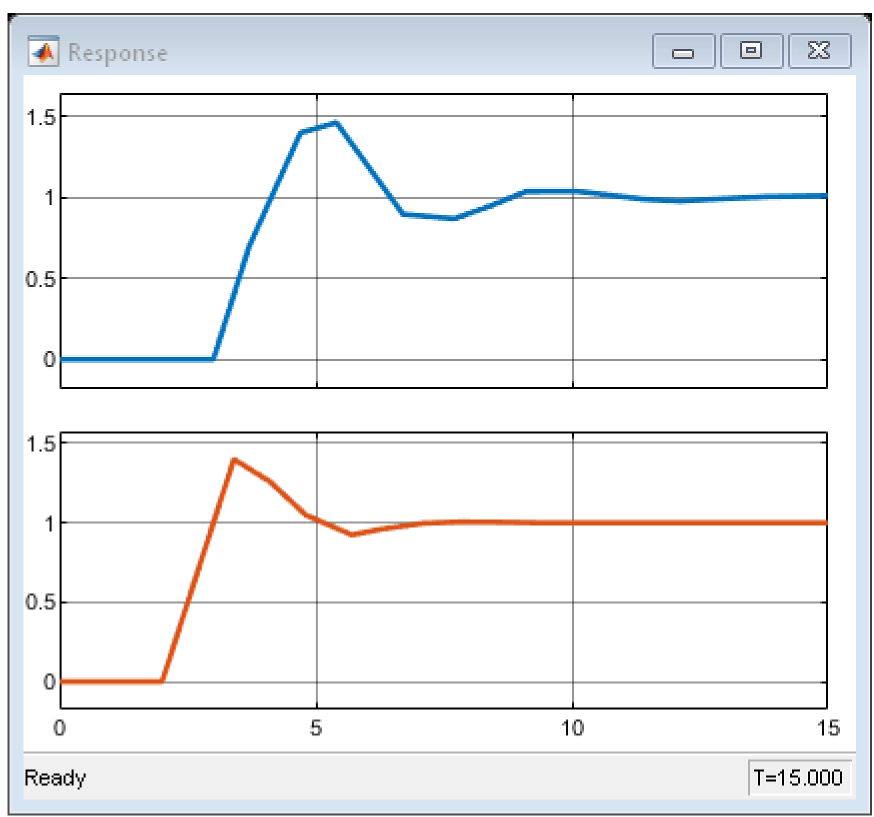


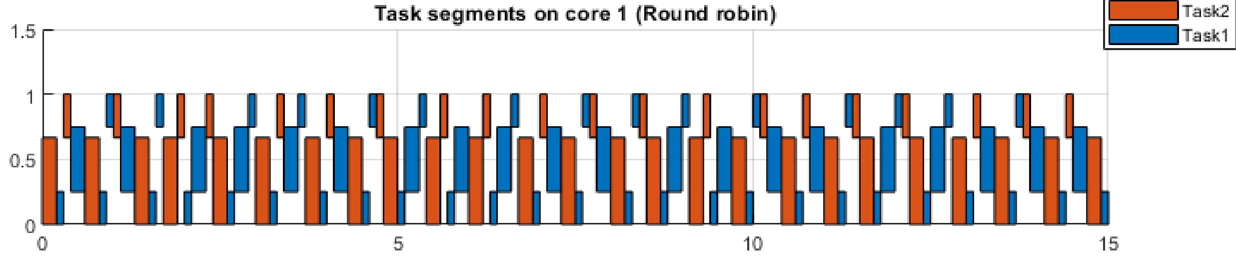




Notice that the performance of control task 2 remains unchanged. This is because the scheduler applies the priority-based policy where processing capacity is maximally assigned to high priority tasks.

**Comparing Different Scheduling Policies**

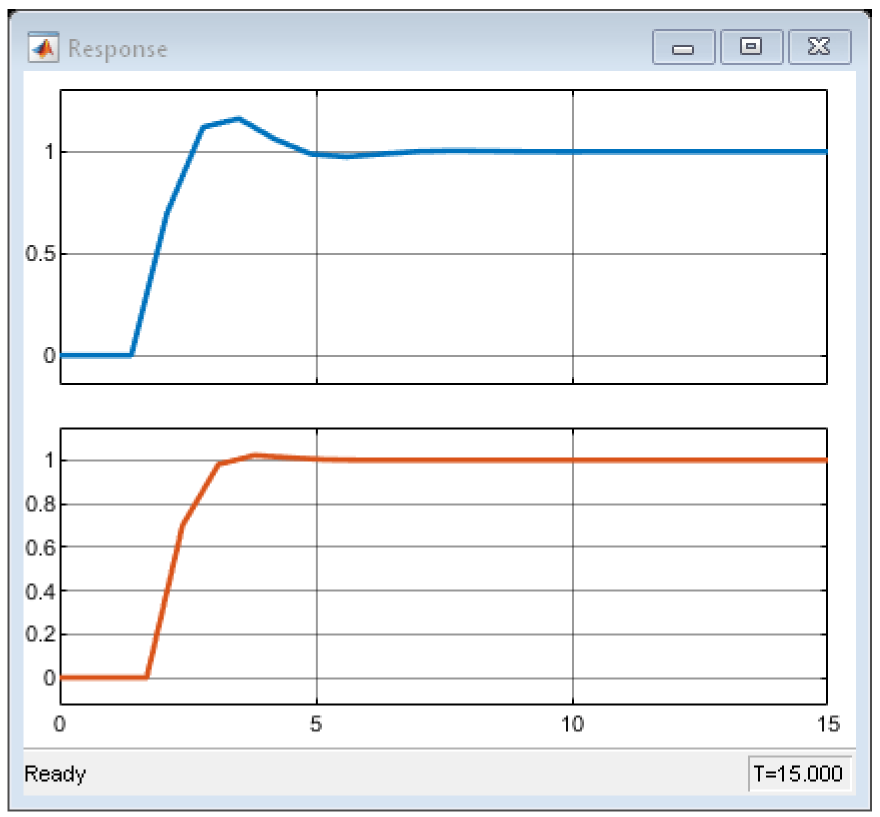
At this point, if the Scheduler switches to use a round robin scheduling policy, the control system performs differently. Compared to the previous case, where processing capacity remains the same, Plant 1 becomes stable, with the cost of degrading the performance of Plant 2. This change is due to the fact that the round-robin policy evenly assigns the processing capacity among all tasks.

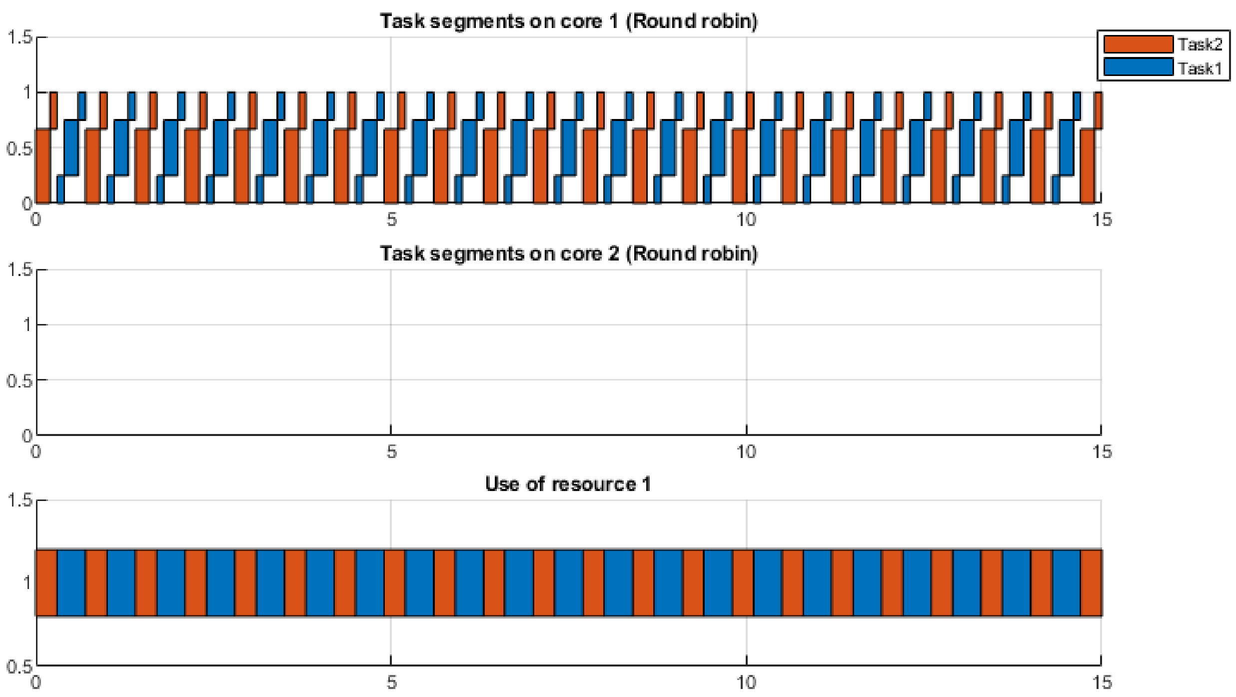



## Comparing Different Resource Allocations

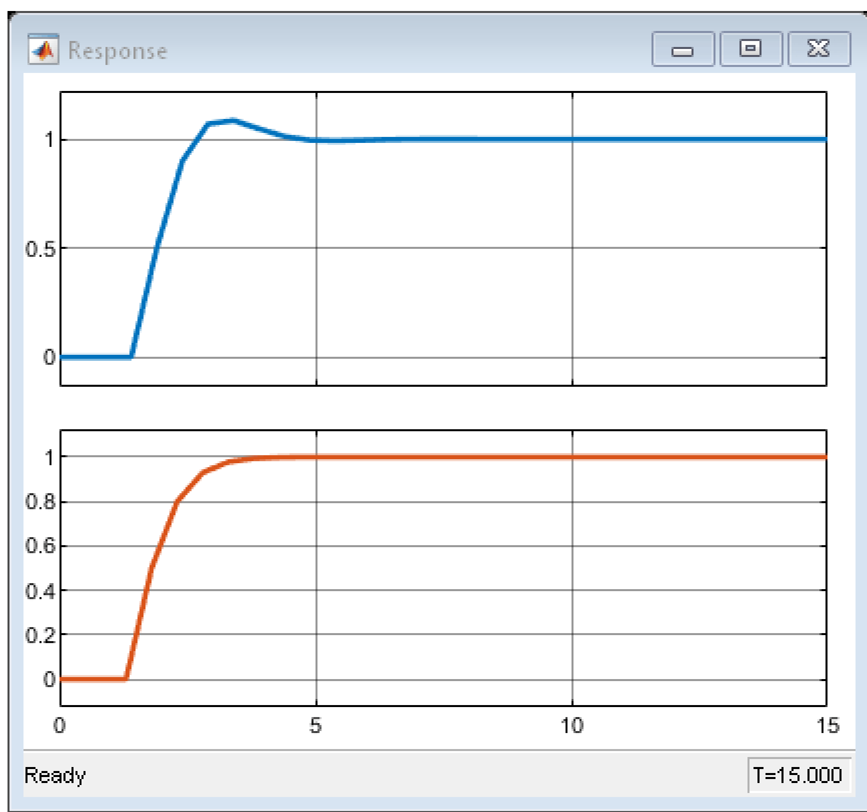
Now let us configure the scheduler back to use two cores, and explore different resource allocation schemes. We add a resource that can be shared by tasks in a mutually exclusive fashion.

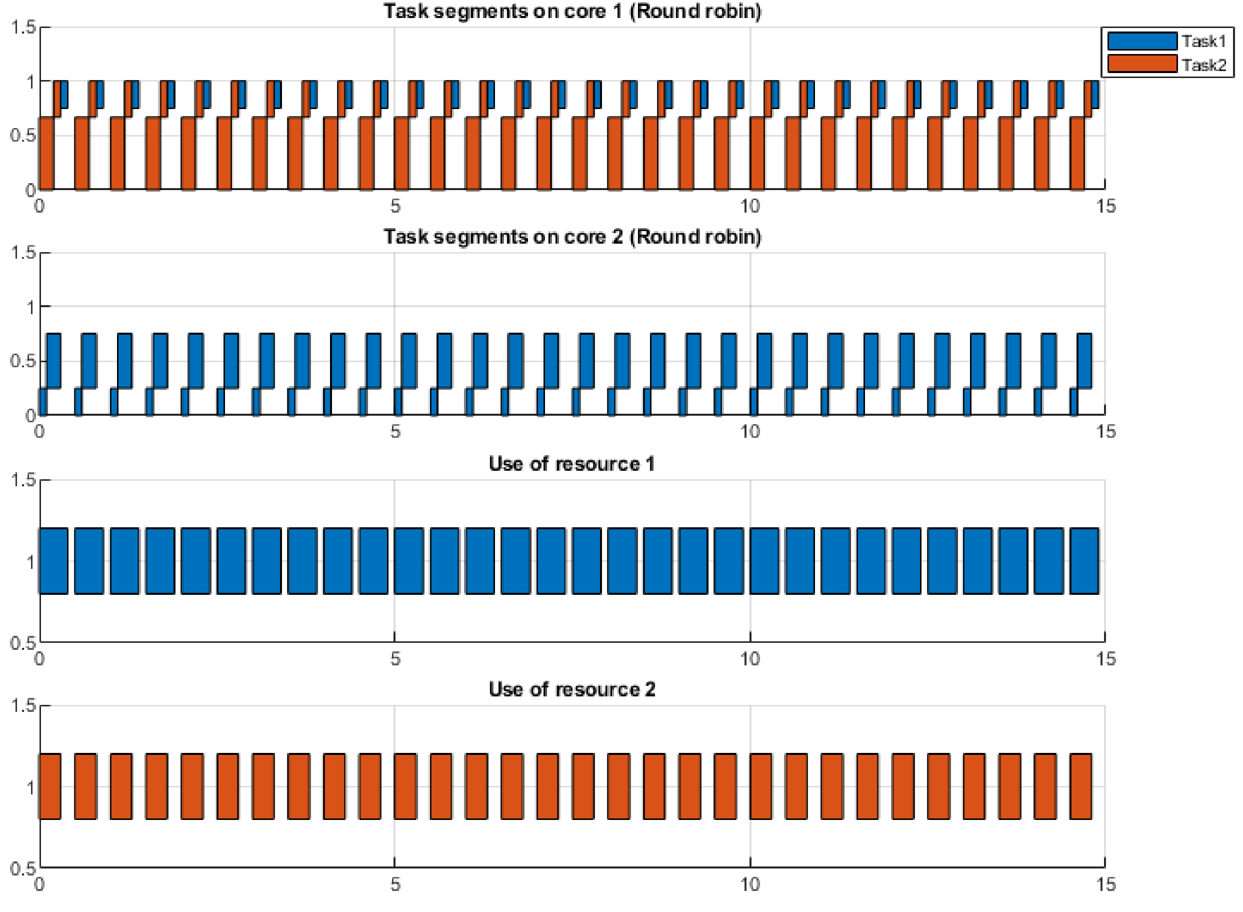

Response
Ready
T=15.000


Task segments on core 1 (Round robin)
Task2
Task1
Task segments on core 2 (Round robin)
Use of resource 1

As indicated by the timing diagram, although concurrent execution is allowed with two cores, the tasks are processed in a sequential fashion. Only one core is in use. This is because a task must wait for the required resource before it can be processed.

You can eliminate such resource contingency by assigning more resources. Let us configure the scheduler block to use 2 resources, and allow a task to have a dedicated resource.






With each task having its own resource, tasks are processed concurrently.

**Related Examples**

- Simulate Scheduler of a Multicore Control System

## See Also

Entity Server | Queue | Resource Pool | Resource Acquirer | Resource Releaser | Entity Generator

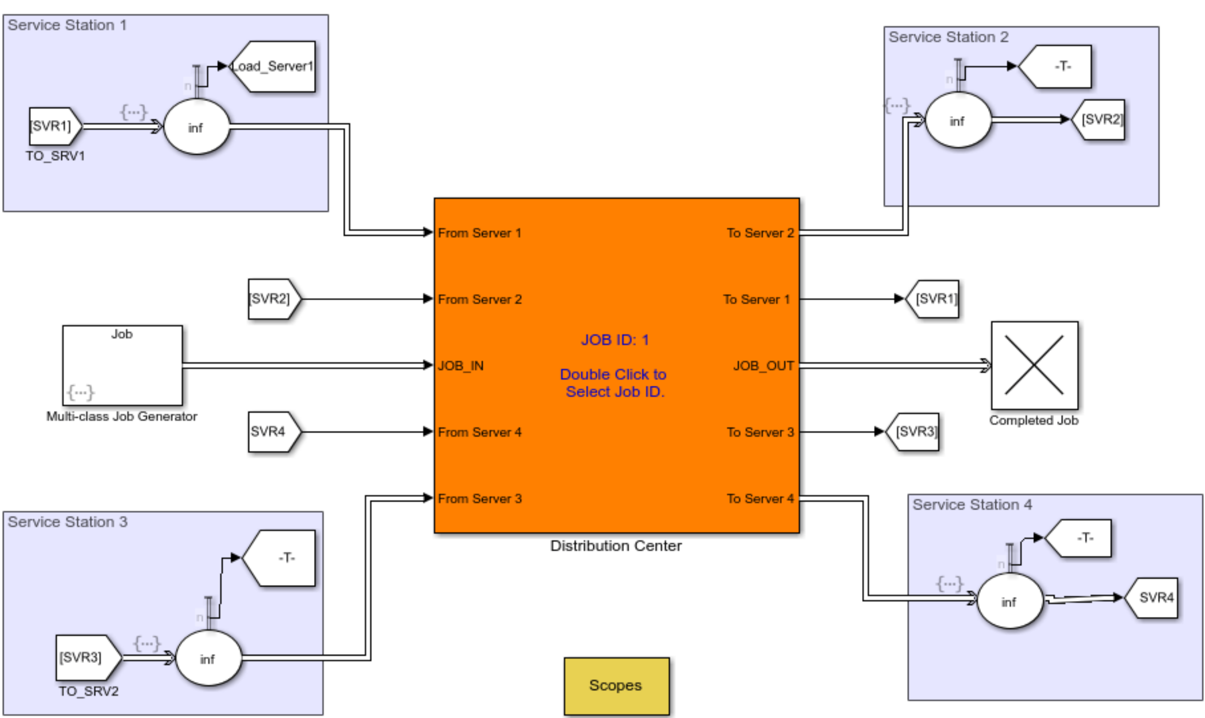
## Related Examples

- “Create a Discrete-Event Model”
- “Manage Entities Using Event Actions”
- “Entity Priorities” on page 1-36
- “Resource Allocation Modeling”

# Distributing Multi-Class Jobs to Service Stations

## Overview

This example shows how to model a central resource that manages distributed processing according to an explicit formula. The example describes a distribution center that manages a series of processes that each job undergoes, with each job carrying information about the series of processes that it must undergo. One of the applications where this could be useful is when you want to model a central dispatcher that routes calls from one call processing station to another.




Copyright 2007-2016 The MathWorks, Inc.

## Multiple-Class Job Generation

This generator produces jobs that possess these attributes:

- `JobClass` - A value of 1, 2, or 3, which determines the value of `ServiceProcess` and `ServiceTime`
- `JobID` - An integer between 1 and 15. This acts as an identifier for each job. This can be used to track a job as it moves through the different service stations.

- `LastServiceLocation` - 0 initially, to be modified during the simulation as the job visits different service stations
- `JobServiceStatus` - A vector of 0s initially, to be modified during the simulation as the job completes different processes
- `ServiceProcess` - A vector that lists the processes that the job undergoes
- `ServiceTime` - A vector that lists the duration of each process that appears in `ServiceProcess`
- `CurrentStep` - 1 initially, to be modified during the simulation as the job progresses through its series of processes

#### Distribution Center Subsystem

This subsystem uses the vector elements of each job's `ServiceProcess` attribute to route the entity to the correct service station. The distribution center also updates information that the job carries about its current state.

During the simulation, each job follows a trajectory from the distribution center to a service station, back to the distribution center, to a (possibly different) service station, and so on. The particular trajectory depends on the `ServiceProcess` attribute value.

#### Service Stations

Each of multiple Service Stations processes the arrived job based on the JobClass and CurrentStep of the job. Upon the completion of the service, the job returns to the distribution center.

#### Experiment with the Model

To see the service history for a particular job in Scopes/Service History for Jobs, enter the job ID in the distribution system 'Display service history for jobs with ID' parameter.

## See Also

Entity Server | Queue | Resource Pool | Resource Acquirer | Resource Releaser | Entity Generator

## Related Examples

- “Create a Discrete-Event Model”
- “Manage Entities Using Event Actions”
- “Entity Priorities” on page 1-36
- “Resource Allocation Modeling”

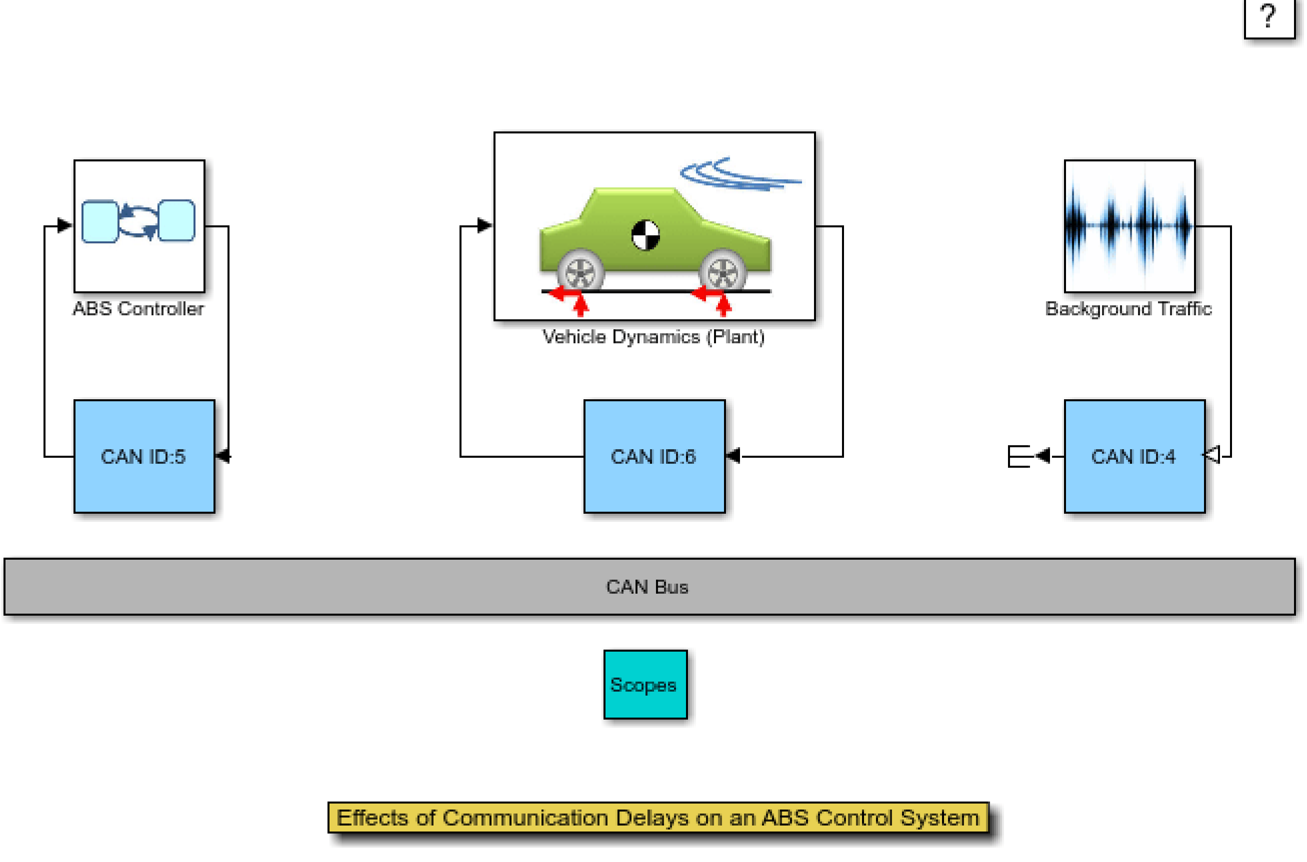
# Effects of Communication Delays on an ABS Control System

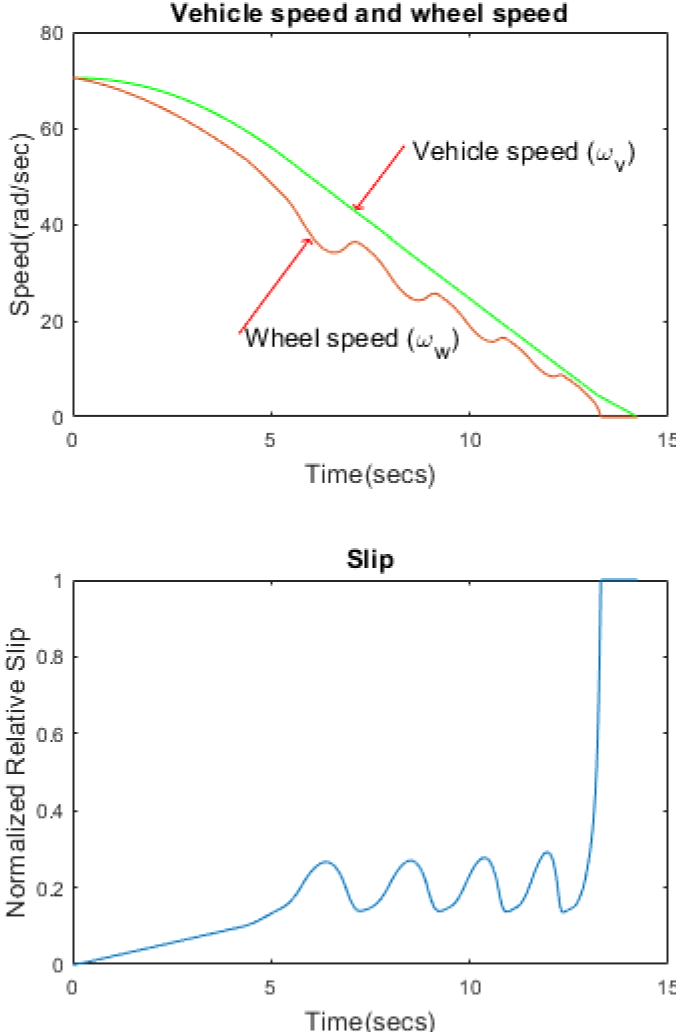
### Overview of Example

This example shows how stochastic network traffic causes timing latency and uncertainty in an anti-lock braking system (ABS) that uses Control Area Network (CAN) communications. The model is representative of a real-world heavily-loaded network and also illustrates a domain-specific model of a distributed system. By including real-world timing effects in a model, you gain confidence about the behavior and robustness of your design before you test it in hardware.

### No Traffic

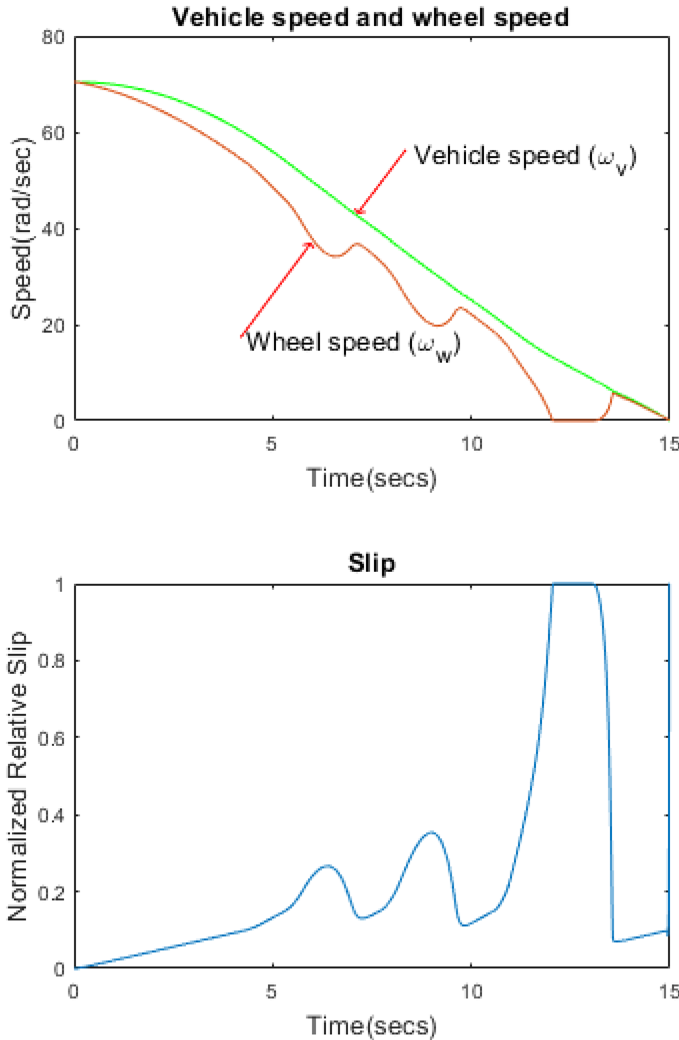
We begin with the ideal scenario of an anti-lock braking system using CAN communications with no background traffic. In this model, we simulate a CAN with no background traffic by manually setting the output of the Manual Switch that is contained in the Background Traffic subsystem block to the "OFF" position, before running the simulation. In this ideal scenario, the software simulates a communication system with a steady rate of network utilization over time and with no delays in message delivery. These ideal conditions result in excellent slip response from the ABS system relative to decreasing wheel speed.





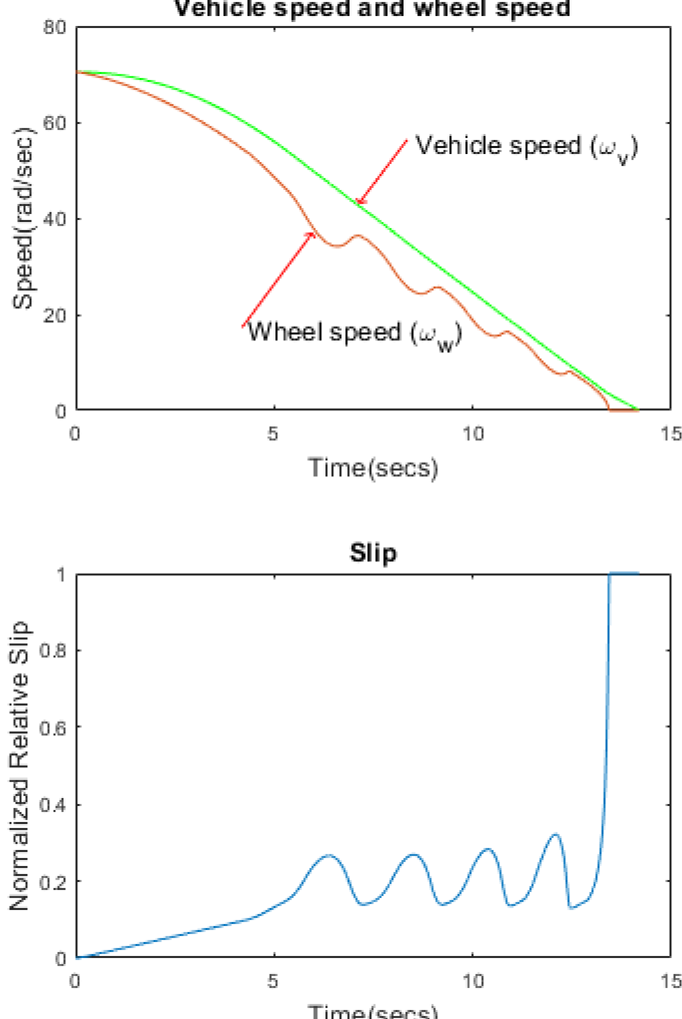

**With Traffic**

Next, we simulate the more realistic scenario of a CAN with some stochastic network traffic. This background traffic results in message delivery delays on the network. To simulate this, we manually set the Manual Switch contained in the Background Traffic subsystem block to the "ON" position. By setting the Manual Switch to this position, we allow the Step Function block that is also contained in the Background Traffic subsystem block to introduce traffic to the network. In this model, the Step Function block is configured to output a value at simulation time, T=6 seconds. If we contrast the updated simulation results with those of our previous ideal scenario, we see that the introduction of message delivery delays to the network results in much poorer slip response from the ABS system relative to the wheel speed.




### Re-prioritization of the CAN Messages

A CAN network processes messages from distributed nodes on the network based on the message priority of each node. In our model, we define a message priority of 5 for the ABS Controller subsystem, 6 for Vehicle Dynamics subsystem, and 4 for Background Traffic subsystem. This means that the CAN network processes messages from the Background Traffic subsystem first. We saw previously that the introduction of background traffic on the network resulted in much poorer slip response from the ABS system. To reduce the negative impact of background traffic, we adjust the CAN message priority of the ABS Controller subsystem and Vehicle Dynamics subsystem to have a higher priority than the Background Traffic subsystem. This change results in a reduced message delivery delay on the network and an improved slip response from the ABS system relative to the wheel speed.




#### Conclusion

This example shows an anti-lock braking system using CAN communications and highlights the negative effect that increased network utilization can produce on latency and response times.

## See Also

Entity Server | Queue | Resource Pool | Resource Acquirer | Resource Releaser | Entity Generator

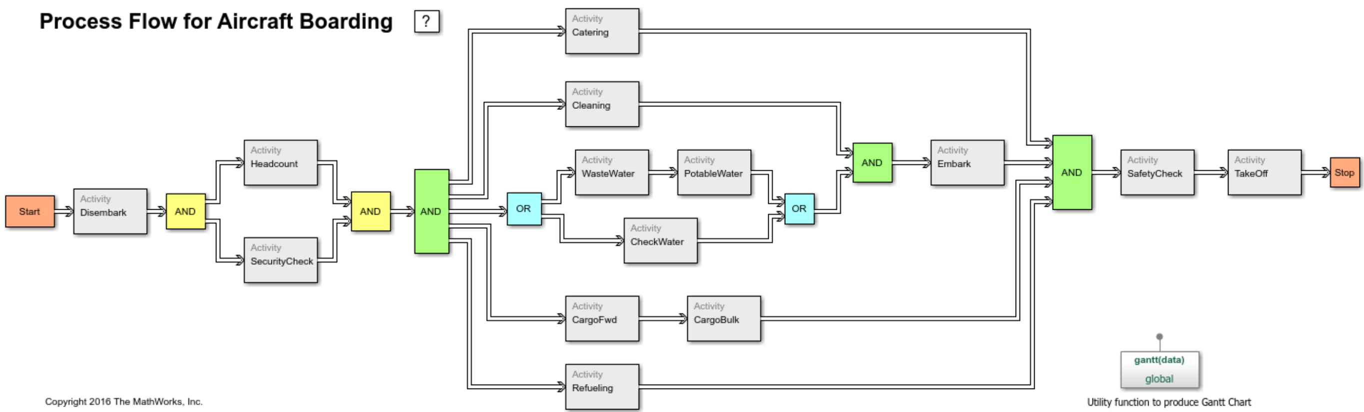
## Related Examples

- “Create a Discrete-Event Model”
- “Manage Entities Using Event Actions”
- “Entity Priorities” on page 1-36
- “Resource Allocation Modeling”

# Aircraft Boarding Process Flow

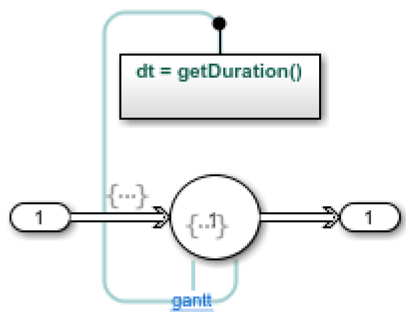
### Description

This example shows how to use SimEvents to model a process such as the boarding of an aircraft. The process consists of multiple activities such as "Disembark", "SecurityCheck", "Refueling" etc. Some activities can be done concurrently, as represented by multiple parallel paths using AND blocks. Some activities are mutually exclusive and these are represented as output paths using OR blocks. Each activity takes up non-zero time. You can use such a model to study various aspects of a process, such as bottlenecks, resource contention, latencies, etc. The model generates a single entity at the start of simulation. This entity represents the control flow in the process. The position of the entity in the model determines which activity is currently running.




### Activity

Use an Entity Server block to model an activity. The service time is a randomized number with a specified mean value. Activities can be sequential, concurrent, or mutually exclusive with respect to each other.




### Sequential Activities

Activities that are chained to each other are considered sequential. This means that the first activity (or set of activities) must be completed before the second activity can start.

### Concurrent Activities

Activities which can be executed simultaneously are concurrent activities. You can use an Entity Replicator to replicate the input control entity into N output entities that will flow concurrently into

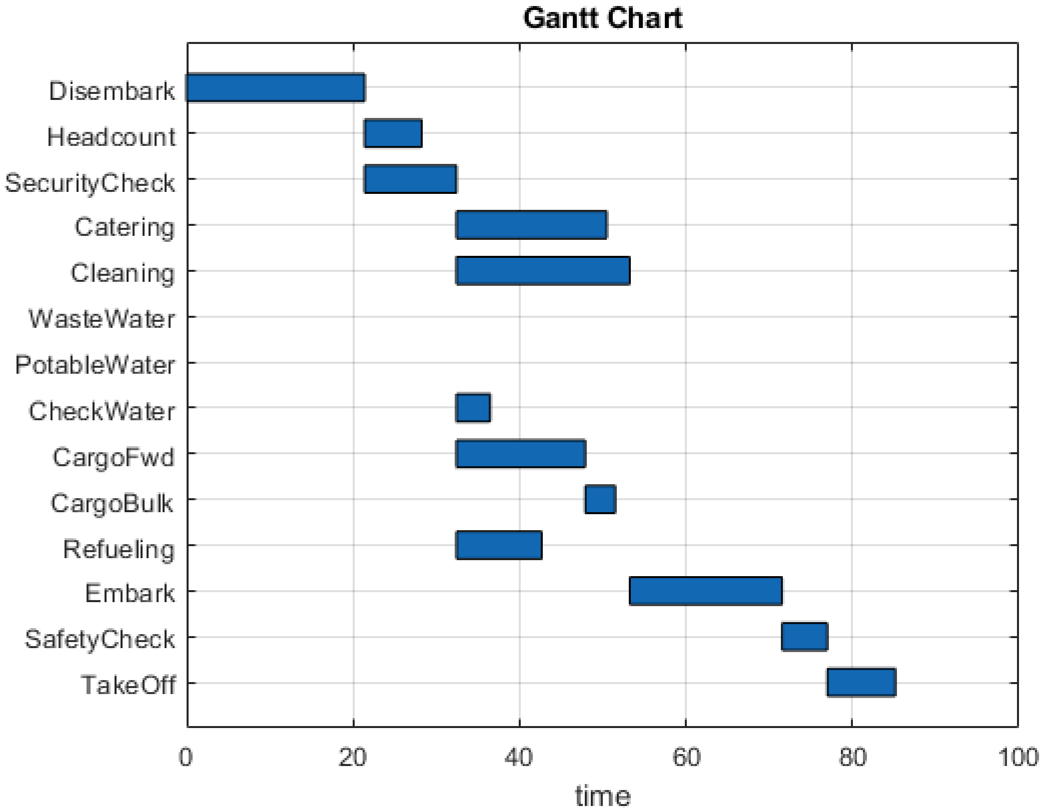
parallel activities. This model uses a masked Entity Replicator block as the AND output block. A synchronization can be described for all activities when concurrent control paths are completed. Use an AND input block to model such synchronization or "join".

### Mutually Exclusive Activities

Two activities of which only one can ever be executed during a given scenario are mutually exclusive. You can use the Entity Output Switch block to model an OR construct in which mutually exclusive activities can be placed on each output. The control flow entity will be routed to one of the N outputs, thus ensuring that only one of the mutually exclusive activity paths is executed.

### Simulation Results

This model produces a Gantt chart of the simulation that shows each activity and how long it took to execute. The Gantt chart shows how concurrent activities are executed in parallel, while sequential activities are only executed when the preceding activities are completed.

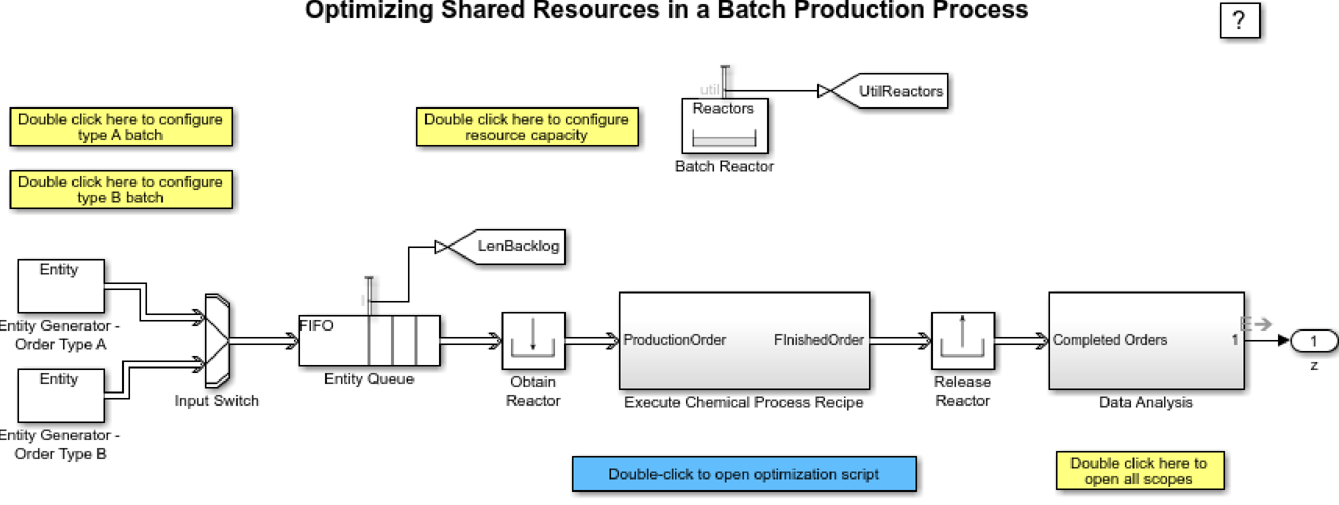


# Optimization of Shared Resources in a Batch Production Process

### Overview

This example shows how to model and optimize the use of shared resources in a system, to identify resource deficiencies and improve capacity planning. The example is based on a batch production process, where production orders are processed only according to the availability of batch reactors. In the example, SimEvents® entities represent both the production orders of the manufacturing process, and the batch reactors that are required to process them. Later in the example, we will find the optimal resource capacities of the system by applying the Genetic Algorithm solver of MATLAB Global Optimization Toolbox.

```
modelname = 'seBatchProduction';
open_system(modelname);
scopes = find_system(modelname, 'LookUnderMasks', 'on', 'BlockType', 'Scope');
cellfun(@(x)close_system(x), scopes);
set_param(modelname, 'SimulationCommand', 'update');
```




### Structure of Model

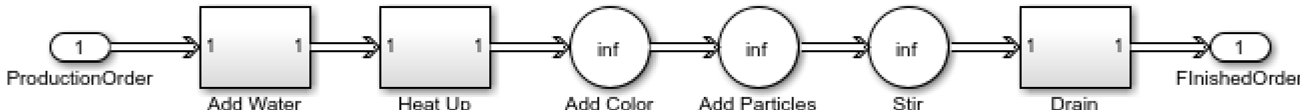
At the top level of the model, the Entity Generators simulate the generation and backlog of production orders by generating entities that represent production orders. When a new entity is generated, the Obtain Reactor block requests a batch reactor to process the order. After the Execute Chemical Process Recipe subsystem completes the order according to a specified chemical process recipe, the block labeled Release Reactor releases the batch reactor back into the pool of resources, where it is now available to process a new order. The Data Analysis subsystem analyzes data related to completion of production orders.

### Shared Resources in the Production Process

The Execute Chemical Process Recipe subsystem simulates the chemical process to produce sol (a type of colloid). A six-step recipe models the main operations in sol production. Execution of these steps requires different resources. A batch reactor provides built-in capabilities to execute steps like

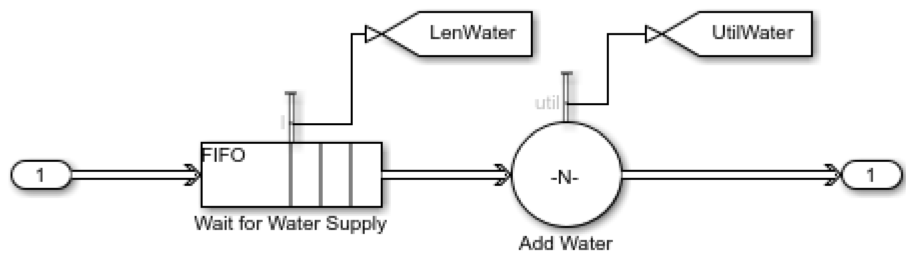
adding color, adding particles and stir. Thus the resources required by these steps do not need to be modeled separately. On the other hand, the steps to add water, heat up and drain require extra resources. These resources are shared by all the batch reactors and are limited by the capacity of the production system.

```
open_system([modelname '/Execute Chemical Process Recipe']);
```




For example, when water usage reaches the full capacity, water pressure is too low for another batch reactor to access. In this case, production in that reactor pauses until the water supply becomes available again. In the Execute Chemical Process Recipe subsystem, the example models such a resource sharing process with a Queue block labeled **Wait for Water Supply** and an Entity Server block labeled **Add Water** in the Add Water subsystem. The **Capacity** parameter of the Entity Server block models the capacity of the water supply. During simulation, the number of entities in the Queue block indicates the number of batch reactors waiting for water. The number of entities in the Server block represents the number of batch reactors accessing water.

```
open_system([modelname '/Execute Chemical Process Recipe/Add Water']);
```



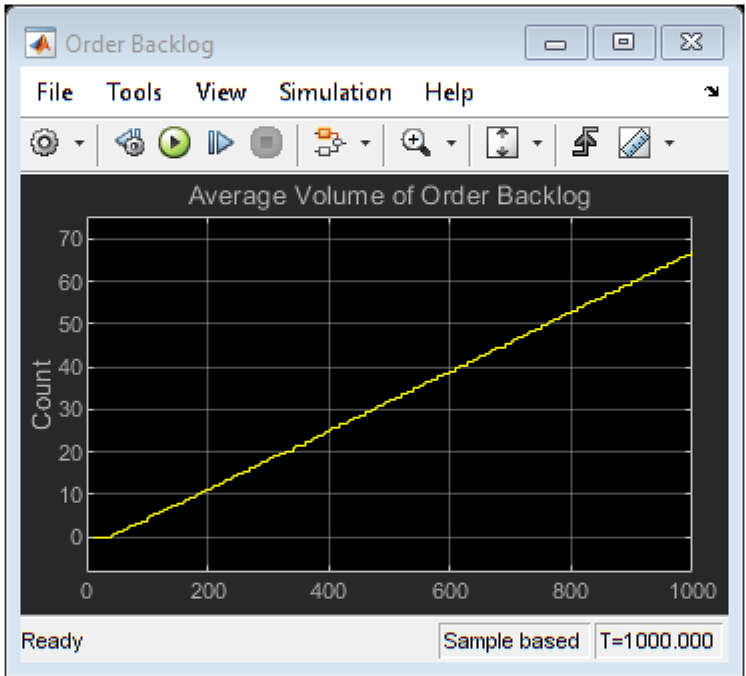

The modeled batch production process is capable of producing two types of batches: type A and type B. Although the main steps required to produce either batch are the same, the chemical process recipes are different. For example, the recipe to produce type B requires more water, so the step to add water takes more time to complete.

### Results and Displays

During simulation, the Data Analysis subsystem displays several results to show the performance of the production process.

The most illustrative result here is the first one, **Average number of orders in backlog**, which represents the wait time for orders as the system struggles to keep up with inflow.

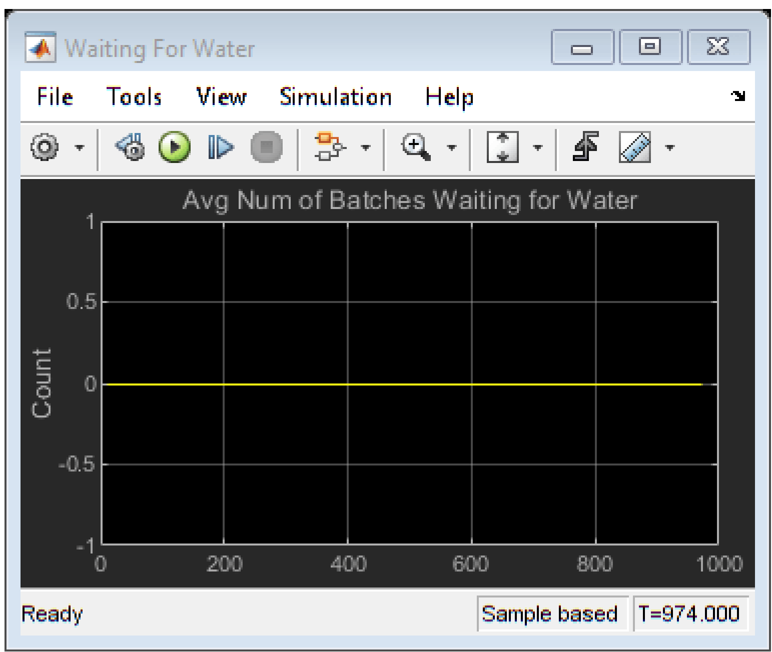
```
sim(modelname);
open_system([modelname '/Data Analysis/Order Backlog']);
```

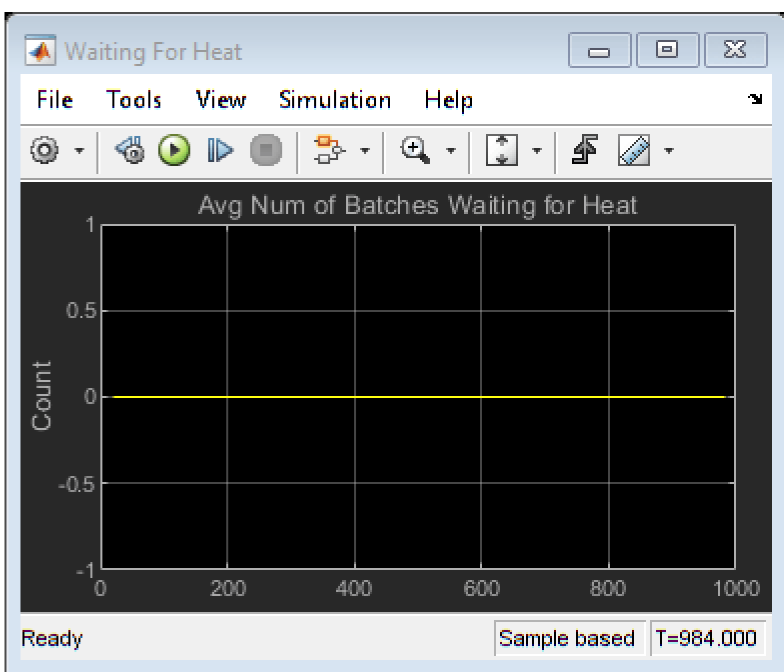


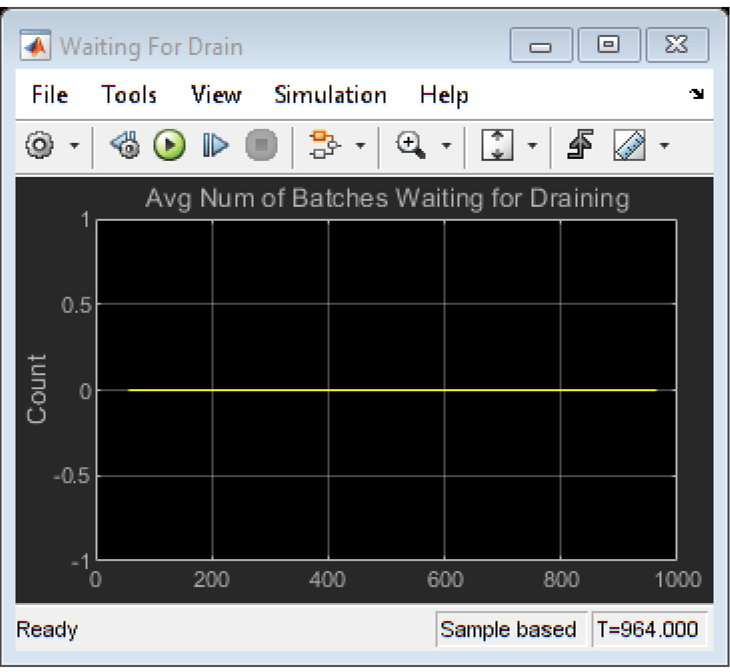

Other results of the system include the following and can be seen in the Data Analysis subsystem:

- Average number of batches waiting for water
- Average number of batches waiting for heat
- Average number of batches waiting for draining
- Utilization of batch reactors
- Utilization of water supply
- Utilization of heat supply
- Utilization of draining facility
- Throughput of type A batch
- Throughput of type B batch

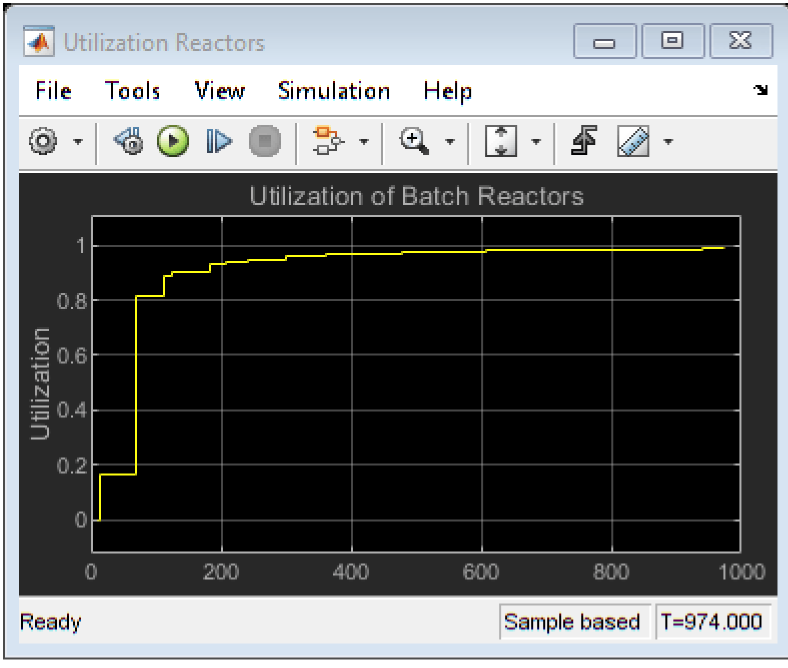
```
open_system([modelname '/Data Analysis/Waiting For Water']);
open_system([modelname '/Data Analysis/Waiting For Heat']);
open_system([modelname '/Data Analysis/Waiting For Drain']);
open_system([modelname '/Data Analysis/Utilization Reactors']);
open_system([modelname '/Data Analysis/Utilization Water']);
open_system([modelname '/Data Analysis/Utilization Heaters']);
open_system([modelname '/Data Analysis/Utilization Drains']);
open_system([modelname '/Data Analysis/ThroughputA']);
open_system([modelname '/Data Analysis/ThroughputB']);
```


Waiting For Water
File Tools View Simulation Help
Avg Num of Batches Waiting for Water
Count
1
0.5
0
-0.5
-1
0 200 400 600 800 1000
Ready
Sample based
T=974.000

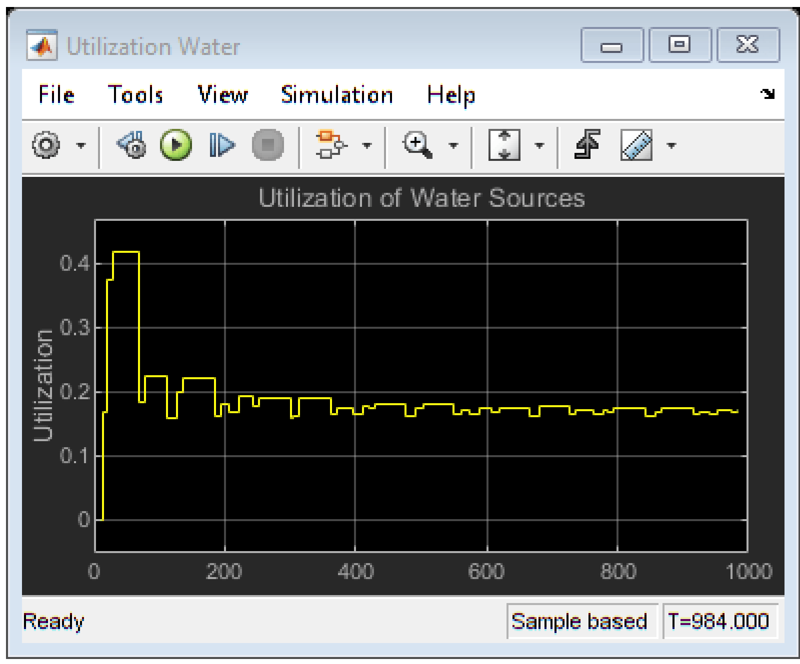

Waiting For Heat
File Tools View Simulation Help
Avg Num of Batches Waiting for Heat
Count
1
0.5
0
-0.5
-1
0 200 400 600 800 1000
Ready
Sample based
T=984.000



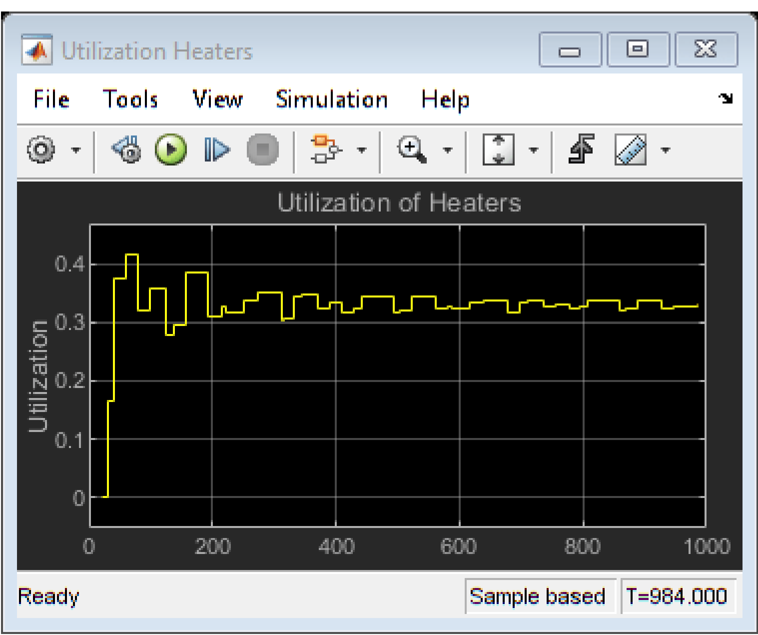
Waiting For Drain
File Tools View Simulation Help
Avg Num of Batches Waiting for Draining
Count
1
0.5
0
-0.5
-1
0
200
400
600
800
1000
Ready
Sample based
T=964.000

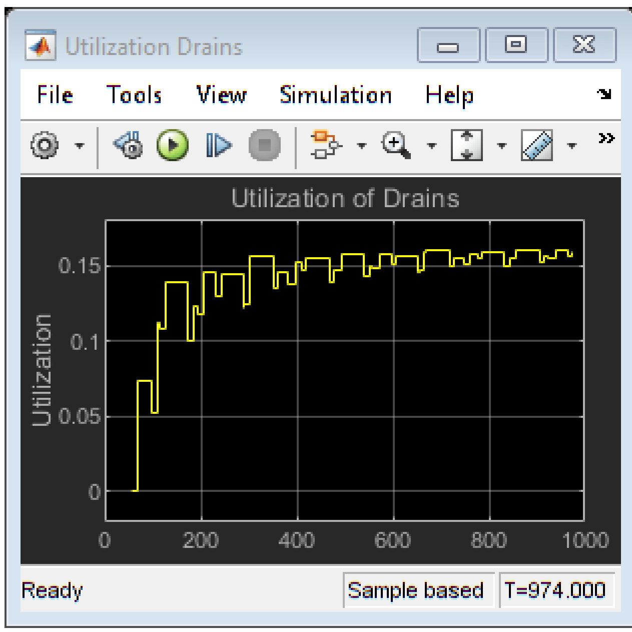
Utilization Reactors
File Tools View Simulation Help
Utilization of Batch Reactors
Utilization
1
0.8
0.6
0.4
0.2
0
0
200
400
600
800
1000
Ready
Sample based
T=974.000

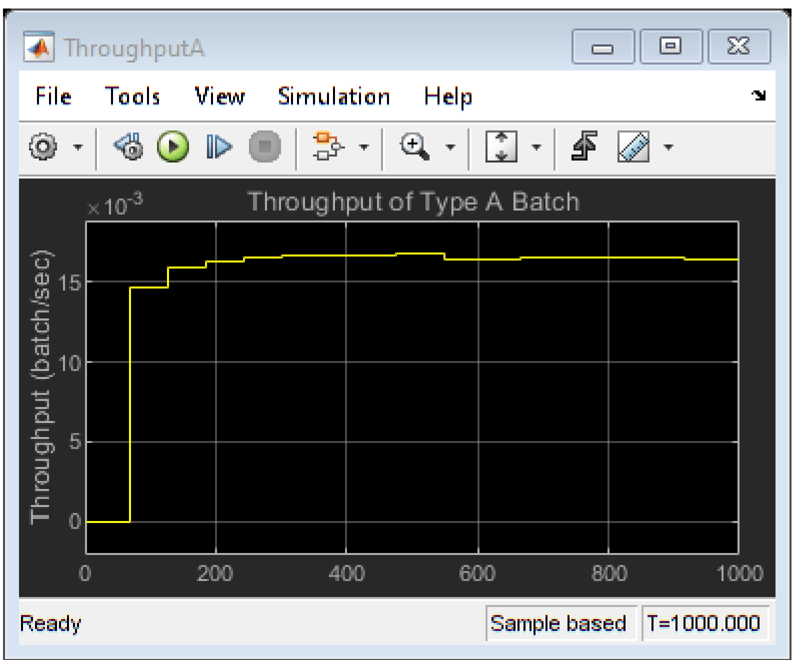


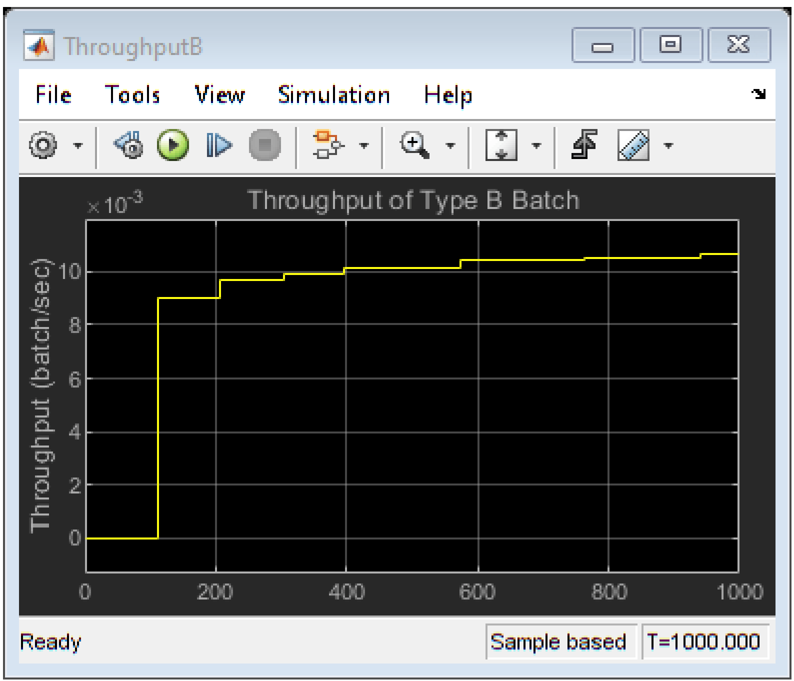
Utilization Water
File
Tools
View
Simulation
Help
Utilization of Water Sources
Utilization
0.4
0.3
0.2
0.1
0
0
200
400
600
800
1000
Ready
Sample based
T=984.000

Utilization Heaters
File
Tools
View
Simulation
Help
Utilization of Heaters
Utilization
0.4
0.3
0.2
0.1
0
0
200
400
600
800
1000
Ready
Sample based
T=984.000


Utilization Drains
File Tools View Simulation Help
Utilization of Drains
Utilization
0.15
0.1
0.05
0
0 200 400 600 800 1000
Ready
Sample based
T=974.000


ThroughputA
File Tools View Simulation Help
×10-3
Throughput of Type A Batch
Throughput (batch/sec)
15
10
5
0
0 200 400 600 800 1000
Ready
Sample based
T=1000.000




**Optimizing Resource Capacities**

We will now apply a Genetic Algorithm solver from the MATLAB Global Optimization Toolbox to this SimEvents model to find optimal resource capacities for this system. The genetic algorithm solves optimization problems by repeatedly modifying a population of individual points. Due to its random nature, the genetic algorithm improves your chances of finding a global solution. It does not require the functions to be differentiable or continuous.

The decision variables in this optimization are:

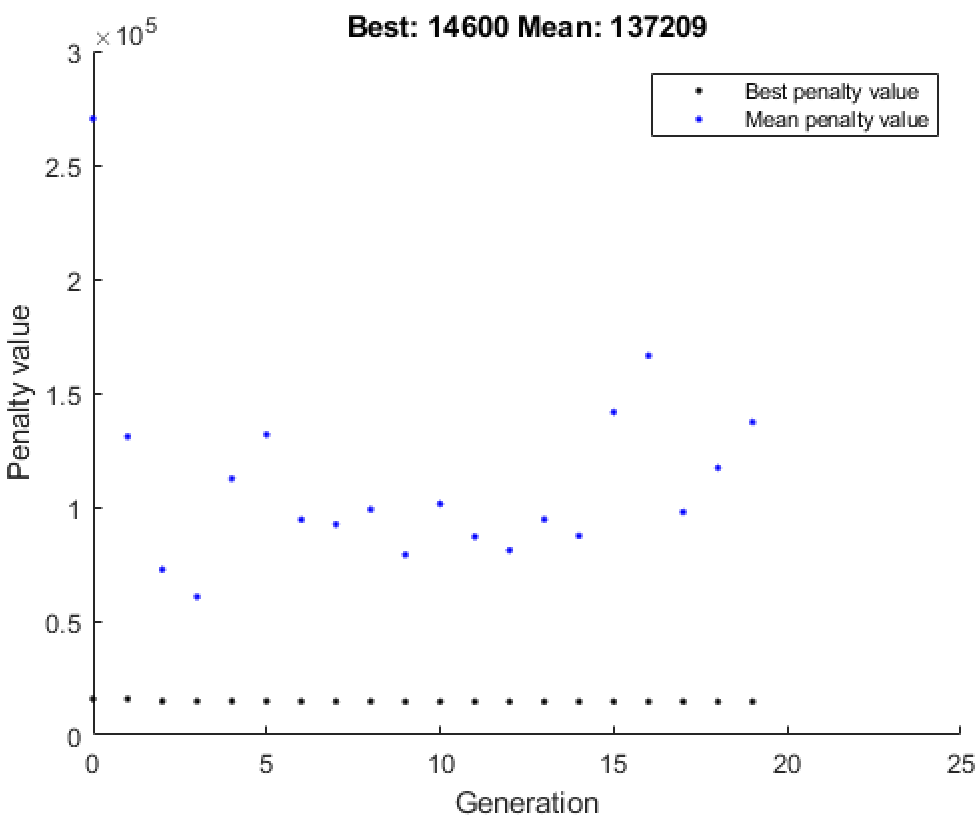
- Number of batch reactors
- Number of water tanks
- Number of heaters
- Number of drains

The genetic algorithm sets these variables as it runs multiple simulations of the model via the variable ResourceCapacity. The starting values of resource capacities are shown below:

```
cellfun(@(x)close_system(x), scopes);

disp('ResourceCapacity before optimization =');
disp(ResourceCapacity);
close_system([modelname '/Data Analysis/Order Backlog']);
ResourceCapacity = seRunOptimizationForBatchProductionProcess();
disp('ResourceCapacity after optimization =');
disp(ResourceCapacity);
```

```
ResourceCapacity before optimization =
     2     2     2     2

Starting parallel pool (parpool) using the 'local' profile ...
Connected to the parallel pool (number of workers: 6).
```

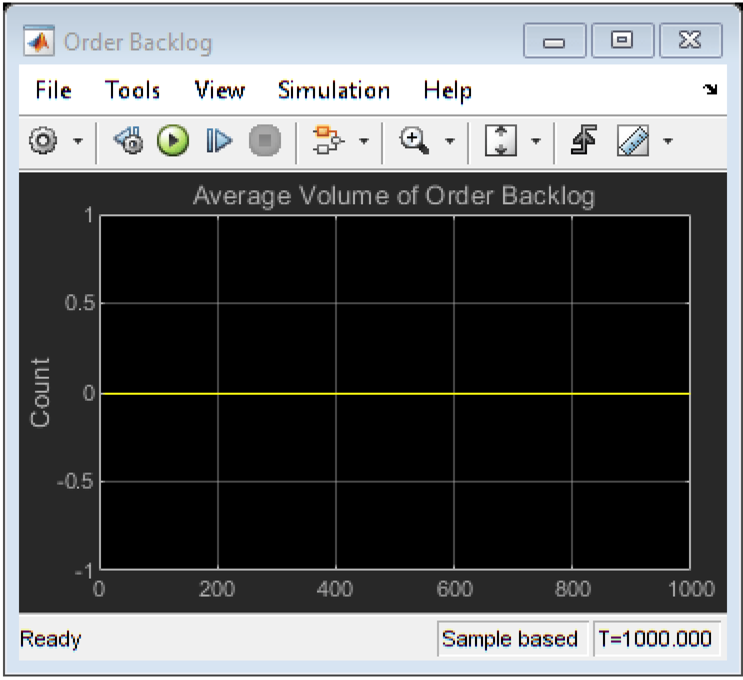
```
Optimization terminated: average change in the penalty fitness value less than options.FunctionTo
and constraint violation is less than options.ConstraintTolerance.
Elapsed time is 112.822855 seconds.
Parallel pool using the 'local' profile is shutting down.
ResourceCapacity after optimization =
    13     2     4     2
```




### Apply Optimization Results

We can now resimulate after applying the results of the optimization process back to the model to see that this significantly reduced the order backlog.

```
sim(modelname);
open_system([modelname '/Data Analysis/Order Backlog']);
```




```
%cleanup
bdclose(modelname);
clear model scopes
```

## See Also

Entity Server | Queue | Resource Pool | Resource Acquirer | Resource Releaser | Entity Generator

## Related Examples

- “Create a Discrete-Event Model”
- “Manage Entities Using Event Actions”
- “Entity Priorities” on page 1-36
- “Resource Allocation Modeling”

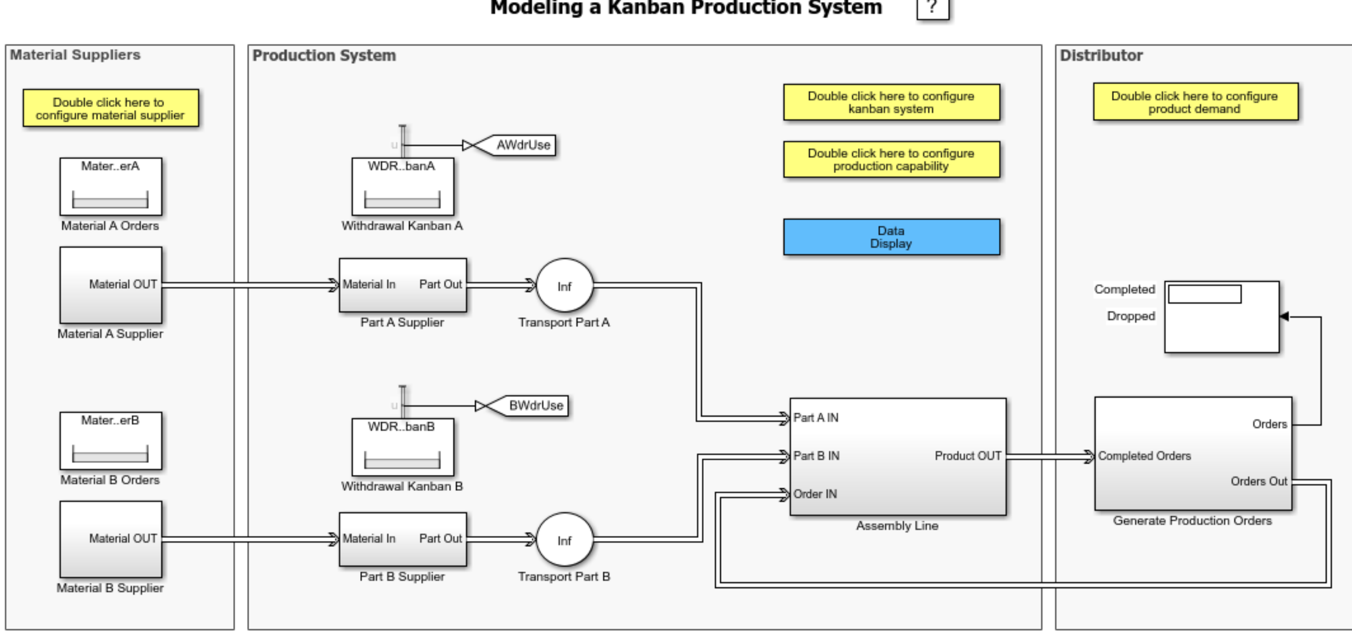
# Modeling a Kanban Production System

### Overview

This example shows a production system that uses kanbans to manage production activities. Analysis of simulation results highlights problems in the system and suggests ways to improve its performance.

### Structure of the Model

The modeled production system includes two part suppliers and an assembly line. The part suppliers use raw materials to manufacture parts. Finished parts are transported to the assembly line to fabricate final products. Completed products are shipped to distributors to fill production orders.




At the top level of the model:

- The Generate Production Orders subsystem simulates the generation of production orders.
- The Assembly Line subsystem fills a production order by assembling two types of parts (referred to as part A and part B) into final products.
- The Part A Supplier subsystem and Part B Supplier subsystem manufacture the parts needed for final assembly.
- The Material A Supplier subsystem and Material B Supplier subsystem replenish the raw materials consumed during parts production.

### Kanban Circulation

"Kanban" comes from the Japanese word for "signboard". A kanban production system is a pull system that determines its production according to the actual demand of the customers. These

systems use kanbans as demand signals that propagate through the production system to trigger and regulate production activities, such as processing and storage.

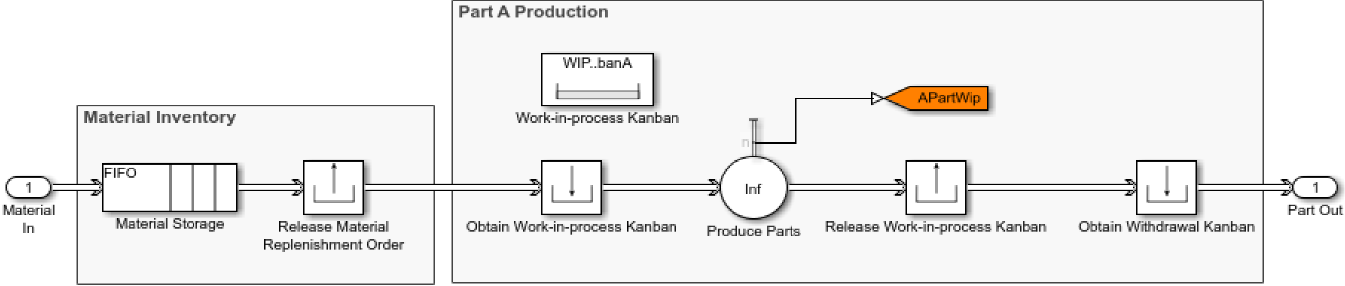
This model simulates the circulation of two types of kanbans: withdrawal kanbans and work-in-process kanbans.

- Withdrawal kanbans manage inventory. Withdrawal kanbans grant the right to withdraw parts from part suppliers to replenish inventory. Factory workers cannot remove the withdrawal kanban from a part in the existing inventory until the part is consumed. During production, the number of withdrawal kanbans issued for a type of parts is fixed. This limits the inventory size for that type of part.
- Work-in-process kanbans manage production. Work-in-process kanbans grant the right to manufacture parts in type and quantity as specified. After a part is produced, factory workers cannot remove the work-in-process kanban from the part until the part is withdrawn for final assembly. During production, the number of work-in-process kanbans issued for a type of parts is fixed. This limits the number of parts being processed by a part supplier.

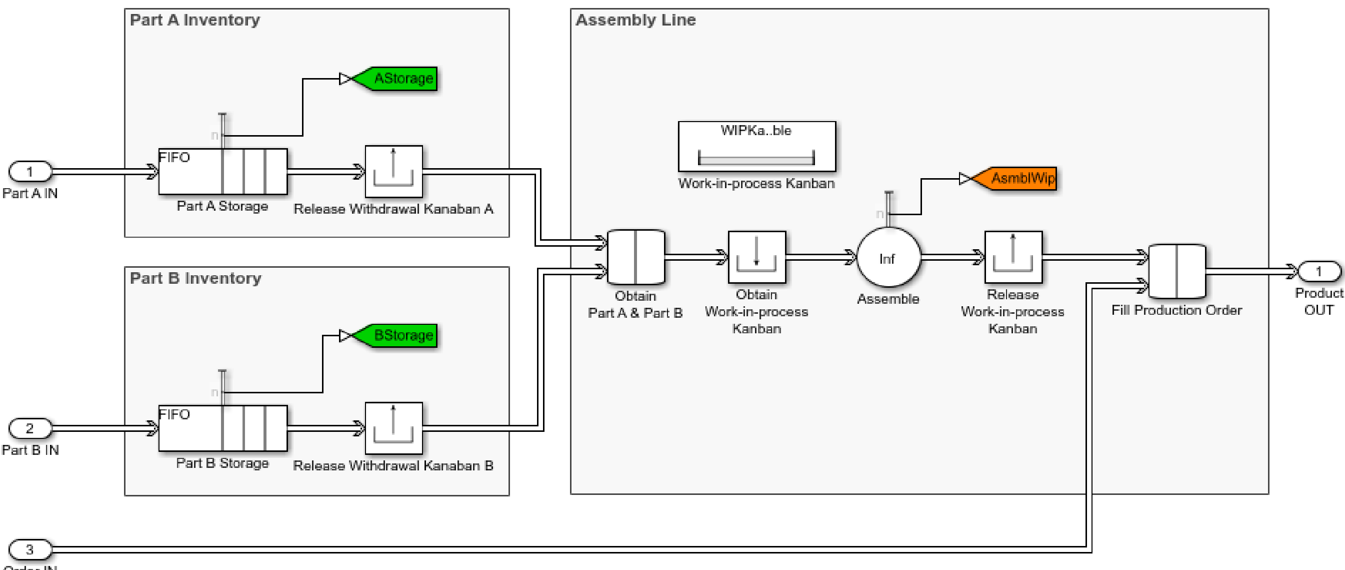
Circulation of withdrawal kanbans for part A is modeled by the following blocks and subsystems:

- Resource Acquirer block labeled Obtain Withdrawal Kanban in Part A Supplier subsystem
- Resource Releaser block labeled Release Withdrawal Kanban A in Assembly Line subsystem
- Resource Pool block labeled Withdrawal Kanban A

The figures below show the Part A Supplier subsystem and Assembly Line subsystem.




Part A Supplier




During simulation, the block labeled Obtain Withdrawal Kanban in the Part A Supplier subsystem must obtain a withdrawal kanban before a part A is transported and stored for final assembly. When a part A in storage is consumed in final assembly, the block labeled Release Withdrawal Kanban A in the Assembly Line subsystem releases the withdrawal kanban. The kanban then returns to the block labeled Obtain Withdrawal Kanban to allow replenishment of part A inventory.

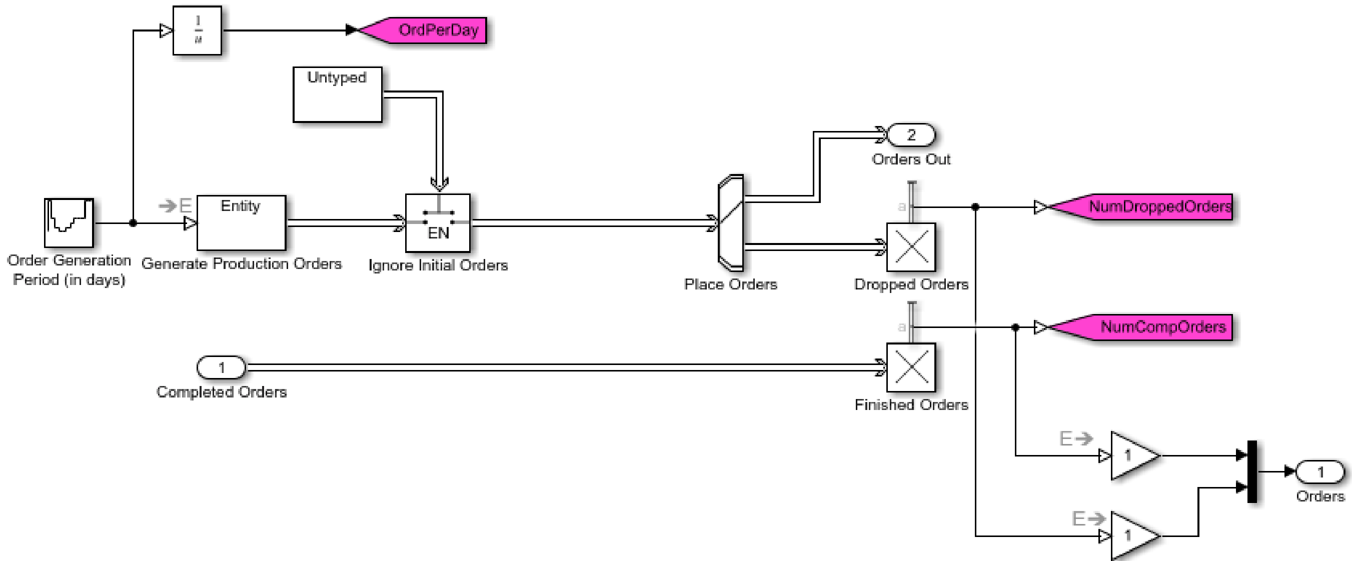
Circulation of work-in-process kanbans is modeled in the same fashion as withdrawal kanbans. For example, in the Part A Supplier subsystem, the block labeled Obtain Work-in-process Kanban requests a work-in-process kanban upon producing a part A. After the part is completed and withdrawn, the block labeled Release Work-in-process Kanban releases the work-in-process kanban. The kanban then returns to the block labeled Obtain Work-in-process Kanban to allow the production of more of part A.

The model uses Resource Pools to model the group of kanbans. To learn about this technique, see "Resource Allocation from Multiple Pools" on page 6-55.

### Dropped Orders

A kanban production system reduces cost and waste by limiting inventories of work-in-process stock and finished products. However, when product demand fluctuates, lack of inventory may cause dropped orders.

This model simulates dropped orders caused by seasonal demand fluctuations. In the `Generate Production Orders` subsystem, the Output Switch block labeled `Place Orders` uses `First port that is not blocked` as its switching criterion. During simulation, the block tries to send an order to the `Assembly Line` subsystem. If the inventory of finished product is empty, the block labeled `Fill Production Order` in the `Assembly Line` subsystem does not accept this order. The block labeled `Place Orders` then drops this order by forwarding it to the Entity Sink block labeled `Dropped Orders`.




### Results and Displays

During simulation, the Data Display subsystem displays these scopes to show the performance of the production system:

- Part A Withdrawal Kanban Backlog
- Part B Withdrawal Kanban Backlog
- Number of Part A in Process
- Number of Part B in Process
- Number of Products in Final Assembly
- Number of Part A in Storage
- Number of Part B in Storage
- Product Demand
- Number of Dropped Orders
- Number of Completed Orders

A Display block at the root level of the model provides a numeric view of the number of orders completed and the number of orders dropped.

### Experimenting with the Model

(*For use with live model only*)

- Open the configuration dialog for product demand by double-clicking a configuration block in the colored region labeled Distributor. Change product demand by changing the **Daily product demand in each month of the year** parameter in this dialog.
- Open the configuration dialog for the kanban system by double-clicking a configuration block in the colored region labeled Production System. Change the number of withdrawal kanbans and work-in-process kanbans issued for the production system by changing the parameters in this dialog.

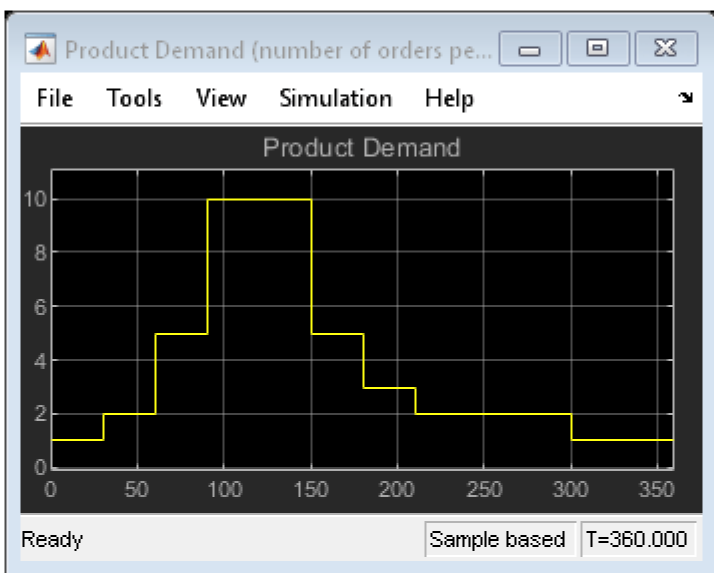
- Open the configuration dialog for production capability by double-clicking a configuration block in the colored region labeled Production System. Change the time it takes to manufacture, transport, and assemble parts or final products by changing the parameters in this dialog.
- Open the configuration dialog for the material suppliers by double-clicking a configuration block in the colored region labeled Material Supplier. Change the time it takes to produce and deliver raw materials by changing the parameters in this dialog.

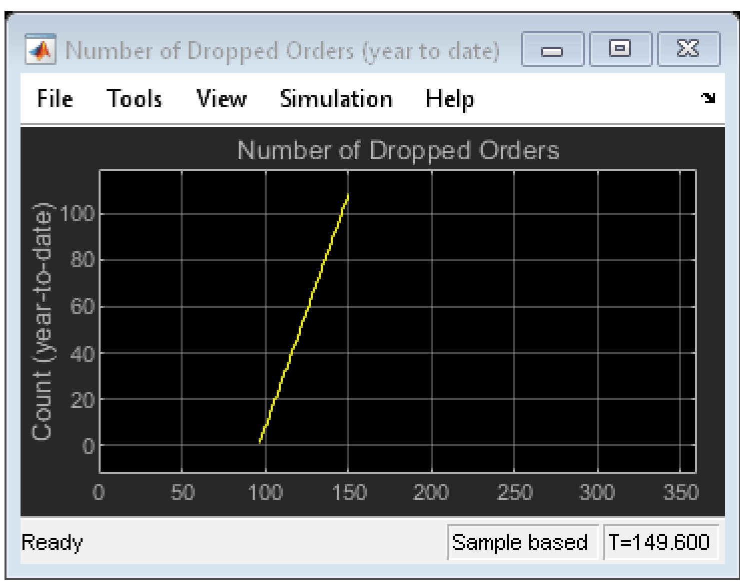
### Using the Model for Performance Analysis

The model with the original configuration represents a kanban production system with significant lost sales in months when demand is at a peak. Analysis of simulation results suggests solutions to address this issue.

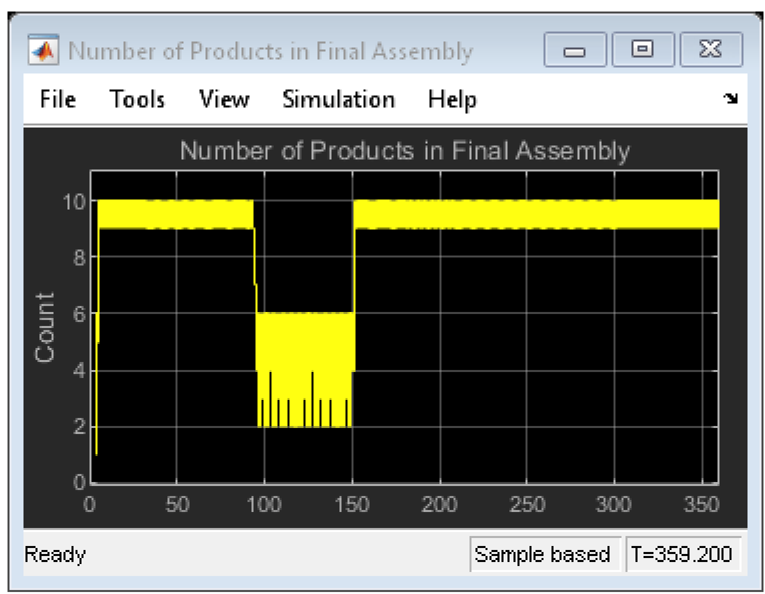
The following steps show how the solutions are developed.

**Step 1:** Run the simulation using the original configuration. As shown in the figures below, the scope labeled Number of Dropped Orders indicates that the production system suffers significant lost sales between day 90 and day 150 of the year. Comparing this result with the scope labeled Product Demand indicates that lost sales happen when product demand is at a peak.

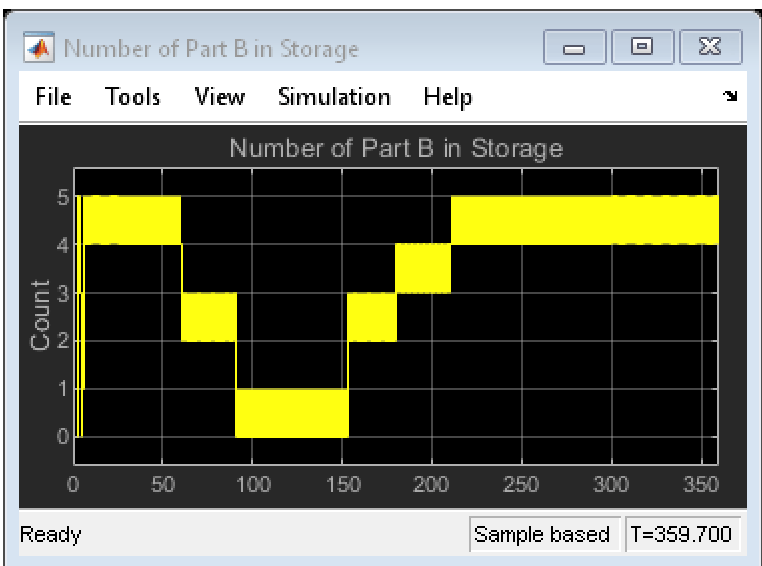




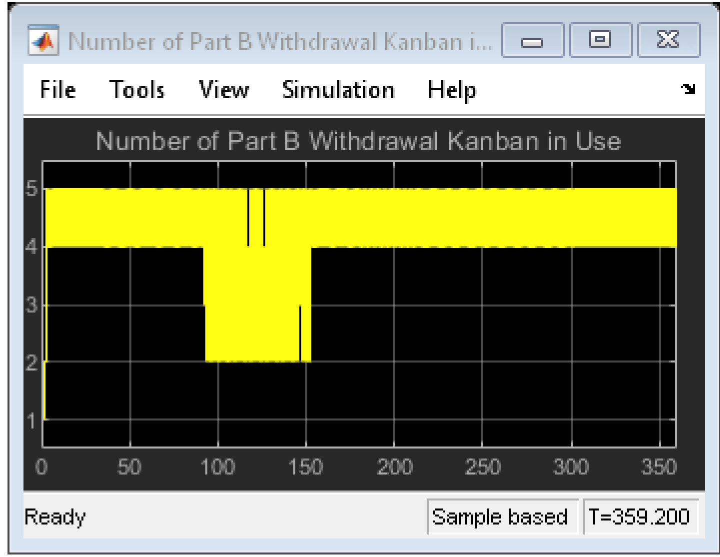

**Step 2:** Comparing the demand in peak season with product supply indicates the assembly line does not provide sufficient production capability. According to the scope labeled Product Demand (see the figure above), 10 products are needed daily between day 90 and day 150. In contrast, as illustrated by the scope labeled Number of Products in Final Assembly (see the figure below), in the same period of time, only about 5 are in production every day - much less than the quantity in demand.




**Step 3:** Further observation of simulation results indicates that the inventory of part B is insufficient in the peak season. As illustrated by the scope labeled Number of Part B in Storage (see the figure below), inventory is empty in the peak season. This explains the inadequacy in the production capability during final assembly - the assembly line is not provided with sufficient part B.




**Step 4:** Simulation results related to part B indicate that the use of withdrawal kanbans for part B is low in the peak season. This is displayed by the scope labeled Number of Part B Withdrawal Kanban in Use shown in the figure below.




Use of withdrawal kanbans is reduced when the assembly line requests a replenishment but the part supplier fails to respond to this request in time. This leads to an analysis of the production capability of part B in the peak season of the year.

**Step 5:** The visual observations in the earlier steps suggest this quantitative analysis:

- According to the scope labeled `Product Demand`, ten final products are required daily in the peak season.
- Since 1 final product is assembled from one Part B and one Part A, to fully satisfy demand, ten Part Bs, are needed for final assembly per day. That is:

```
Part B demand = 10 /day
```

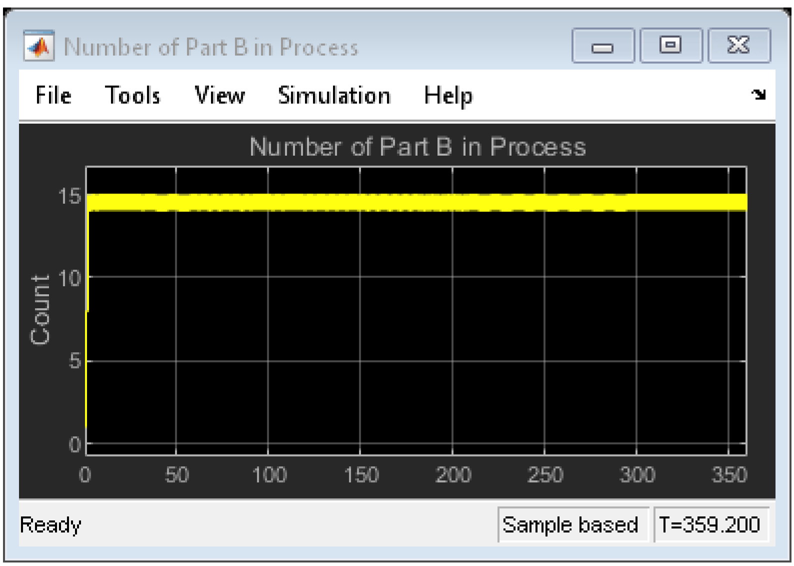
- According to production capability configurations, it takes the part supplier 1.5 days to produce one part B. According to kanban system configurations, 12 work-in-process kanbans are issued for part B. This limits the maximal number of parts produced in parallel to 12. Thus, the maximal production rate of part B is:

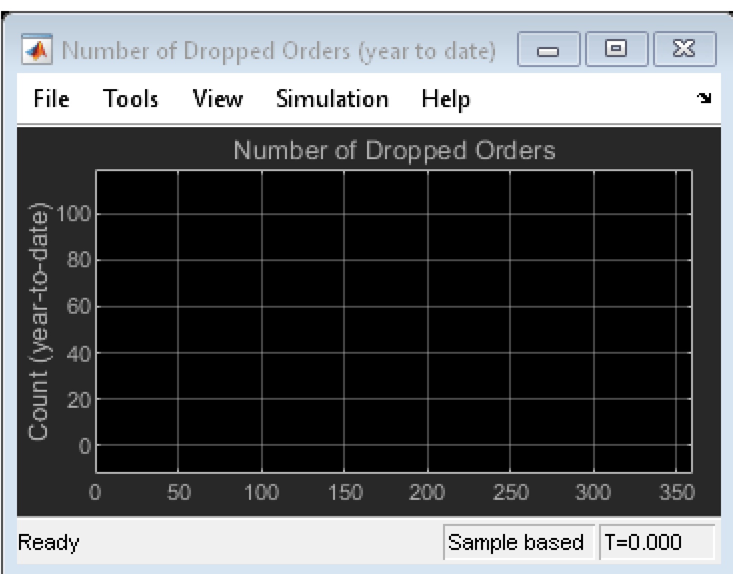
```
Maximal part B production rate = 12/1.5 = 8 /day
```

**Step 6:** Comparing demand and maximal production rate of part B indicates the inadequacy in production capacity. Two possible solutions are:

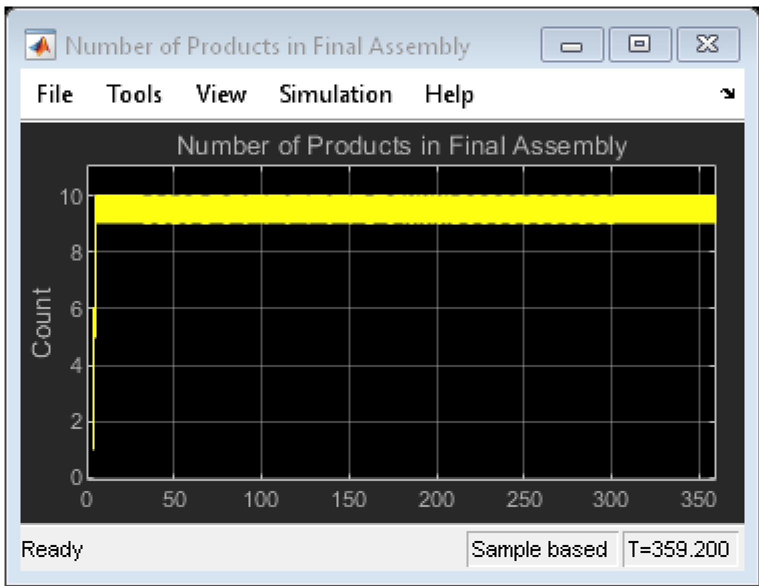
- Issue more work-in-process kanbans for part B to allow more parts to be produced in parallel. To increase the maximal production rate of part B to above 10, issue at least 3 more work-in-process kanbans.
- Reduce production cycle of part B to increase production rate. Production cycle needs to shorten by at least 0.3 day to meet required production rate.

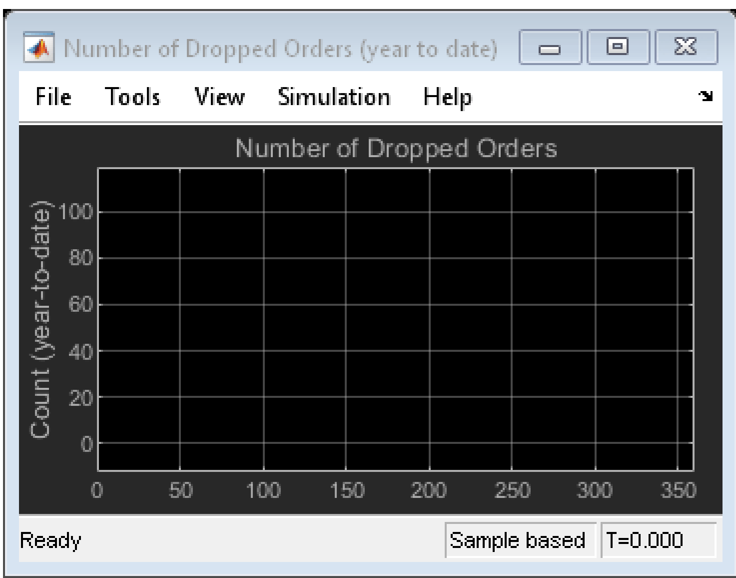
**Step 7:** To verify solution 1, reconfigure the kanban system by increasing the **Number of work-in-process kanbans for part B** parameter to 15. Simulation results indicate that with such an update, fifteen part Bs are produced in parallel (see the scope labeled Number of Part B in Process below). As indicated by the scope labeled Number of Dropped Orders, the increase in part B supply eliminates the occurrence of dropped orders.






To verify solution 2, starting from the original configuration, reconfigure production capability by reducing the **Time it takes to produce a part B** parameter to `1.2` day. With the increase in production capability, 10 final products are in assembly daily (see the scope labeled Number of Products in Final Assembly below). As illustrated in the scope labeled Number of Dropped Orders below, such production capability can fully satisfy product demand, resulting in no loss of sales over the year.






The above steps explore the root cause of lost sales due to seasonal fluctuation in product demand. Quantitative analysis suggests two solutions to respond to such demand fluctuations. Simulation verifies that both solutions can indeed help the production system avoid seasonal lost sales.

## See Also

Entity Server | Queue | Resource Pool | Resource Acquirer | Resource Releaser | Entity Generator

## Related Examples

- “Create a Discrete-Event Model”
- “Manage Entities Using Event Actions”
- “Entity Priorities” on page 1-36
- “Resource Allocation Modeling”

# Job Scheduling and Resource Estimation for a Manufacturing Plant

### Overview

This example shows you how to model a manufacturing plant. The plant consists of an assembly line that processes jobs based on a pre-determined schedule. This example walks you through a workflow for:

- Analyzing the impact of job schedule on throughput
- Estimating the number of workers

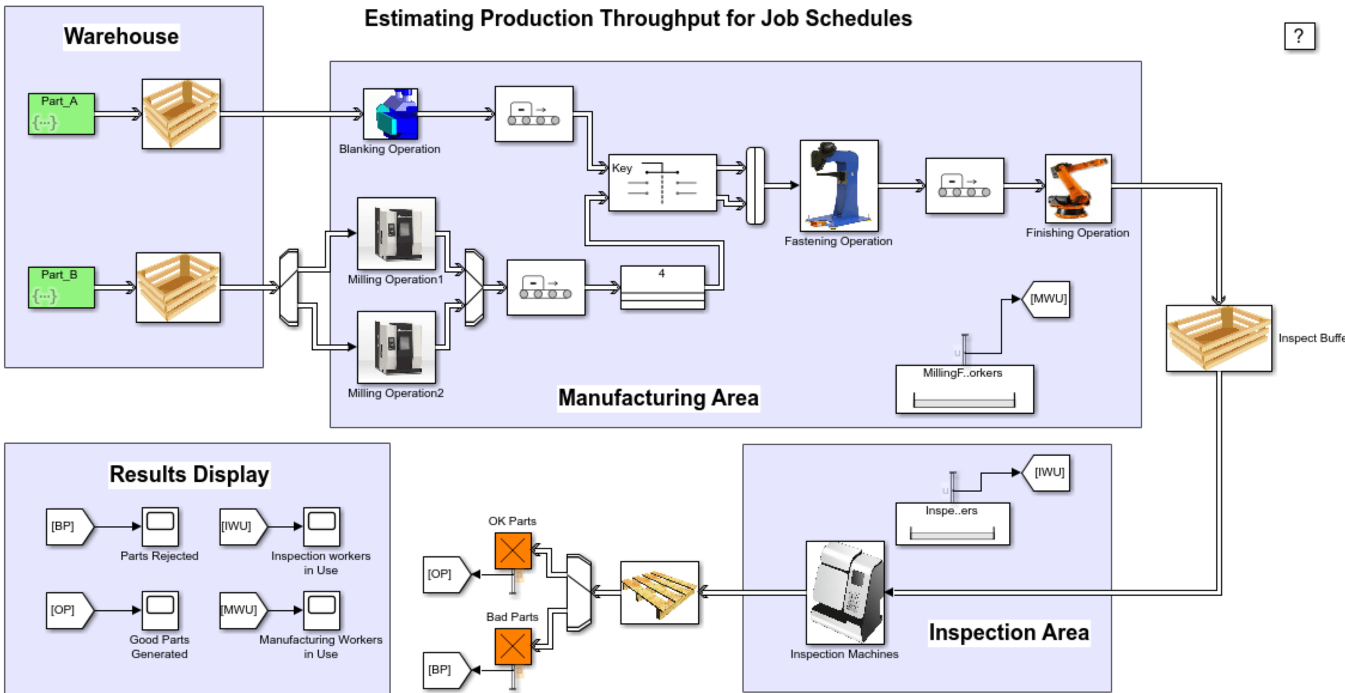
### Structure of the Model

The manufacturing plant caters to the production of 40 different product variants based on pre-defined schedules. Each variant requires two parts, PartA and PartB that correspond to that particular variant. Each part goes through a sequence of manufacturing steps. The following modeling details are specified in an Excel file that are read during model initialization:

- Schedule of part arrival in the plant
- Operation times for variants at each station along the assembly line
- Number of workers in different worker pools
- Rejection rate at the inspection area

The following script reads the excel file and initializes all the parameters.

```
% Initialization of variables used in the model
excelFile   = 'seEstimatingAssemblyLineThroughput.xlsx';

schedule    = xlsread(excelFile, 'MfgSchedule');
optimes     = xlsread(excelFile, 'OperationTimes');
parameters  = xlsread(excelFile, 'Parameters');

numMfgWorkers = parameters(1);         % number of workers in Manufacturing area
numInspectWorkers = parameters(2);     % number of workers in Inspection area
discard_rate = parameters(4)/100;     % quality rejection rate
seed = 12345;                         % random number seed
modelname = 'seEstimatingAssemblyLineThroughput';
open_system(modelname);
scopes = find_system(modelname,'LookUnderMasks','on','BlockType','Scope');
cellfun(@(x)close_system(x), scopes);
```




The manufacturing plant mainly consists of two areas:

- **The Manufacturing area**
- **The Inspection area**

**The Manufacturing area:** The plant receives *job orders* that are to be fulfilled. A *job order* specifies the variant ID and the required quantity for that particular variant. The Entity Generators generate parts based on a pre-defined sequence that satisfies the *job order*. In this example the sequence is either generated from a MATLAB script or is read from the excel sheet. The following script reads the *job order* requirements from the excel file.

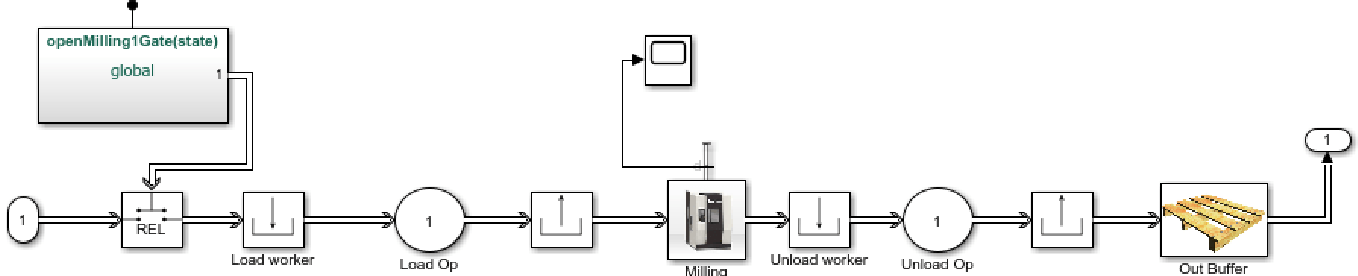
```
requirements = xlsread(excelFile, 'Requirements');
```

To manufacture a particular variant, PartA and PartB that correspond to the variant are brought in together into the manufacturing area. The parts go through the following steps before leaving the manufacturing area:

1. PartA goes through Blanking operation
2. PartB goes through Milling operation
3. Both the parts are then fastened
4. The assembly then goes through a Finishing operation

Average operation completion times for each variant are tabulated in the excel sheet. A 4% variation in operation completion times is assumed. Workers from the manufacturing worker pool load and unload parts from the Milling and Fastening machines.

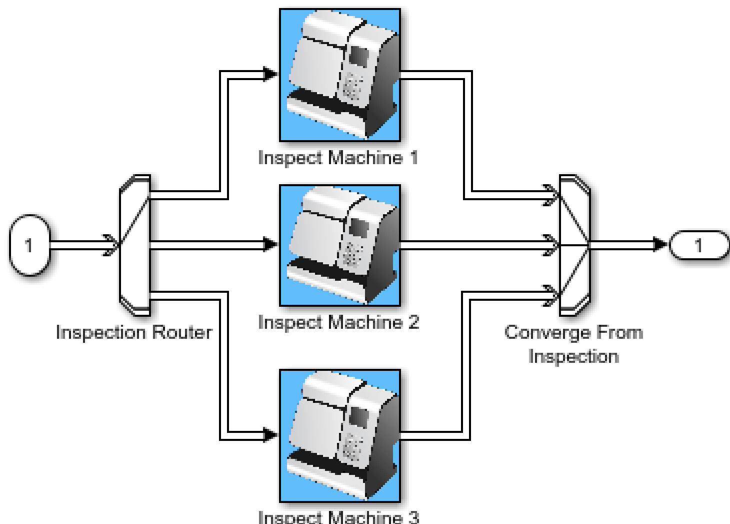
```
open_system([modelname '/Milling Operation1']);
```




```
close_system([modelname '/Milling Operation1']);
```

**The Inspection area:** The finished product enters the Inspection area, where the product is either certified to be ok or is rejected and scrapped. This example assumes a 5% rejection rate in the inspection area. Workers from the inspection worker pool load and unload parts from the three inspection machines.

```
open_system([modelname '/Inspection Machines']);
```




```
close_system([modelname '/Inspection Machines']);
```

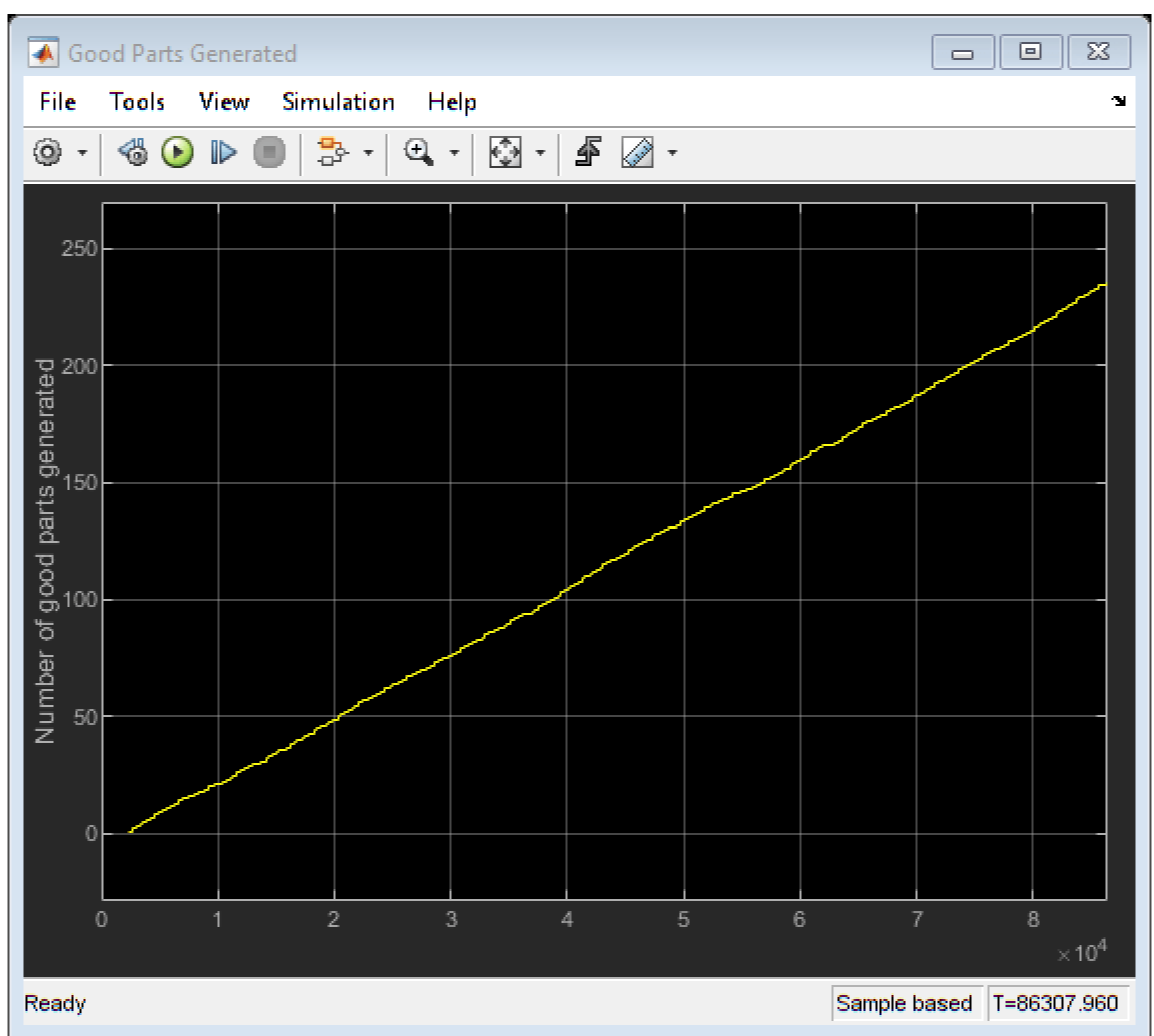
### Analyzing The Impact of Job Schedule on Throughput

To meet the *job order* requirements with the best throughput, different schedules can be generated. In this example, throughput is the total number of good products produced by the plant. The sheet named 'MfgSchedule' shows a few schedules that satisfy the *job order*. Following scripts generate job schedules based on certain criteria:

- **Schedule 1: Shortest job first on the Blanking machine:**

This schedule puts the operation having shortest running time on the Blanking machine first and the longest one at the end. The idea here is to push as many parts into the plant as early as possible. The throughput is then examined:

```
idx = 1;
S1 = sortrows(optimes(:, [1 2]), 2);
for i = 1:length(S1)
    repeat = requirements(S1(i), 2);
    for j = 1:repeat
        newSchedule(idx) = S1(i);
        idx = idx + 1;
    end
end
scheduleID = size(schedule, 2) + 1;
schedule(:, scheduleID) = newSchedule';
sim(modelname);
open_system([modelname '/Good Parts Generated']);
```

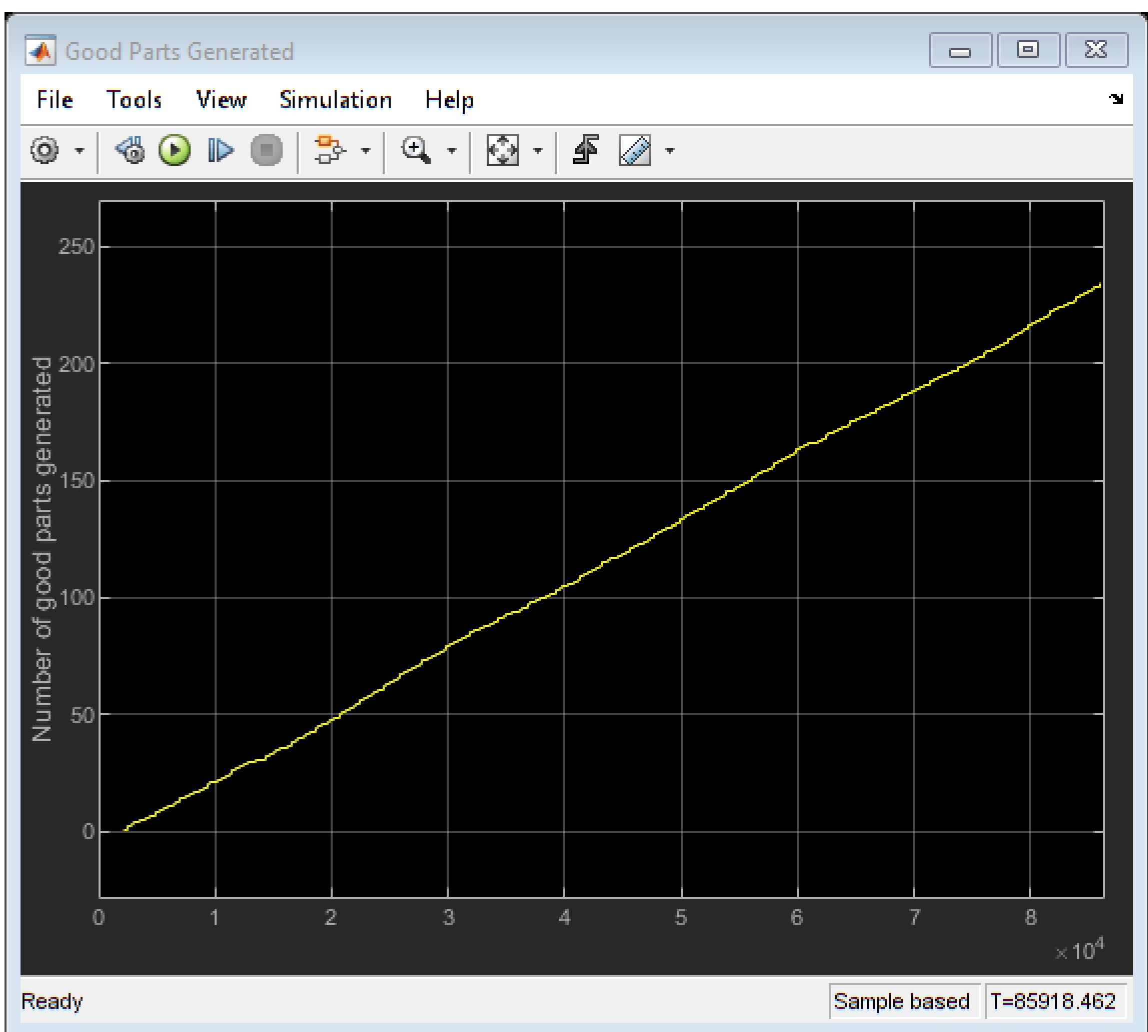



```
close_system([modelname '/Good Parts Generated']);
```

- **Schedule 2: Shortest job first on the Milling machines:**

This schedule puts the operation having the shortest running time on the Milling machines first and the longest one at the end. The idea again is to push as many parts into the plant as early as possible from the other starting branch of the plant. The throughput is then examined:

```
idx = 1;
S2 = sortrows(optimes(:, [1 3]), 2);
for i = 1:length(S2)
    repeat = requirements(S2(i), 2);
    for j = 1:repeat
        newSchedule(idx) = S2(i);
        idx = idx + 1;
    end
end
scheduleID = size(schedule, 2) + 1;
schedule(:, scheduleID) = newSchedule';
sim(modelname);
open_system([modelname '/Good Parts Generated']);
```

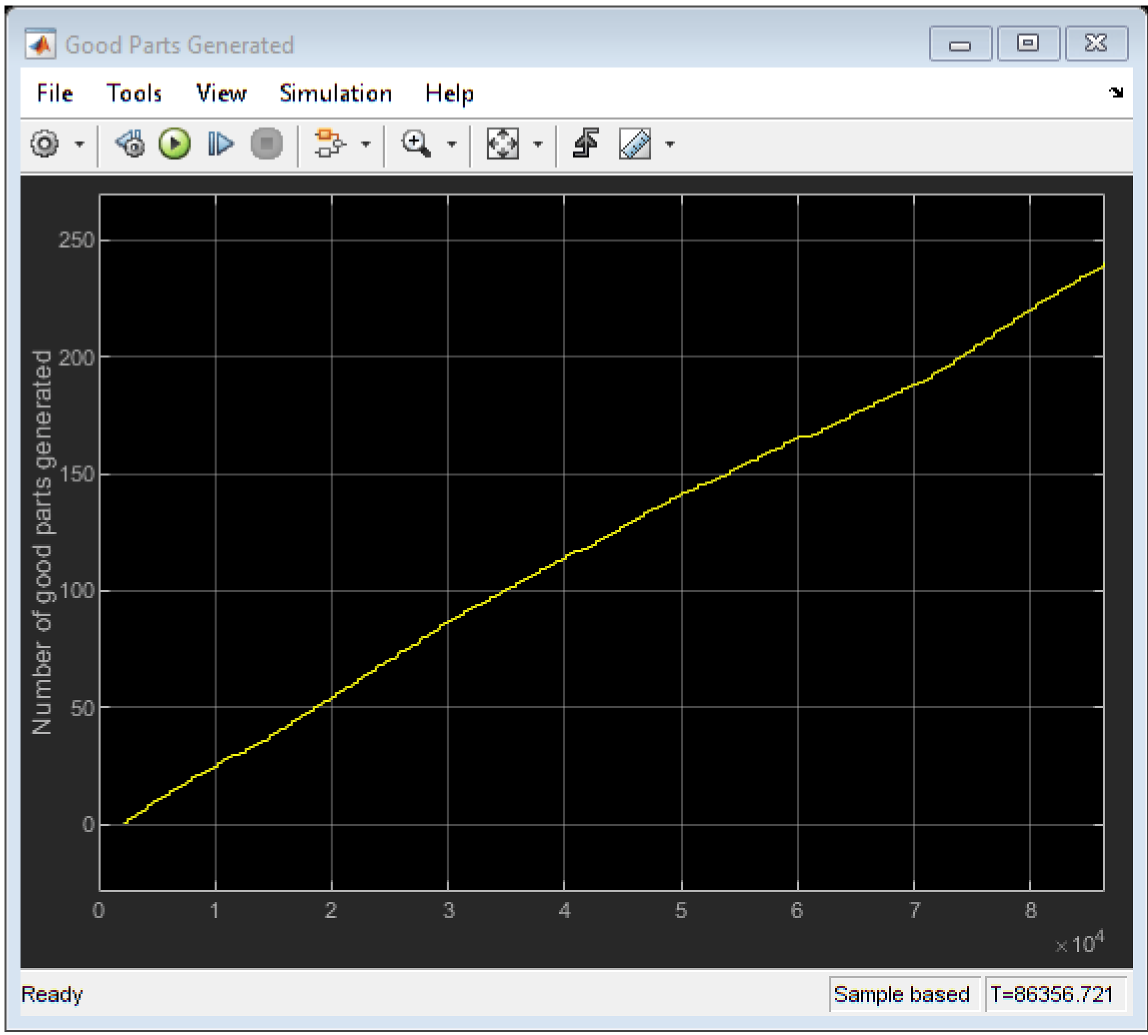



```
close_system([modelname '/Good Parts Generated']);
```

- **Schedule 3: Shortest job first on the Fastening machine:**

This schedule puts the operation having shortest running time on the Fastening machine first and the longest one at the end. The idea here is to push parts out of the bottleneck machine as early as possible. The throughput is then examined:

```
idx = 1;
S4 = sortrows(optimes(:, [1 5]), 2);
for i = 1:length(S4)
    repeat = requirements(S4(i), 2);
    for j = 1:repeat
        newSchedule(idx) = S4(i);
        idx = idx + 1;
    end
end
scheduleID = size(schedule, 2) + 1;
schedule(:, scheduleID) = newSchedule';
sim(modelname);
open_system([modelname '/Good Parts Generated']);
```

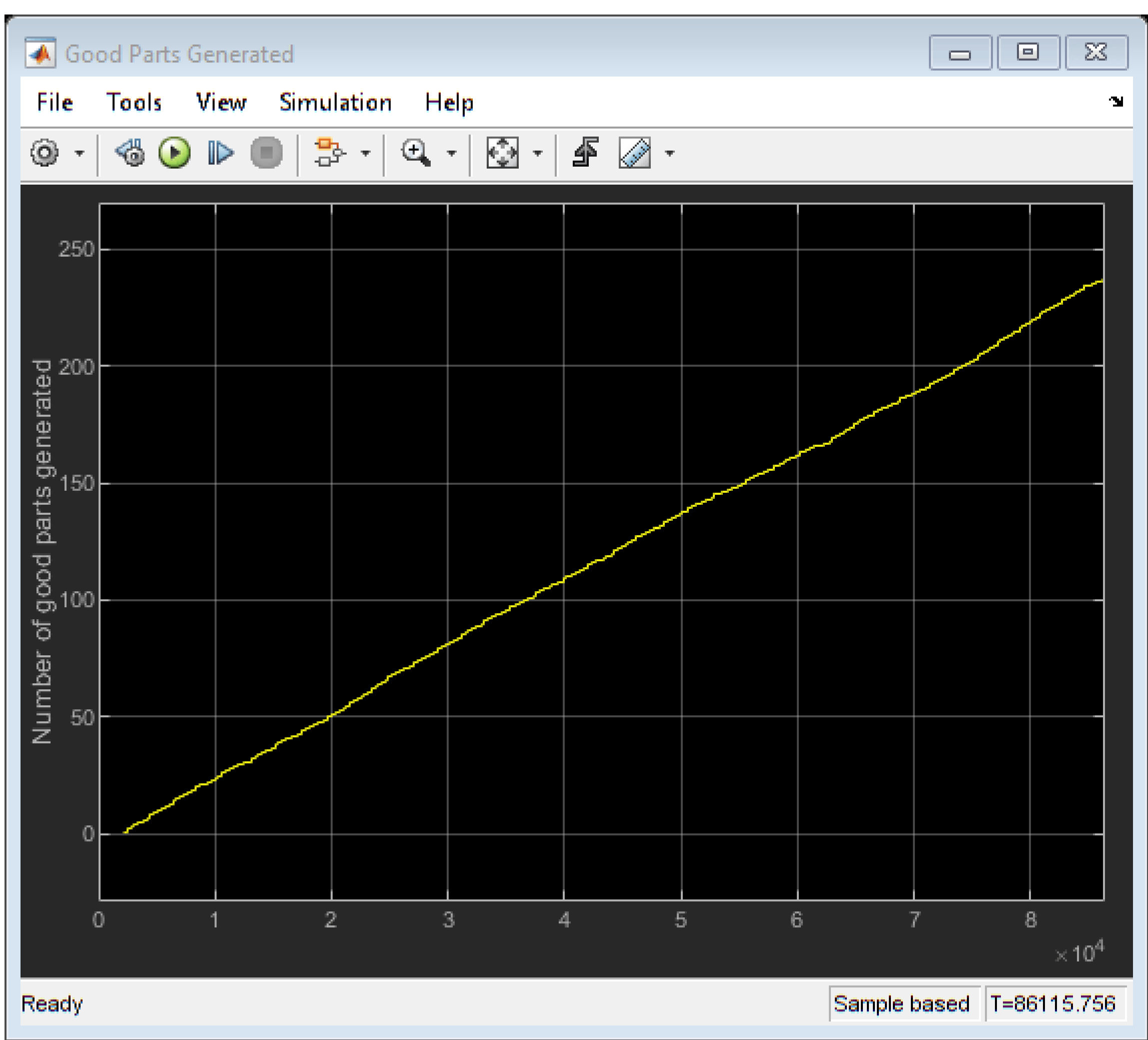



```
close_system([modelname '/Good Parts Generated']);%%
```

- **Schedule 4: Shortest job first using the cumulative manufacturing time:**

This schedule takes into account the cumulative run time on all the machines. The operation having the shortest cumulative run time is put first and the longest one goes to the end. The throughput is then examined:

```
idx = 1;
cumulativeSum = sortrows([optimes(:, 1) sum(optimes(:, [2 3 5 6]), 2)], 2);
for i=1:length(cumulativeSum)
    repeat = requirements(cumulativeSum(i), 2);
    for j = 1:repeat
        newSchedule(idx) = cumulativeSum(i);
        idx = idx + 1;
    end
end
scheduleID = size(schedule, 2) + 1;
schedule(:, scheduleID) = newSchedule';

sim(modelname);
open_system([modelname '/Good Parts Generated']);
```

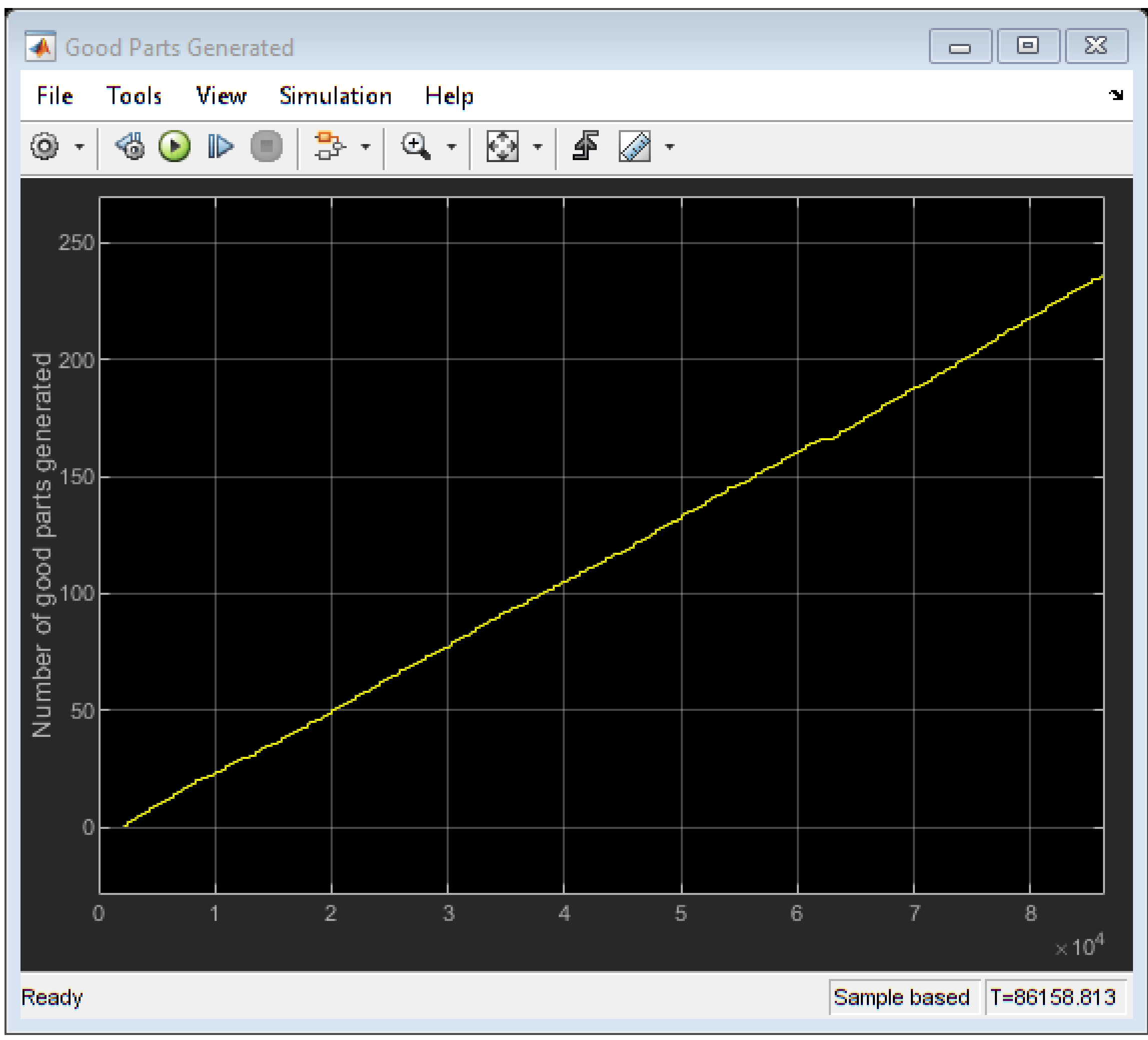



```
close_system([modelname '/Good Parts Generated']);
```

- **Schedules 5 to 8: Random schedules:**

Schedules 5 to 8 in the excel sheet are all random schedules which satisfy the *job order*. These schedules can be generated by starting from any schedule and generating a random permutation using the RANDPERM function. Following are the results for 'Schedule 8':

```
scheduleID = 9;
sim(modelname);
open_system([modelname '/Good Parts Generated']);
```




```
close_system([modelname '/Good Parts Generated']);
```

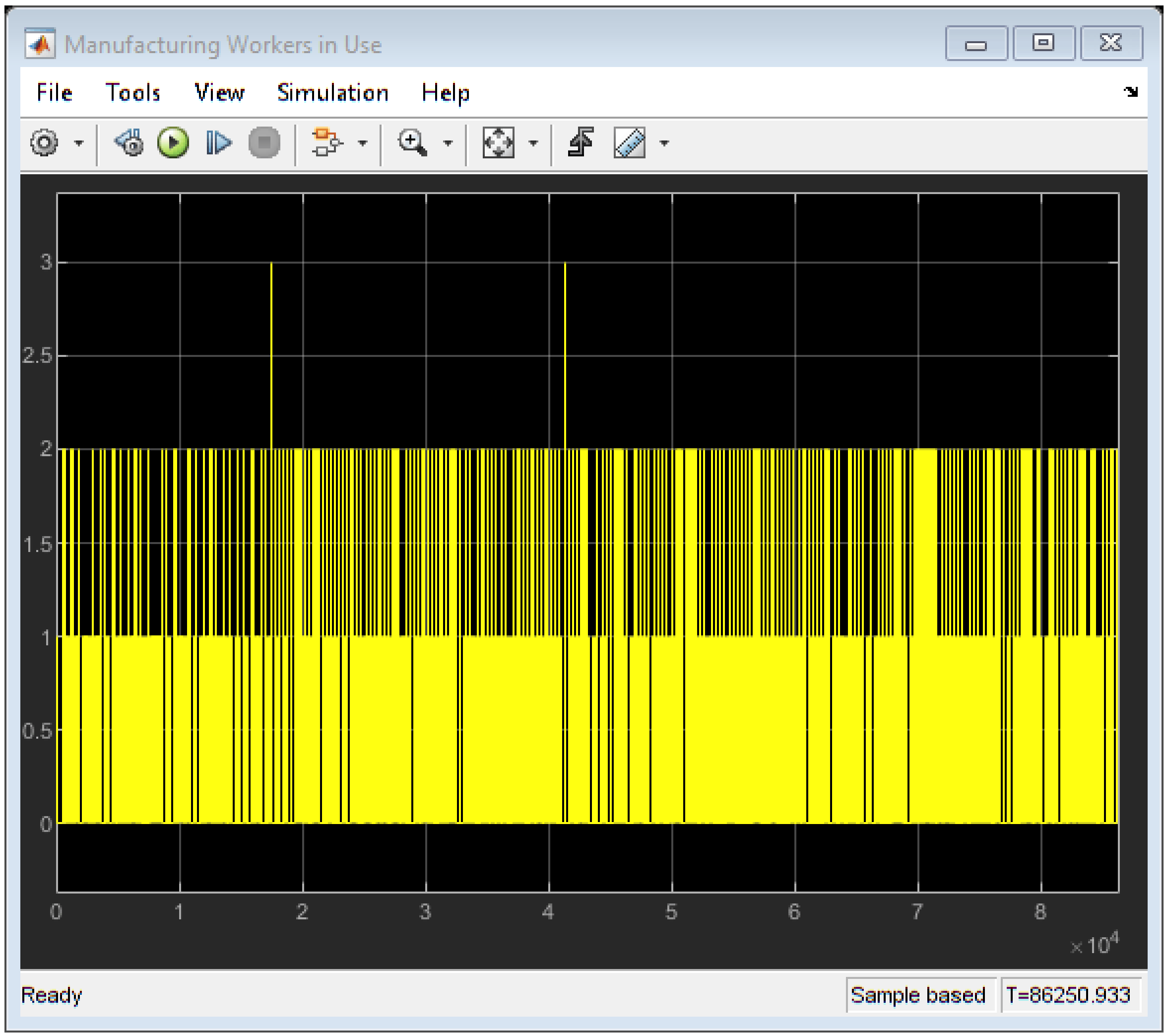
**Simulating all of the above strategies suggests that the schedule associated with 'Shortest job first on the Fastening Machine', 'Schedule 3' gives us the best throughput.**

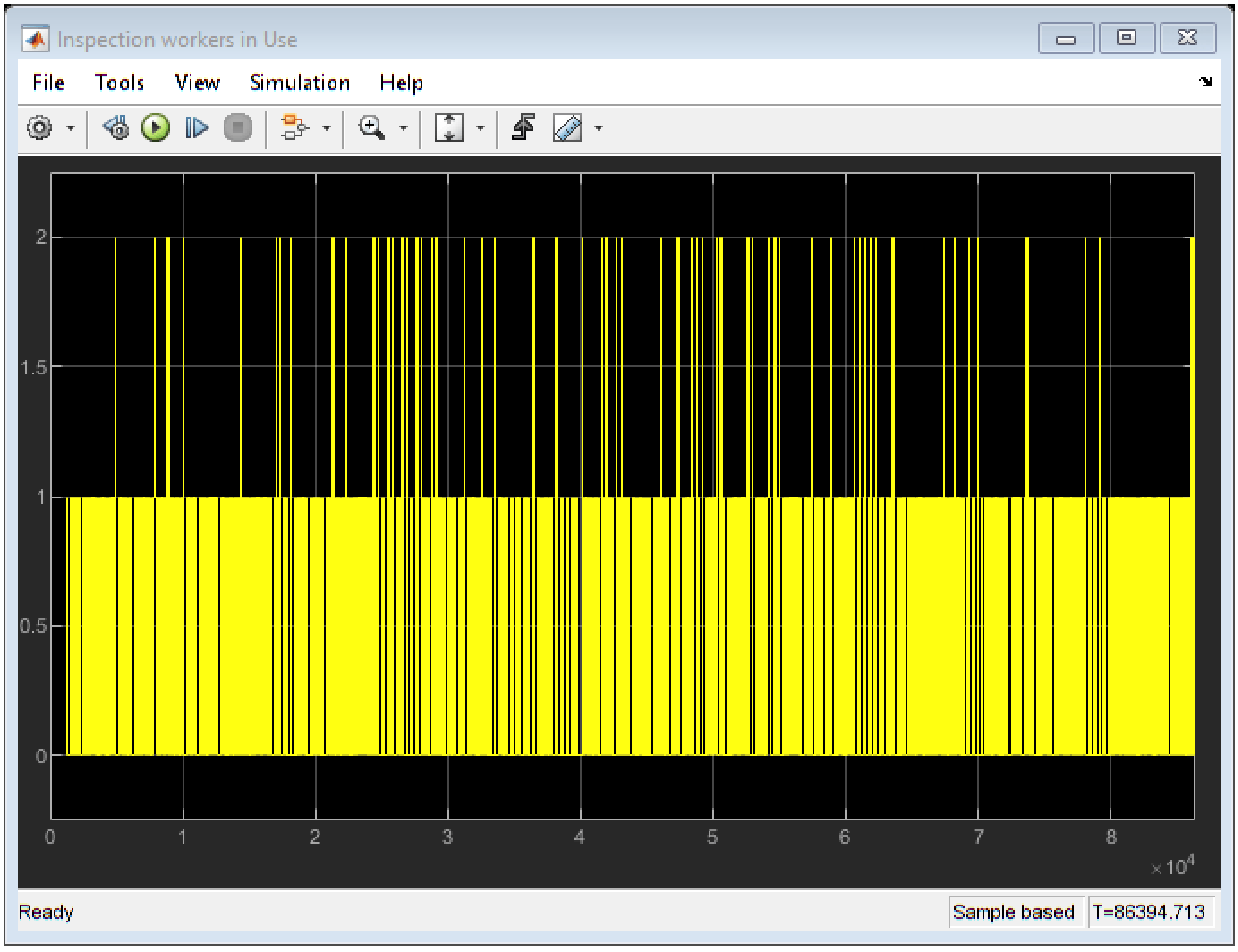
### Estimating the Number of Workers

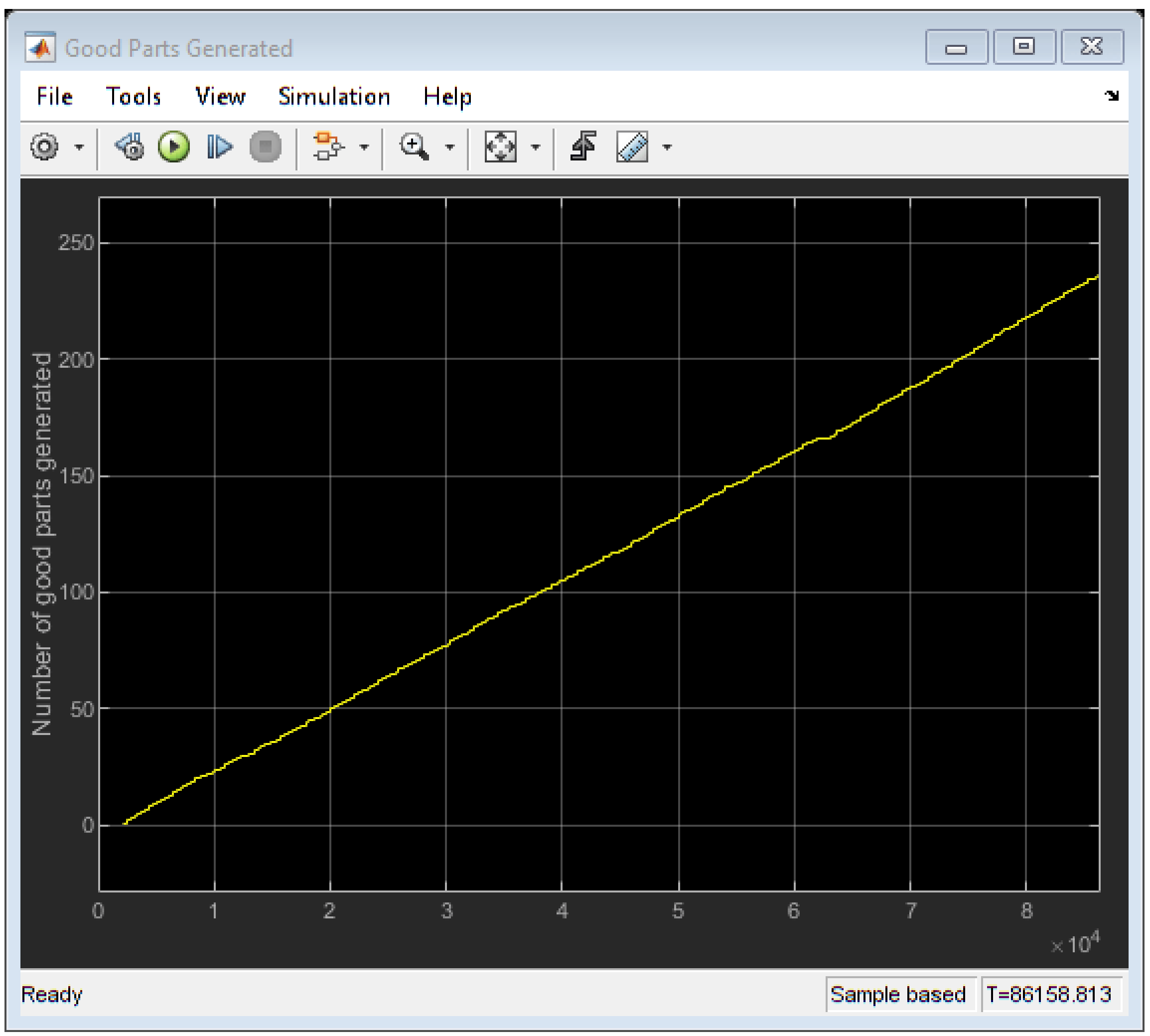
After selecting the best schedule, an estimate of the number of workers needed in the two worker pools is made. We start with three workers working in the Manufacturing area and three in the Inspection area.

```
numMfgWorkers = 3;
numInspectWorkers = 3;
```

```
sim(modelname);
open_system([modelname '/Manufacturing Workers in Use']);
open_system([modelname '/Inspection workers in Use']);
open_system([modelname '/Good Parts Generated']);
```



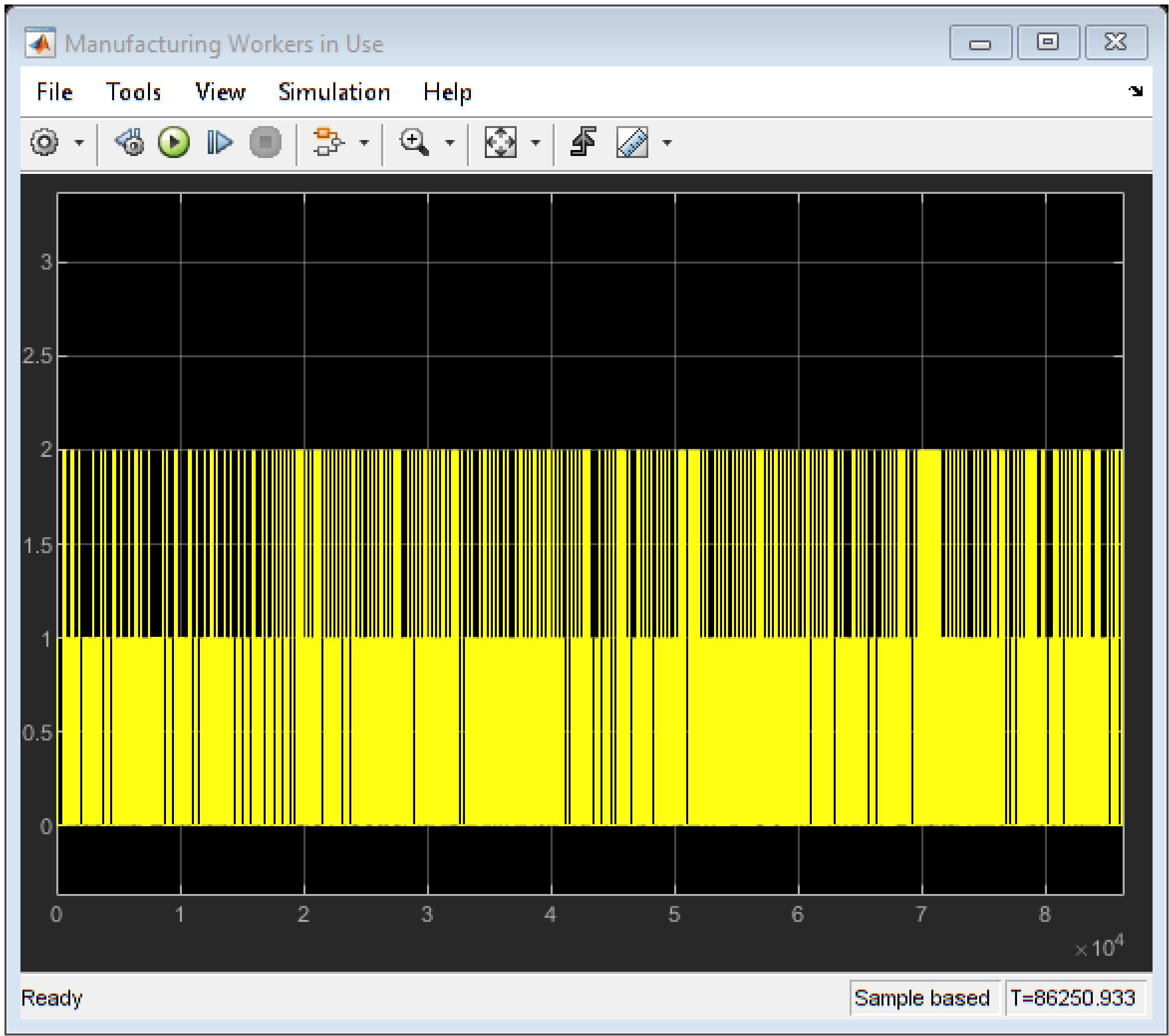

Inspection workers in Use
File Tools View Simulation Help
2
1.5
1
0.5
0
0 1 2 3 4 5 6 7 8
× 10^4
Ready
Sample based
T=86394.713

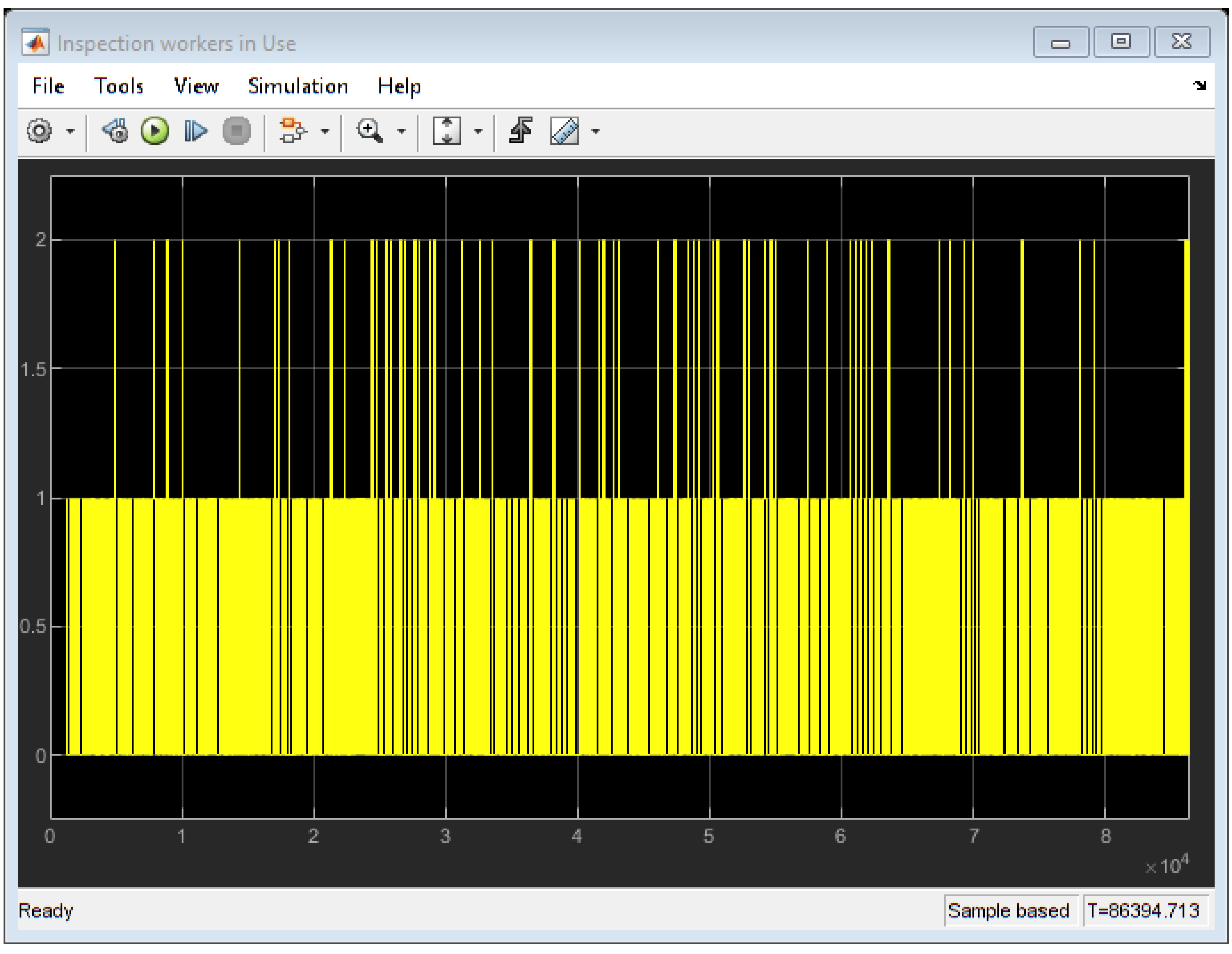


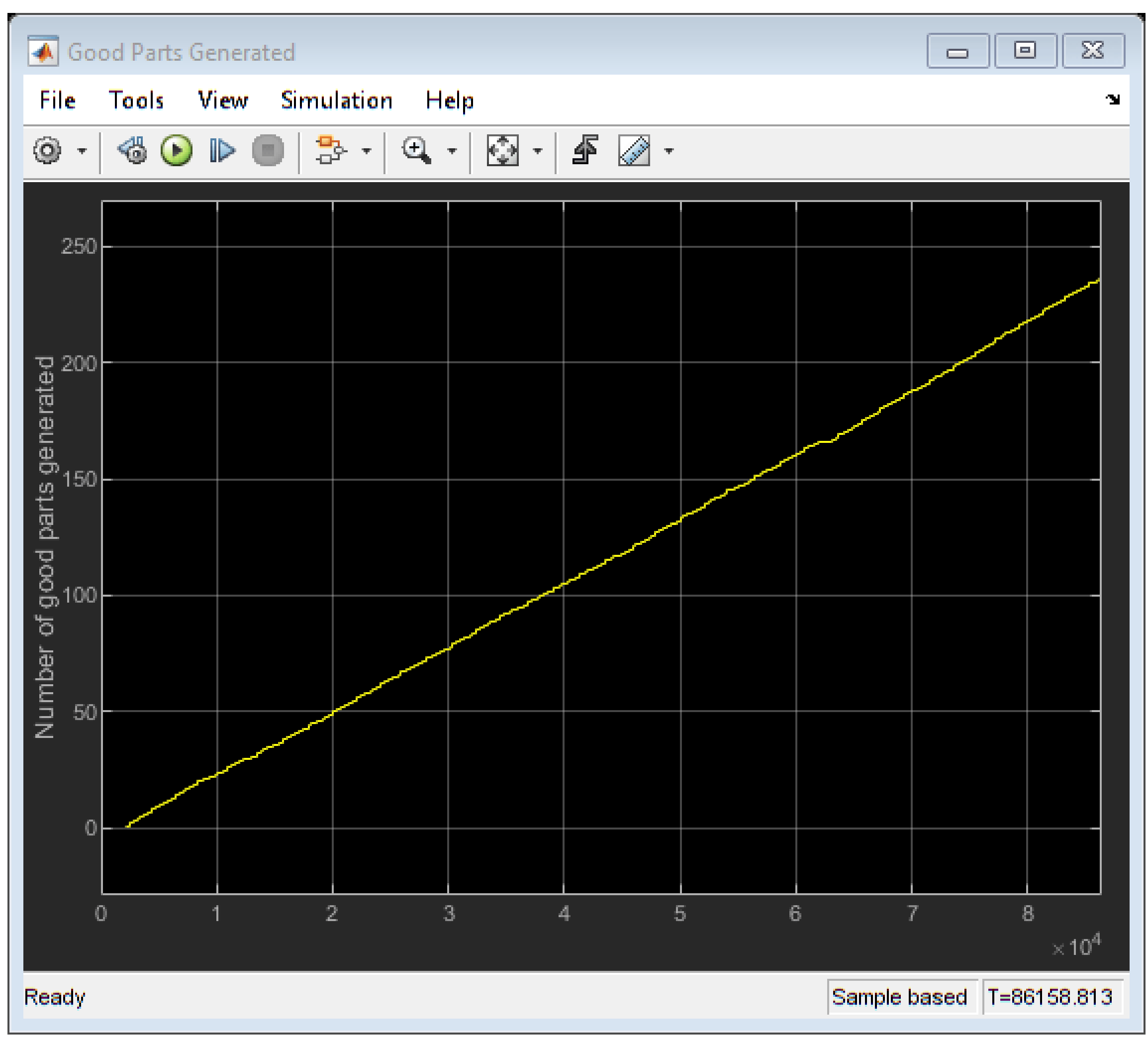

```
close_system([modelname '/Manufacturing Workers in Use']);
close_system([modelname '/Inspection workers in Use']);
close_system([modelname '/Good Parts Generated']);
```

From the scopes we see that the maximum number of workers in the Manufacturing and Inspection pools used at any given point in time rarely exceeds two. Reducing the number of workers to two shows that there is no impact on throughput with better worker utilization.

```
numMfgWorkers = 2;
numInspectWorkers = 2;

sim(modelname);
open_system([modelname '/Manufacturing Workers in Use']);
open_system([modelname '/Inspection workers in Use']);
open_system([modelname '/Good Parts Generated']);
```


Manufacturing Workers in Use
File Tools View Simulation Help
3
2.5
2
1.5
1
0.5
0
0 1 2 3 4 5 6 7 8
$\times 10^4$
Ready
Sample based
T=86250.933


Inspection workers in Use
File
Tools
View
Simulation
Help
2
1.5
1
0.5
0
0
1
2
3
4
5
6
7
8
$\times 10^4$
Ready
Sample based
T=86394.713




```
close_system([modelname '/Manufacturing Workers in Use']);
close_system([modelname '/Inspection workers in Use']);
close_system([modelname '/Good Parts Generated']);
```

### Conclusion

This example shows how we can use SimEvents to model a job shop. The use of MATLAB scripts allows us to experiment and arrive at the best schedule.

```
% The following script closes and cleans up the model
bdclose(modelname);
clear numMfgWorkers numInspectWorkers modelname excelFile ...
    scheduleID discard_rate scopes schedule requirements ...
    seed optimes parameters;
```

## See Also

Entity Server | Queue | Resource Pool | Resource Acquirer | Resource Releaser | Entity Generator

## Related Examples

- “Resource Allocation Modeling”
- “Create a Discrete-Event Model”
- “Manage Entities Using Event Actions”
- “Entity Priorities” on page 1-36

# Modeling Load Within a Dynamic Voltage Scaling Application

### Overview

This example shows how, depending on the workload, a AT90S8535 microcontroller uses a dynamic voltage scaling (DVS) feature to adjust the input voltage. By lowering the input voltage when the workload is low, the microcontroller reduces energy consumption while guaranteeing quality of service. The DVS controller is based on an online gradient estimation technique called infinitesimal perturbation analysis (IPA). In a single simulation of a parameterized system, not the large number of simulations required by a traditional finite-difference approach, IPA can provide sensitivity information that yields a first-order approximation of the system performance metrics as a function of the parameters.

### Applying IPA to the Controller

The performance metric to minimize is the average cost per job, given by

$$J(\theta) = wP(\theta) + S(\theta) = wc_2\left[V_t/\left(1 - c_1/\theta\right)\right]^2 + S(\theta)$$

where

- $\theta$ is the average service time of a job, which is a function of the input voltage V. That is, finding the optimal value of $\theta$ also yields the optimal value of V.
- $w$ is a weighting constant.
- $P(\theta)$ is the average energy consumption of a job in Joules.
- $S(\theta)$ is the average system time for jobs, which measures quality of service. This model uses an M/M/1 queuing system, so a closed-form expression for $S(\theta)$ provides a way to compare the IPA results in the simulation with theoretical results.
- $c_1$ and $c_2$ are device-dependent constants.
- $V_t$ is the device minimum input voltage.

To find a value of $\theta$ for which $dJ/d\theta$ is 0, this model uses a gradient method with constant step size $\Delta = 10^{-5}$. The $k$ th iteration of the optimization, which occurs upon the departure of the $k$ th job, uses the estimate $\theta_k$ to produce

$$\theta_{k+1} = \theta_k - \Delta \cdot \left.\frac{dJ}{d\theta}\right|_{\theta_k} = \theta_k - \Delta \cdot \left( \frac{2wc_1c_2V_t^2\theta_k}{(\theta_k - c_1)^3} + \text{IPA estimation of } \left.\frac{dS}{d\theta}\right|_{\theta_k} \right)$$

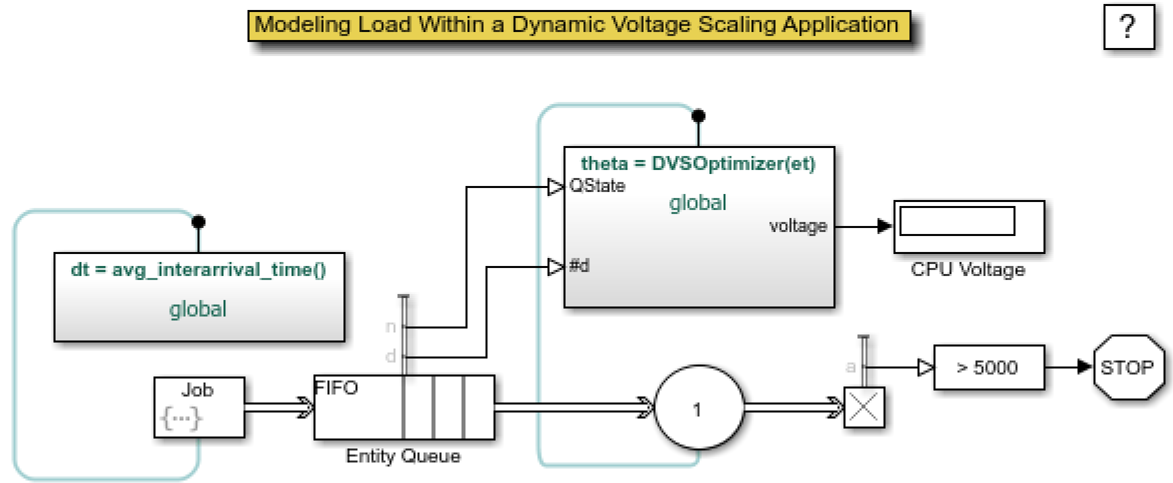
To learn about the IPA estimation of $dS/d\theta$ , see the works listed in References.

### Structure of the Model

The model includes these components:

- Job Arrivals section: Provides source of jobs that form the workload

- FIFO Queue, Single Server, and other blocks in the blue section: Provides queuing for jobs in the system
- DVS Optimizer subsystem: Uses the queue length, $\theta_k$ value, service time for the latest job, and total number of jobs to compute $\theta_{k+1}$ and the corresponding updated input voltage.

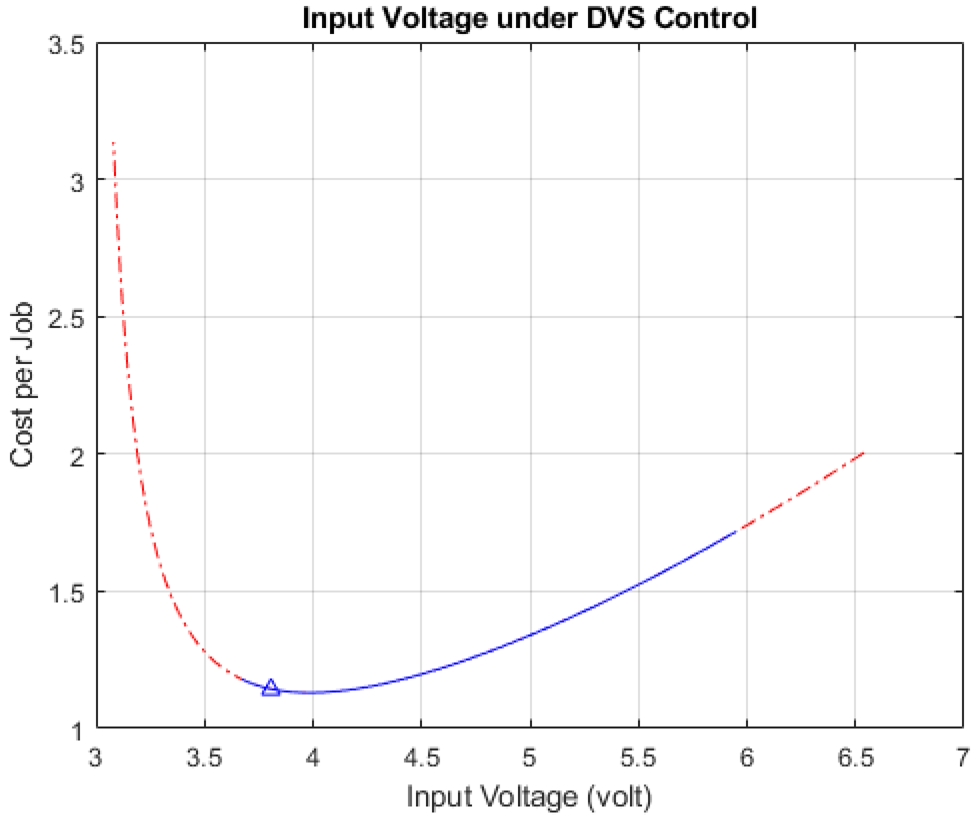



### Results and Displays

The model includes these visual ways to understand its performance:

- A dynamic plot showing how the DVS controller varies the voltage during the simulation to reduce the average cost per job.
- A Display block that shows the average service time for jobs.
- A Display block that shows the corresponding input voltage.

To experiment, try changing the value of the Avg Interarrival Time block before running the simulation.




**References**

[1] Cassandras, C. G., and S. Lafortune. *Introduction to Discrete Event Systems*. Boston, MA: Kluwer Academic Publishers, 1999.

[2] Li, W., C. G. Cassandras, and M. I. Clune. "Model-Based Design of a Dynamic Voltage Scaling Controller Based on Online Gradient Estimation Using SimEvents." *Proceedings of 45th IEEE Conference on Decision and Control*. 2006, pp. 6088-6092.

[3] Weiser, M., B. Welch, A. Demers, and S. Shenker. "Scheduling for Reduced CPU Energy." *Proceedings of the 1st Symposium on Operating Systems Design and Implementation*. 1994, pp. 13-23.

## See Also

Entity Server | Queue | Resource Pool | Resource Acquirer | Resource Releaser | Entity Generator

## Related Examples

- “Adjust Entity Generation Times Through Feedback” on page 6-11
- “Create a Discrete-Event Model”
- “Manage Entities Using Event Actions”
- “Entity Priorities” on page 1-36
- “Resource Allocation Modeling”

# Modeling Machine Failure

### Overview

This example shows how to model random failures and scheduled maintenance of a machine during regular operation. In the model, the machine can transition between three different states.

- Regular operation
- Planned maintenance
- Random failure

In the regular operation state, the machine acquires a worker and processes raw materials to produce finished products. In the planned maintenance state, the machine gets into a service mode, and after a fixed service time it returns to regular operation. The machine can also sporadically breakdown and enter a random failure state. The breakdown repair time is also random and the machine returns to regular operation after the repair is complete. During the planned maintenance and the random failure states the machine acquires a serviceman.

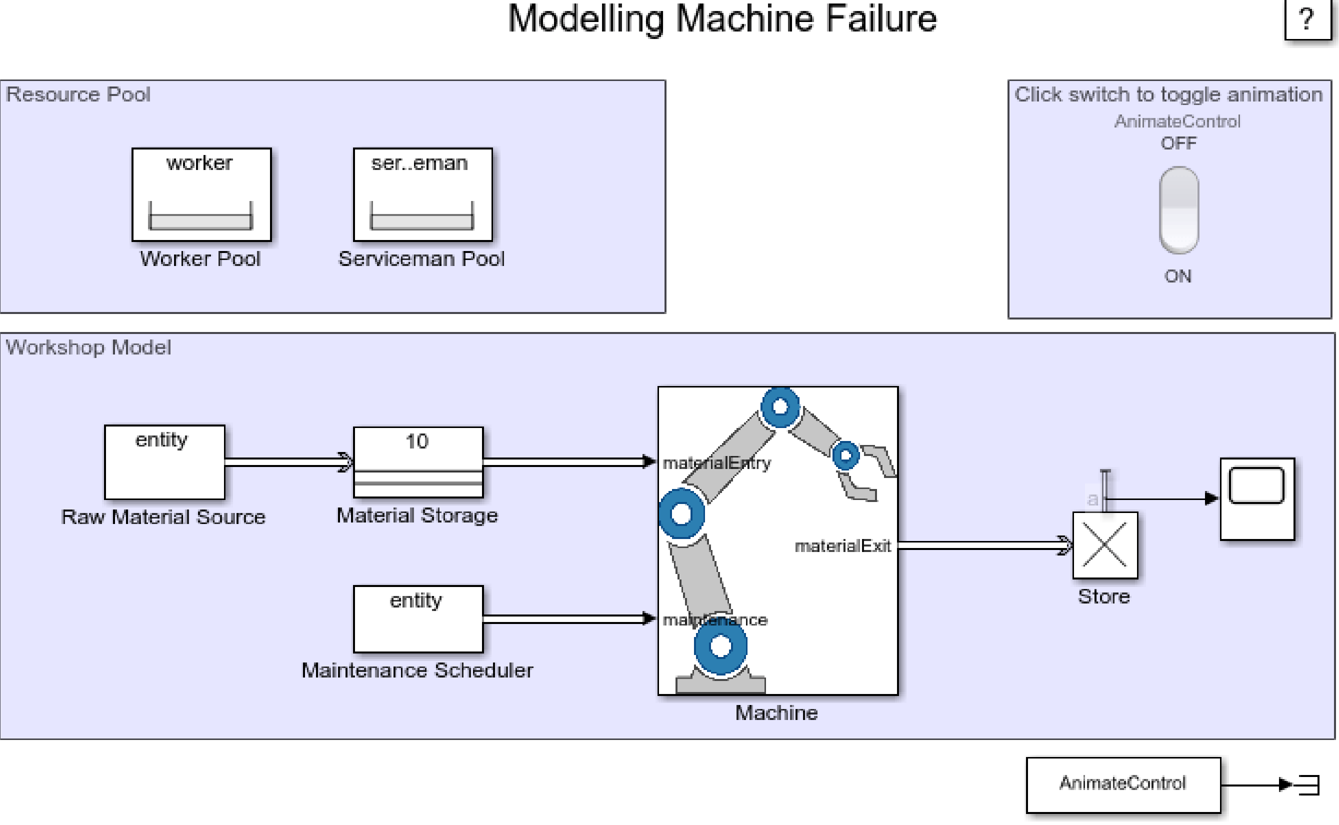
After fixing a sporadic machine failure, the following options are available to continue operation.

1. The operation resumes with the existing semi-processed material in the machine.
2. The operation resumes by discarding the semi-processed material as a waste and by taking the next raw material for processing.

### Structure of the Model

The model contains the following major components.

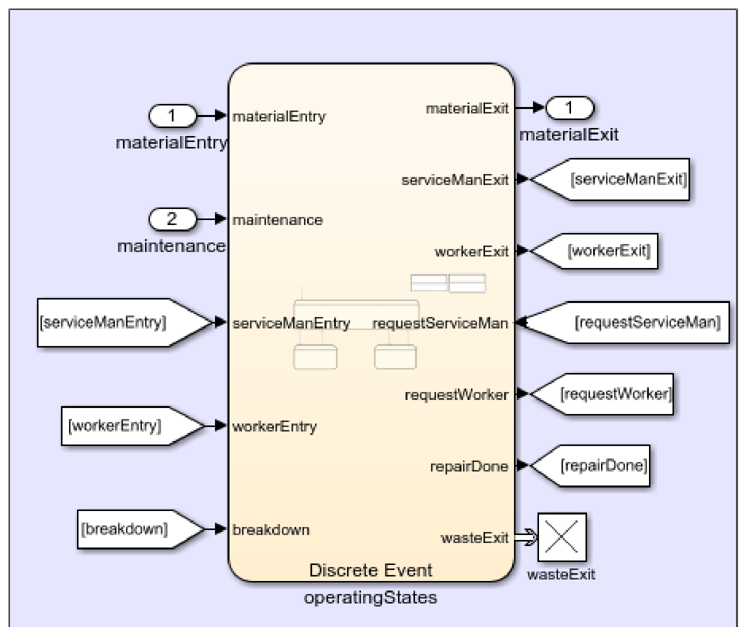
- **Raw Material Source:** Generates raw materials periodically to be sent to the storage.
- **Material Storage:** Represents the storage space of the raw materials.
- **Maintenance Scheduler:** Generates an entity to trigger a scheduled machine maintenance.
- **Machine:** Models a machine which can receive entities from Maintenance Scheduler and transitions between regular operation, planned maintenance and random failure states.
- **Store:** Represents the departure of all finished goods.
- **Worker pool:** Represents the available worker resource for regular operation.
- **Serviceman Pool:** Represents the available service man resource for scheduled maintenance and failure repair.
- **AnimateControl:** Models the switch to turn the animation on or off.

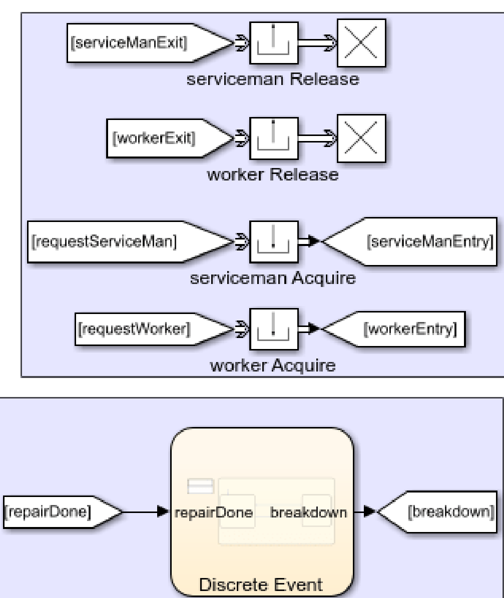



### Structure of the Machine Block

The Machine block contains two Discrete-Event chart blocks, namely 'breakdownGenerator' and 'operatingStates' along with the Resource Acquirer and the Resource Releaser blocks.

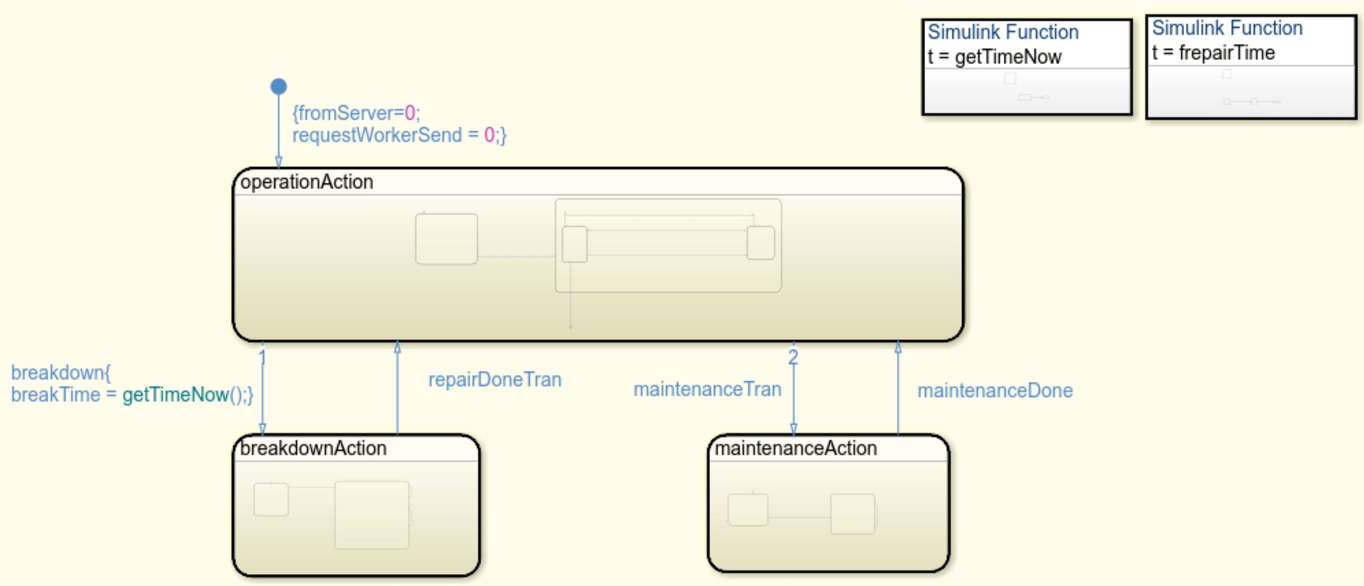
- **Breakdown Generator:** Sends a message, 'breakdown', to indicate the breakdown of the machine and accepts a message, 'repairDone', that indicates the completion of the repair. Random 'breakdown' messages are generated from a gaussian distribution.
- **Operating States:** Encapsulates three possible Machine block states which are 'breakdownAction', 'operationAction' and 'maintenanceAction'. On entering any of these states, the first action is to acquire the required resource and proceed with further actions.




### Operating States

- **Breakdown:** When entering the 'breakdownAction' state, the machine requests a serviceman who performs the repair action. After the repair is complete, the machine releases any acquired resources, and prepares to exit the 'breakdownAction' state. The random time spent for a repair is generated from a gaussian distribution. If the machine breakdown interrupts any ongoing process, after the repair, the machine either terminates it to start a new process or resumes the interrupted process. If a scheduled maintenance overlaps with a breakdown repair time, additional time is taken to complete the maintenance.
- **Maintenance:** When entering the 'maintenanceAction' state, the machine requests a serviceman who performs the service action. After the service is complete, the machine releases any acquired resources and exits the 'maintenanceAction' state. If a breakdown repair time overlaps with a service time, additional time is taken to repair the machine. In the event of a scheduled maintenance, the machine waits to complete any pending operations and only after their completion it enters the 'maintenanceAction' state.
- **Machine Operation:** When entering the 'operationAction' state, the machine requests a worker before proceeding with its operation. Then the machine fetches raw materials and switches to the processing state. After the processing is complete in a fixed duration, the machine releases the finished product and switches back to an idle state during which it waits for raw material. If the operation state is interrupted by a breakdown event, you can specify the action of the machine to resume or to terminate the operation after the repair.




### Model Parameters

- **Processing Time:** Time required to process the raw material to a finished product.
- **Maintenance Time:** Time required to service the machine during periodic maintenance.
- **Mean time between failures (MTTF) :** Average time between two consecutive breakdowns. Breakdowns are random events that are generated from a gaussian distribution.
- **Standard deviation in failure:** Standard deviation of the gaussian distribution representing breakdowns.
- **Mean time to repair (MTTR):** Average time to repair a machine in a breakdown state. Random repair time is generated from a gaussian distribution.
- **Standard deviation in repair:** Standard deviation of the gaussian distribution representing the repair time.
- **Resume operation post repair:** Checkbox to resume any pending operations in the machine after a breakdown. Otherwise, the materials are discarded and a new operation starts.

### Visualization

Enabling the toggle switch labelled 'AutomationControl' allows you to visualize the machine operations listed below.

- Raw materials are transferred to the storage, sent to the machine's processing queue, and processed by the machine into a finished product.
- Machine acquires a worker from the worker pool to operate. The worker is sent back to the pool during breakdown or maintenance.
- Machine acquires a serviceman from the pool in the event of a breakdown or a maintenance. When the service or repair is complete, the serviceman is sent back to the pool.
- The material is sent to the waste bin to be discarded.
- The animation displays the quantity of the materials that are unloaded, in storage, wasted and processed.

- The animation also displays the quantity of available workers and serviceman.

## See Also

Resource Pool | Resource Acquirer | Resource Releaser

## Related Examples

- “Discrete-Event Chart Precise Timing” on page 8-6
- “Trigger a Discrete-Event Chart Block on Message Arrival” on page 8-9
- “Entity Priorities” on page 1-36
- “Resource Allocation Modeling”

# Inventory Management

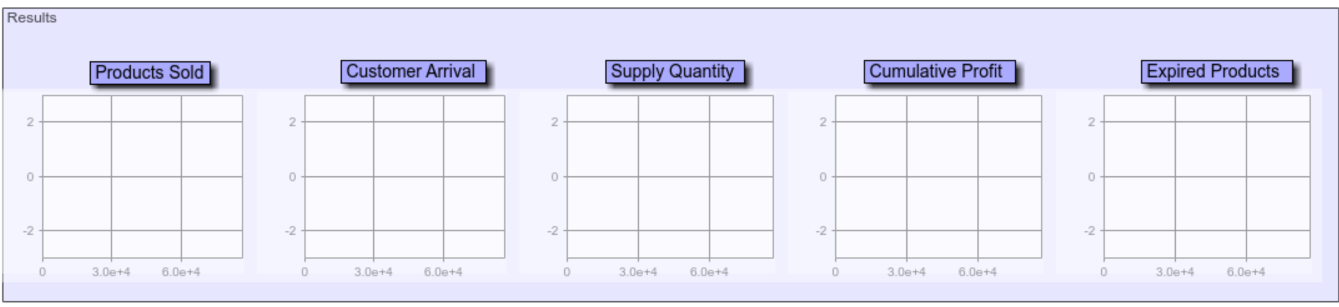
## Description

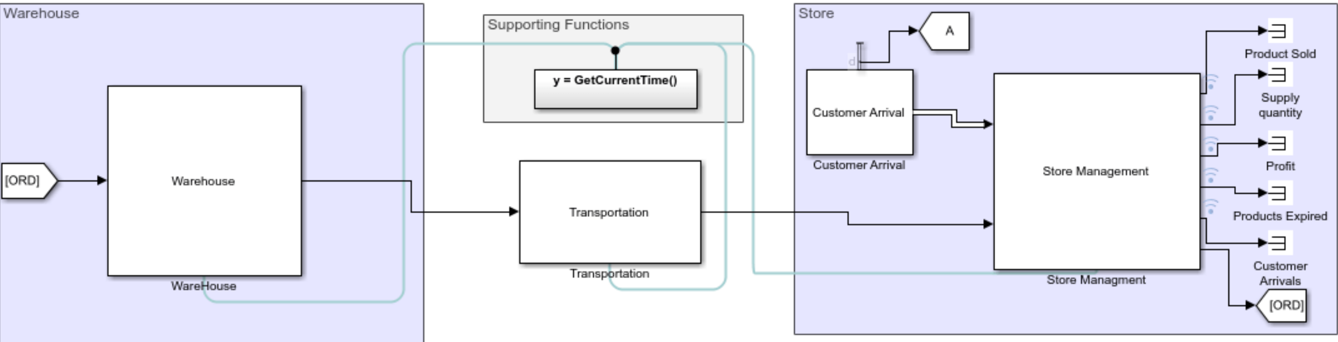
This example shows how to build a simple inventory management system for a retail store. This example includes:

- Random customer arrivals to the store with the number of products requested by each customer also randomly distributed
- Tracking available inventory at the end of the day
- Tracking and disposal of expired products
- Placing periodic orders for fresh products
- Store profitability analysis

Inventory Management

?



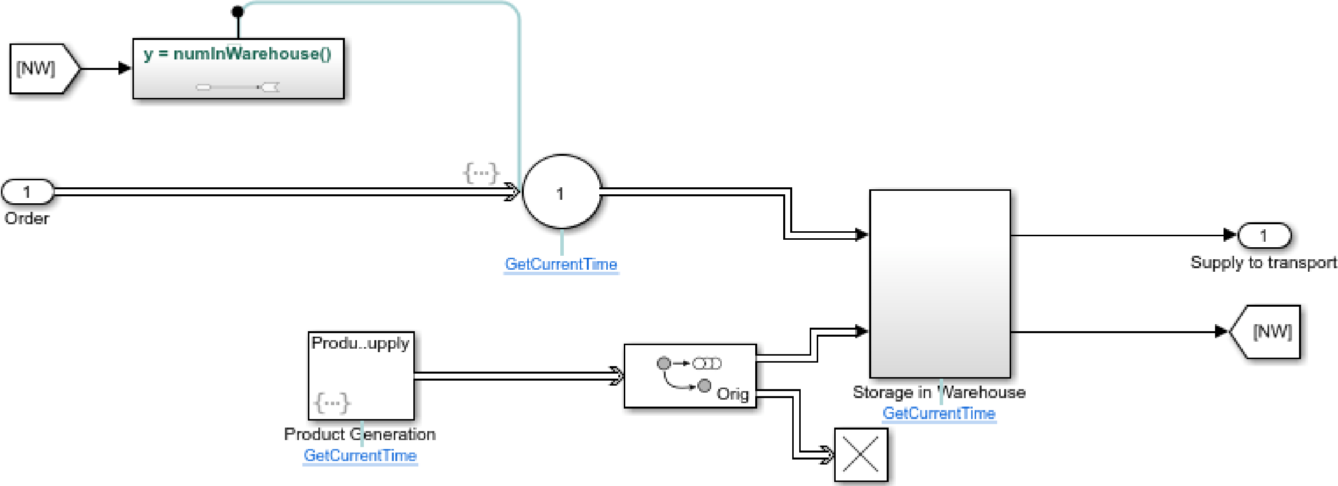




Copyright 2019 The MathWorks Inc.

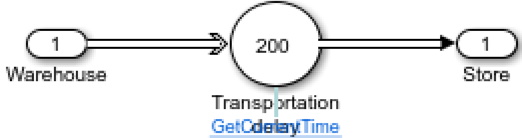
## Structure of the Model

The model includes these components:

- **Warehouse**: The warehouse generates and stores products in shelves. The products have limited shelf life and they are dispatched when a product order is received. During the generation process, products are marked with their manufacture day, and they are periodically checked for disposal if their storage duration exceeds the maximum days they are allowed to stay on the shelf. Warehouse component allows you to specify the initial quantity of available products and the maximum number of days they are allowed to remain on the shelf.

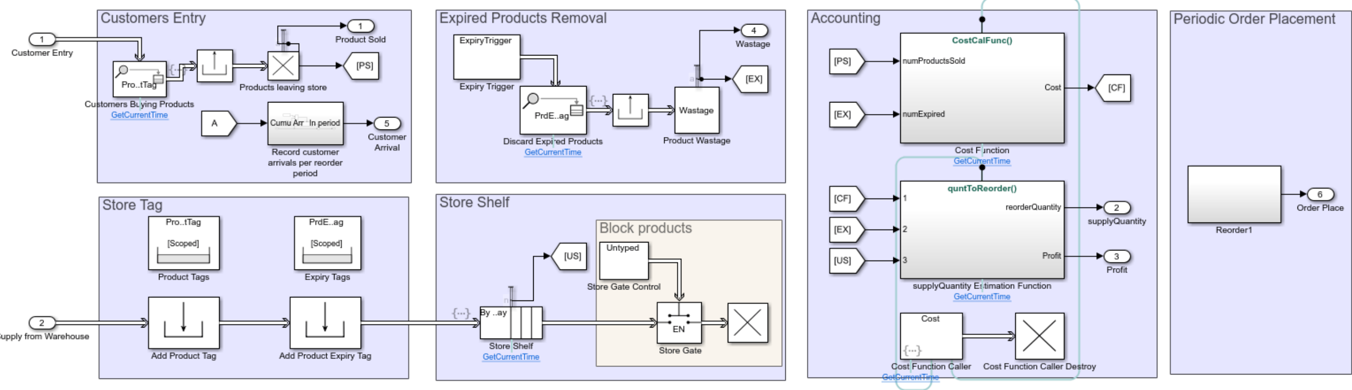



- **Transportation**: The Transportation block represents the delay, which is the duration between a product's dispatch from the warehouse and its arrival at the store. The default delay is set to two hours.




- **Customer Arrival**: The arrival of customers at the store is modeled as a Poisson process and you can specify the mean time between arrivals. The number of products required by each customer is also random and it is generated from a discrete uniform distribution. You can specify the upper bound of this uniform distribution.

- **Store Management**:




- **Store Tag**: This area models the part of the retail store that receives products from the warehouse and applies the 'Product' and 'Expiry' tags on them. These tags allow us to search for products later on.

- **Customers Entry**: Represents customers entering the store to pick up products from the shelves and their departure from the store. This is modeled using the 'Entity Find' block, which looks for entities in the system that have the 'Product' tag associated with them.
- **Store Shelf**: This area contains a Queue where the products are stored. Customers pick up products from here. An 'Entity Gate' that is perpetually closed ensures that products don't flow out of the store.
- **Expired Products Removal**: This area models the periodic removal of expired products from the store's shelves. This is modeled using the 'Entity Find' block. The find block is triggered periodically to perform a search for entities that have the 'Product' tag associated with them. It then looks for products that have exceeded the shelf life and discards them.
- **Accounting**: This area models the investigation of profitability of the retail store for the duration between consecutive product ordering points. The profit is calculated as a function of the product procurement price, product holding cost and the product selling price. The profit also plays a role in determining the number of products that the retail store orders. If the store is profitable in the current period, then the new quantity to be ordered is the sum of the previously ordered quantity plus any unfulfilled orders. This is also adjusted for expired and unsold products.
- **Periodic Order Placement**: This area models periodic order placement by the retail store. An order is placed with the Warehouse for supplying a fresh batch of products to the retail store. You can specify the period by setting the value of the reorder point.

#### Results

The model is simulated for 60 days. One unit of simulation time represents 1 minute of wall clock time. Based on the model parameters set, plots are generated showing the number of products sold, the number of customers who arrived at the store, the product order size, number of expired products in the store and the store's profitability. Observe that for each period, the optimal store order quantity is around 85 for the given customer arrival rate.

## See Also

Entity Server | Queue | Resource Pool | Resource Acquirer | Resource Releaser | Entity Generator

## Related Examples

- “Create a Discrete-Event Model”
- “Manage Entities Using Event Actions”
- “Entity Priorities” on page 1-36
- “Resource Allocation Modeling”

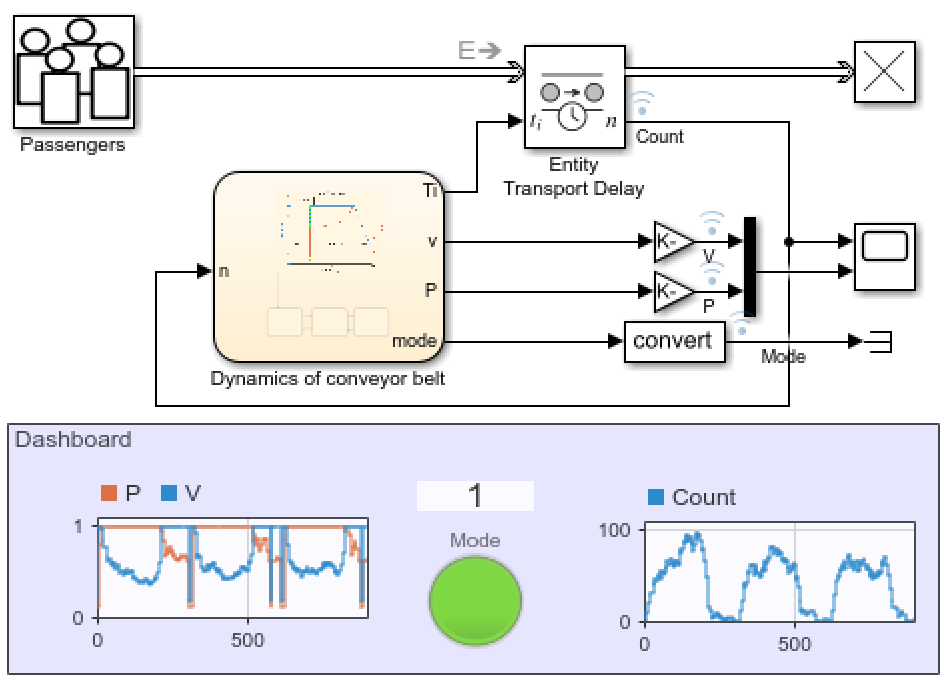
# Modeling Cyber-Physical Systems

Cyber-physical systems combine computer and physical systems to achieve design goals. Simulation of cyber-physical systems requires a combination of modeling techniques such as continuous-time, discrete-time, discrete-event, and finite state modeling. Simulink® and its companion products provide functionality to apply a wide range of modeling techniques and seamlessly integrate them in one simulation environment, which is ideal for modeling cyber-physical systems.

This example shows how continuous-time, discrete-event, and finite-state modeling techniques combine to simulate the behavior of a variable speed conveyor belt system. In SimEvents®, entities are discrete items of interest in a discrete-event simulation. Because passengers are discrete individuals, they are modeled by SimEvents® entities, created by the Entity Generator block. A Stateflow® chart models the operational modes and motor dynamics of the variable speed conveyor belt. Finally, the Entity Transport Delay block models passenger throughput as a function of conveyor belt dynamics, providing a bridge between the discrete-event and continuous-time domains.

**Note:** The example uses blocks from SimEvents® and Stateflow®. If you do not have a SimEvents or Stateflow license, you can open and simulate the model but only make basic changes such as modifying block parameters.

### Model Structure




The model includes these key components:

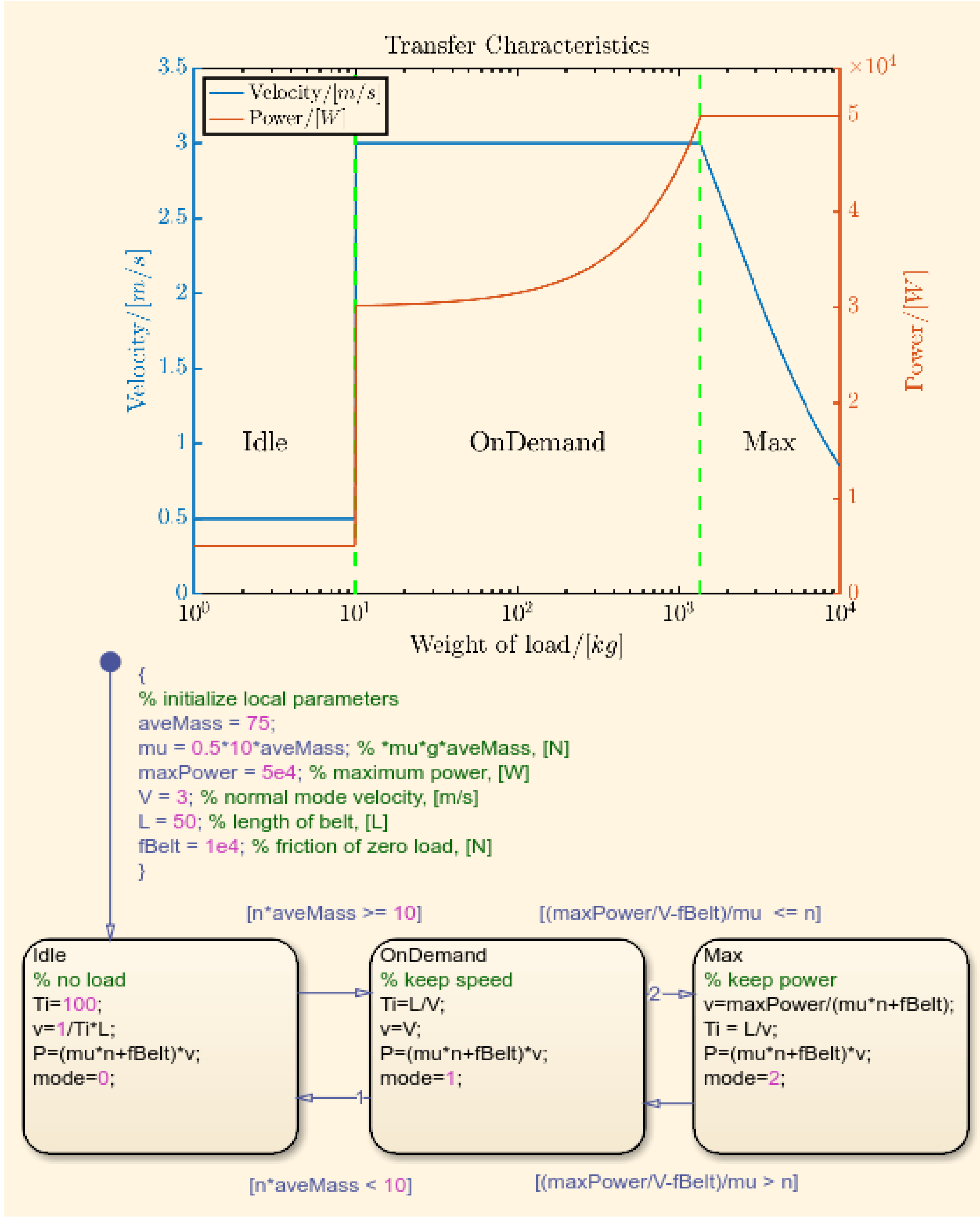
- **Passengers** — Models the arrival of passengers as a Poisson process. The output is a sequence of SimEvents® entities corresponding to the passengers who step on the conveyor belt. The

distribution of inter-arrival time ($\Delta t$) of a Poisson process is $P(\Delta t) = \lambda e^{-\lambda \Delta t}$, where $\lambda$ is the arrival rate. $\lambda$ is modeled by a MATLAB action in the Entity Generator block for *rush hour, normal hour,* and *free hour*. The passenger arrival rate changes with time as:

$$\lambda(t) = \begin{cases} 2, & mod(t,300) \subset [0,180), \quad \text{rush hour} \\ 0.5, & mod(t,300) \subset [180,240), \quad \text{normal hour} \\ 0.1, & mod(t,300) \subset [240,300), \quad \text{free hour} \end{cases}$$

- **Entity Transport Delay** — Holds the passengers on the conveyor belt until they arrive at the other terminal, based on the time delay calculated by the Stateflow chart.
- **Dynamics of conveyor belt** — Models the operation of a variable speed conveyor belt. See the Conveyor Belt Dynamics section for more details.
- **Dashboard** — Shows the runtime status of the conveyor belt. The color of the `Mode` Lamp indicates the mode of the conveyor belt.

#### Conveyor Belt Dynamics




A Stateflow® chart models the dynamics of the variable speed conveyor belt. Note in the chart that the velocity and power of the belt are plotted against a logarithmic scale of the load weight. The conveyor belt has these modes:

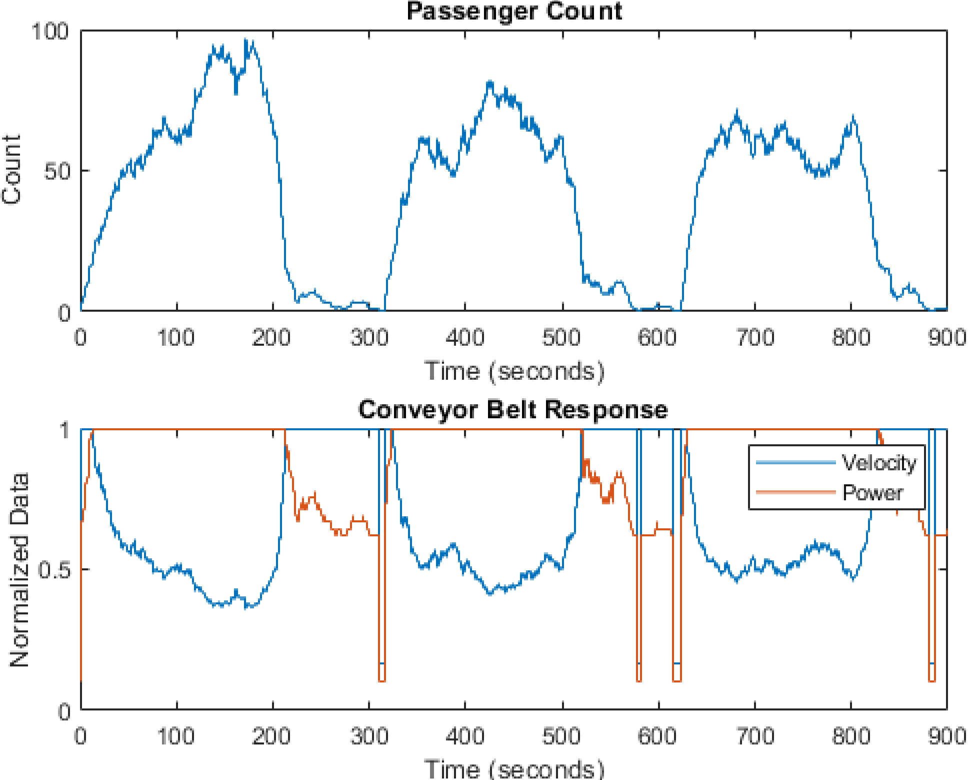
- **Idle** — The weight of the load is small. The belt maintains a low velocity to save energy. The `Mode Lamp` is gray in this mode.

- **OnDemand** — This is the normal operating mode, which maintains the optimal velocity for passenger comfort and throughput. The power will proportionally increase with the weight of the load. The `Mode` Lamp is green in this mode.
- **Max** — Maximum power mode. The weight of the load is too large for the conveyor belt to maintain the optimal velocity. The conveyor belt operates at the maximum possible velocity that does not exceed the maximum power. The `Mode` Lamp is red in this mode.

#### Results

The Scope and blocks in the DashBoard show the simulation results.

Simulation results: 1. Number of passengers versus simulation time. 2. Velocity (blue) and power (red) versus simulation time.




Three operation cycles are observed within a time span of 900. Each cycle has a period of 300, which aligns with the period of the arrival rate. The top plot shows the number of passengers on the conveyor belt over time, and the bottom plot shows the velocity and power of the conveyor belt. The velocity and power are normalized for better visualization.

The first two thirds of each period correspond to *rush hour*, and the number of passengers on the conveyor belt increases dramatically. Consequently, the conveyor belt enters into the **Max** mode quickly, which is characterized by the maximum output power with a velocity that is inversely proportional to the number of passengers. In the last third of each period, the airport is in the *normal hour* followed by the *free hour*. Therefore, the number of passengers on the conveyor belt drops and even becomes zero for some time.

The conveyor belt then operates in **OnDemand** and **Idle** modes accordingly. In **OnDemand** mode, the velocity is locked to a default value, and the power is proportional to the number of passengers. In **Idle** mode, both the velocity and power are maintained at low values to reduce energy consumption. Overall, the conveyor belt operates according to the load of the airport.

# 802.11 MAC and Application Throughput Measurement

This example shows how to measure the MAC and application layer throughput in a multi-node 802.11a/n/ac/ax network using SimEvents®, Stateflow®, and WLAN Toolbox™. The system-level model presented in this example includes functionalities such as configuring the priority of the traffic at the application layer, capability to generate and decode waveforms of Non-HT, HT-MF, VHT, HE-SU and HE-EXT-SU formats, MPDU aggregation and enabling block acknowledgment of MPDUs. The application layer throughput calculated using this model is validated against published calibration results from the TGax Task Group [ 4 ] for Box 3 scenarios (Tests 1a, 1b, and 2a) specified in TGax evaluation methodology [ 3 ]. The obtained application layer throughput is within the range of minimum and maximum throughput specified in published calibration results [ 4 ].

### Throughput in 802.11 Networks

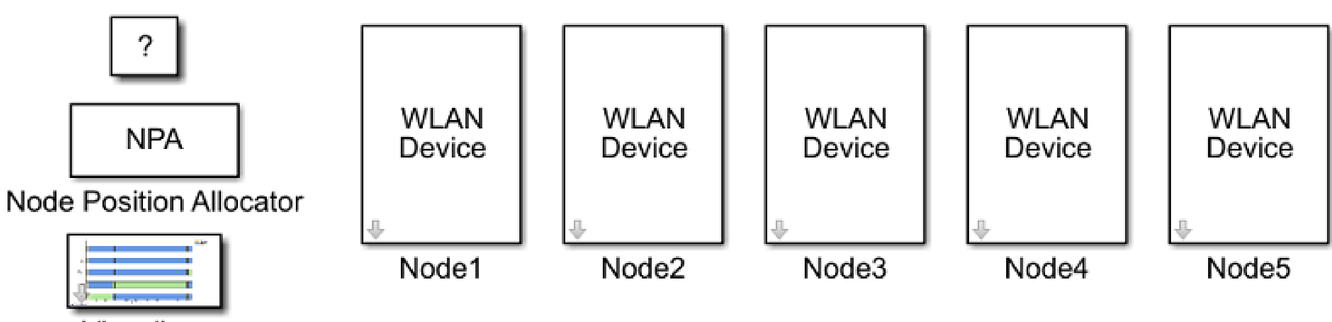
The IEEE® 802.11™ working group is continually adding features to 802.11 specification [ 1 ] to improve the throughput and reliability in WLAN networks. Throughput is the amount of data transmitted over a period of time. Medium Access Control (MAC) layer throughput refers to the amount of data successfully transmitted by the MAC layer over a period of time. MAC protocol data unit (MPDU) is the unit of transmission at MAC layer. In 802.11n, MPDU aggregation was introduced to increase the throughput. When MPDU aggregation is supported, MAC layer aggregates multiple MPDUs into an aggregated MPDU (A-MPDU) for transmission. This reduces the overhead of channel contention for transmitting multiple frames, resulting in enhanced throughput. In 802.11ac [ 1 ] and 802.11ax [ 2 ], the maximum limits for an A-MPDU length were increased resulting in even better throughput in WLAN networks.

### Model 802.11 Network

This example models a WLAN network with five nodes as shown in this figure. These nodes implement carrier-sense multiple access with collision avoidance (CSMA/CA) with physical carrier sense and virtual carrier sense. The physical carrier sensing uses the clear channel assessment (CCA) mechanism to determine whether the medium is busy before transmitting. Whereas, the virtual carrier sensing uses the RTS/CTS handshake to prevent the hidden node problem.

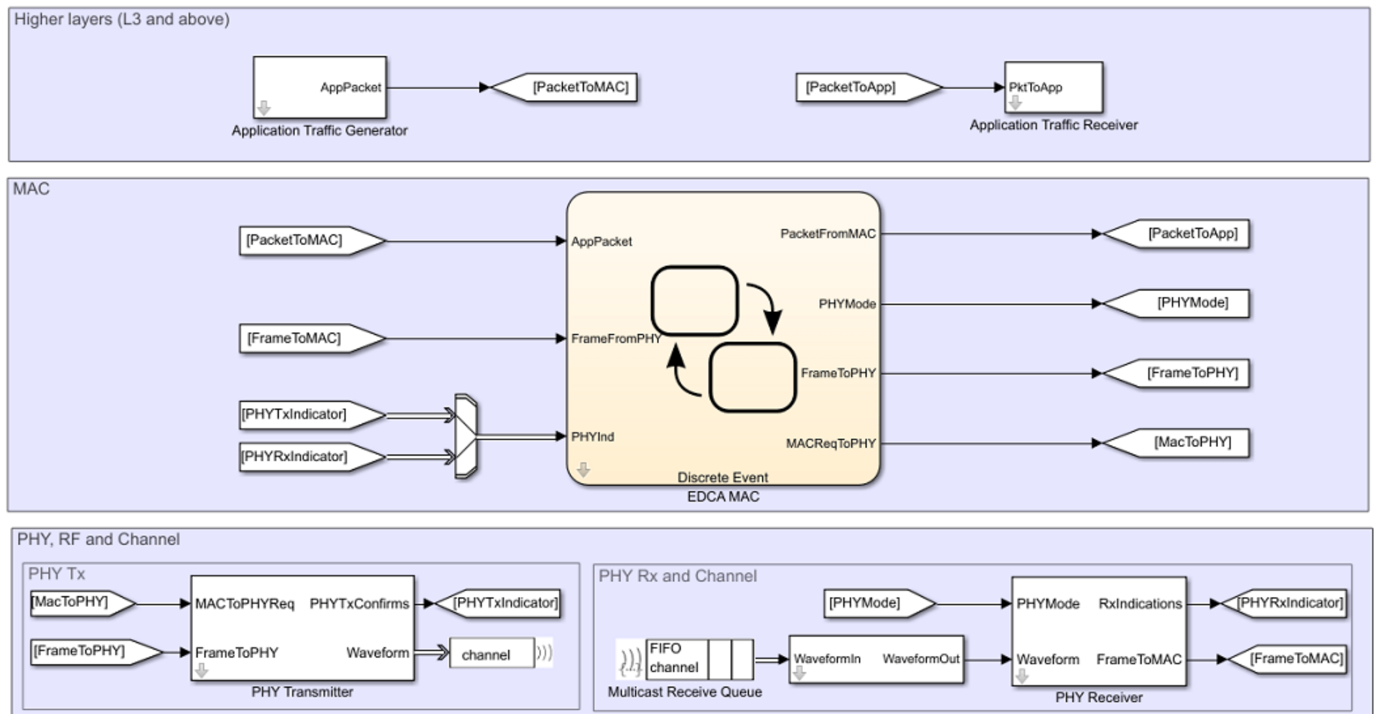
The model in the example displays various statistics such as the number of transmitted, received, and dropped packets at PHY and MAC layers. Moreover, the runtime figures that help in analyzing/estimating the node-level and network-level performance are also displayed in this model. This model is validated against the published calibration results from the TGax Task Group [ 4 ] for Box 3 scenarios (Tests 1a, 1b, and 2a) specified in TGax evaluation methodology [ 3 ].

**WLAN Network**



### Components of a WLAN Node

The components of a WLAN node are shown in this figure. The information is retrieved by pressing the arrow button for each node in the above figure.

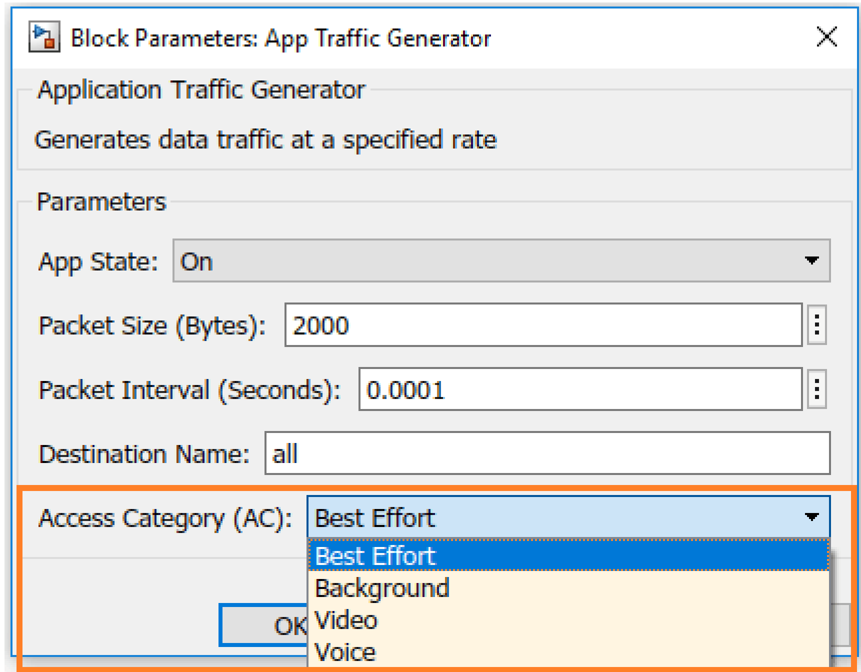



### Application, EDCA MAC and PHY Block Enhancements

This example is an enhancement over “Multi-Node 802.11a Network Modeling with PHY and MAC” (WLAN Toolbox) example. Refer to the example documentation page for more information about each layer in a WLAN node. The application, the EDCA MAC and the PHY blocks used in this example has these enhancements over “Multi-Node 802.11a Network Modeling with PHY and MAC” (WLAN Toolbox).

#### Application:

The application layer has the capability to generate data with different priority levels as shown in this figure. These priority levels are configured using `Access Category` property in the mask parameters of the Application Traffic Generator block inside a WLAN node.

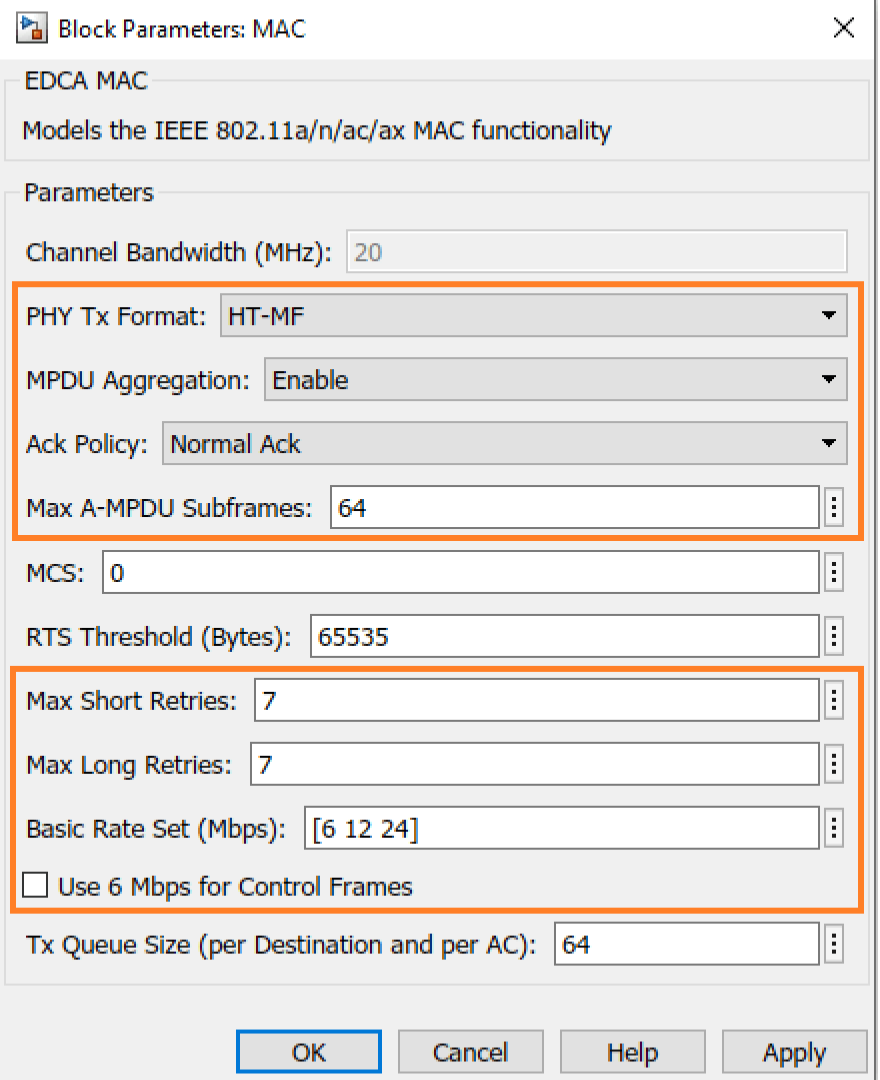



**EDCA MAC:**

The EDCA MAC block used in this example has these enhancements over MAC block used in “Multi-Node 802.11a Network Modeling with PHY and MAC” (WLAN Toolbox) example

- Generate and decode MAC frames of high efficiency single user (HE-SU), high efficiency extended range single user (HE-EXT-SU), very high throughput (VHT), high throughput mixed format (HT-MF) and Non-HT formats. These formats are configured using the `PHY Tx Format` property in the mask parameters of the MAC EDCA block inside a WLAN node as shown in this figure.
- Aggregate MPDUs to form an A-MPDU. This can be configured by setting `PHY Tx Format` to one of `HT-MF`, `VHT`, `HE-SU`, or `HE-EXT-SU`. In case of `HT-MF`, `MPDU Aggregation` property must also be enabled for A-MPDU generation.
- Acknowledge multiple MPDUs in an A-MPDU with a single block acknowledgment (BA) frame. MAC assumes a pre-configured BA session between the transmitter and the receiver of an A-MPDU.
- Enable/disable acknowledgments. This can be configured using the `Ack Policy` property.
- Maintain separate retry limits for shorter frames (less than RTS threshold) and longer frames (greater than or equal to RTS threshold). These limits can be configured using the `Max Short Retries` and `Max Long Retries` properties.
- Transmit multiple streams of data using the multiple-input multiple-output (MIMO) capability. You can configure this capability using the `Number of Transmit Chains` property. This property is applicable only when the value of `PHY Tx Format` property is set to `VHT`, `HE-SU`, or `HE-EXT-SU`. The MIMO capability can also be used for `HT` format through the `MCS` property. The range of values [0, 7], [8, 15], [16, 23], and [24, 31] correspond to one, two, three, and four streams of data respectively.
- Adapt the data rate according to the channel conditions through the `Rate Adaptation Algorithm` property. This is applicable only when the value of `PHY Tx Format` property is set to `Non-HT`. You can choose between `Auto Rate Fallback (ARF)` and `Minstrel` algorithms. To maintain a constant data rate throughout the simulation, `Fixed-Rate` option is available.

- Enable parallel transmissions between the basic service sets (BSSs) through the `Enable Spatial Reuse with BSS Color` property. This property is applicable only when `PHY Tx Format` property is set to `HE-SU`, `HE-EXT-SU`, or `HE-MU-OFDMA`. This model does not support the spatial reuse (SR) functionality. To study the impact of SR with BSS coloring on the network throughput, refer "Spatial Reuse with BSS Coloring in 802.11ax Residential Scenario" (WLAN Toolbox) example.




**PHY:**

Capability to generate and decode waveforms of Non-HT, HT-MF, VHT, HE-SU and HE-EXT-SU formats

**Throughput Measurement**

Throughput varies for different configuration parameters pertaining to the application, MAC & PHY layers. Any change in the configuration may either increase or decrease the throughput. You can vary the combination of these parameters to measure and analyze the throughput.

- `MCS`: PHY data rate
- `PHY Tx Format`: PHY transmission format
- `Packet Size`: Application packet size
- `Max A-MPDU Subframes`: Maximum number of subframes in an A-MPDU
- `Max Tx Queue Size`: MAC transmission queue size

Along with above parameters, you can also vary the node positions, Tx & Rx gains, channel loss, number of nodes in the network, MAC contention parameters, number of transmit chains and rate adaptation algorithms to analyze MAC throughput. This example demonstrates the measurement and analysis of the MAC throughput by varying packet size in the `Application Traffic Generator` block.

**Application Packet Size**

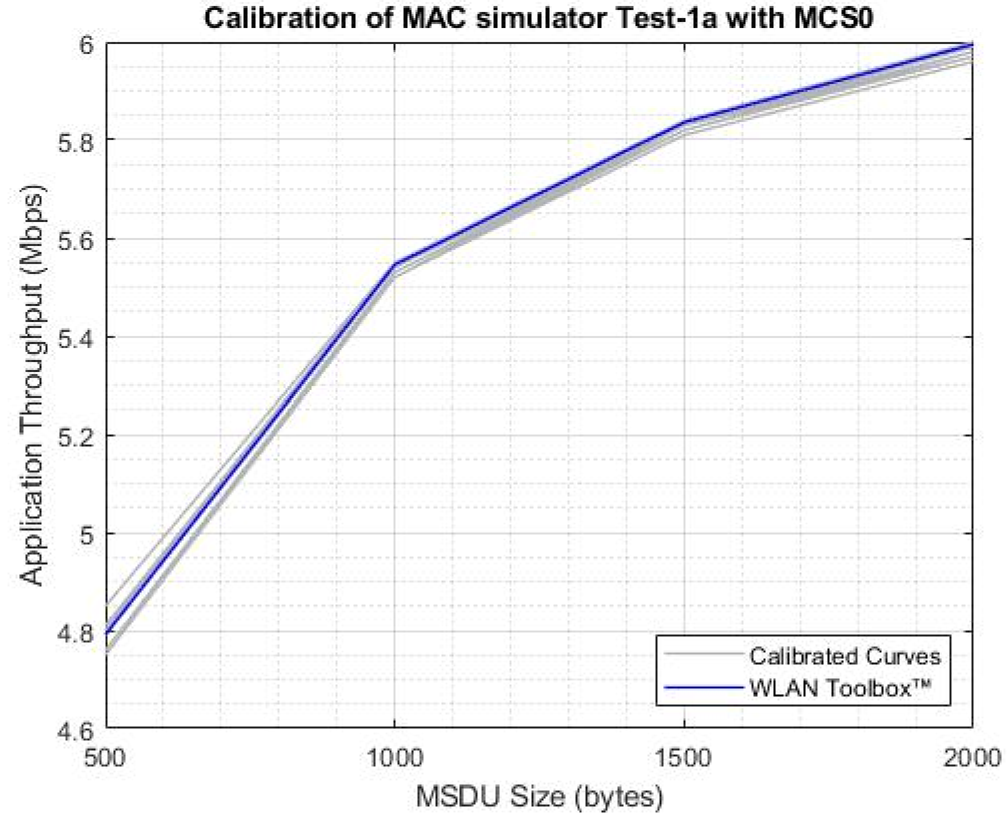
Throughput is directly proportional to the application packet size. Smaller packet size results in greater number of packets to be transmitted. At the MAC layer, there is an overhead of contention time for each transmitted packet. This is because the MAC layer makes sure that the channel is idle for a specific amount of time (Refer section 10.3.2.3 of [ 1 ]) before transmitting any packet. Therefore, as the packet size decreases, the contention overhead increases resulting in lower throughput.

**Model Configuration**

You can configure the application packet size using these steps:

1. Open model `WLANMACThroughputMeasurementModel.slx`
2. To go inside a node subsystem, click on the downward arrow at the bottom left of the node
3. To open mask parameters of the application, double click on `Application Traffic Generator`
4. To enable application, set `App State` to 'On'
5. Configure the value of `Packet Size`

Run the simulation and observe the throughput. The TGax calibration results for test-1a in [ 4 ] are shown below:



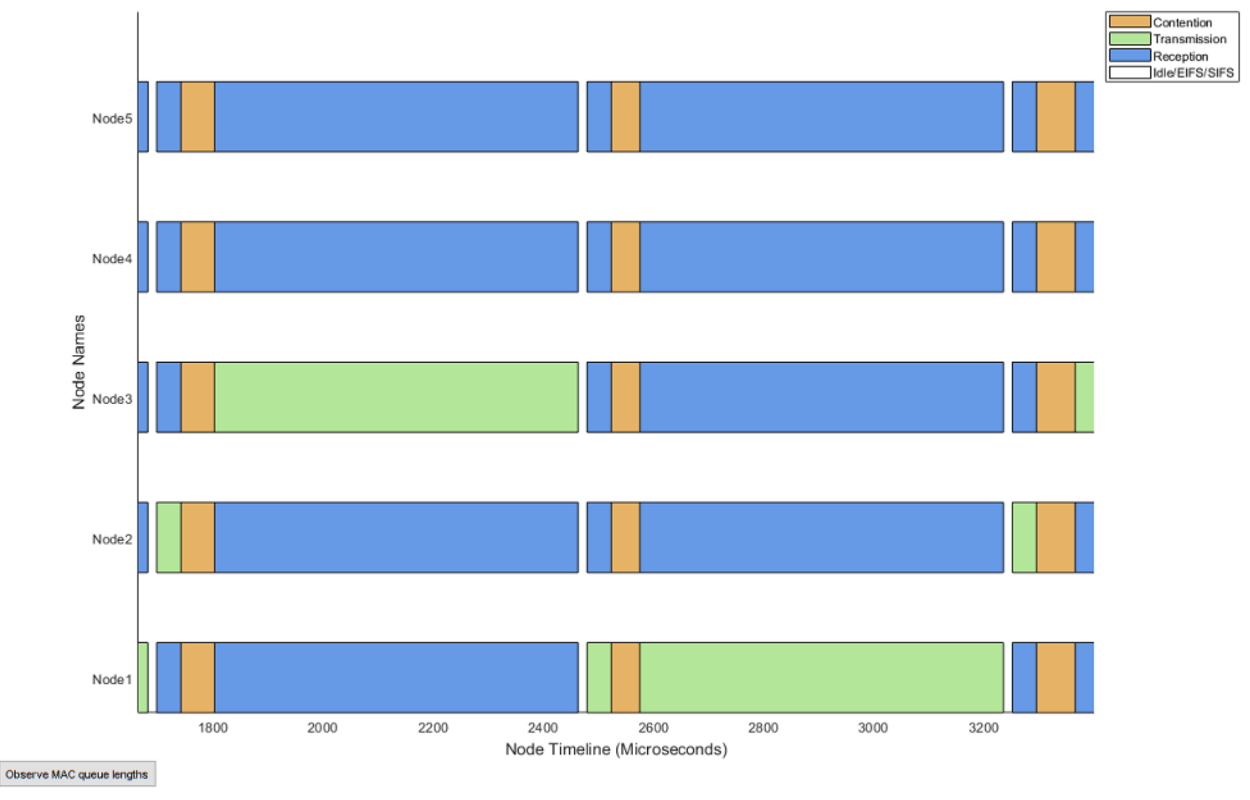

The above plot compares the calibration results for WLAN Toolbox against the published results of other companies listed in [ 4 ]. The blue colored curve represents the results of WLAN Toolbox, while the grey colored curves represent the results of other companies.

**Simulation Results**

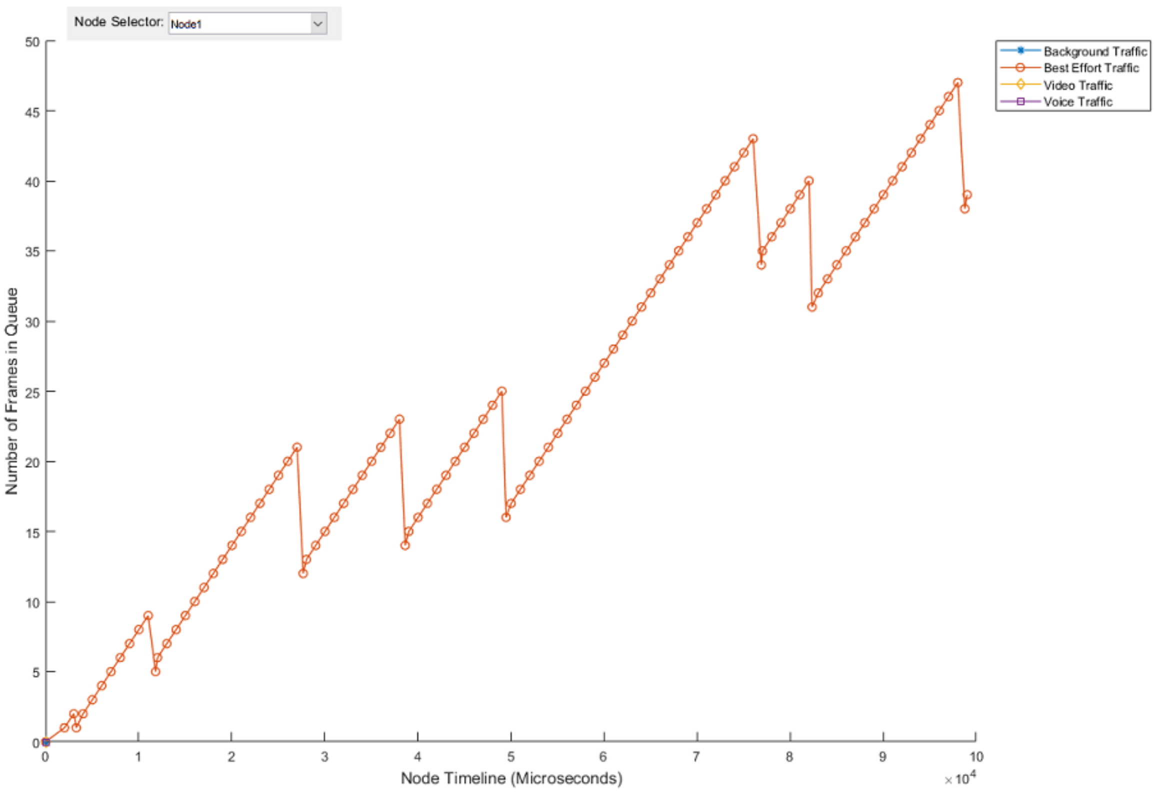
The simulation of the model generates:

1. A run-time visualization showing the time spent on channel contention, transmission, and reception for each node
2. An optional run-time visualization (during the simulation) showing the number of frames queued in MAC transmission queues for a selected node.
3. A bar graph showing metrics for each node such as number of transmitted, received, and dropped packets at PHY and MAC layers
4. A MAT file `statistics.mat` with detailed statistics obtained at each layer for each node

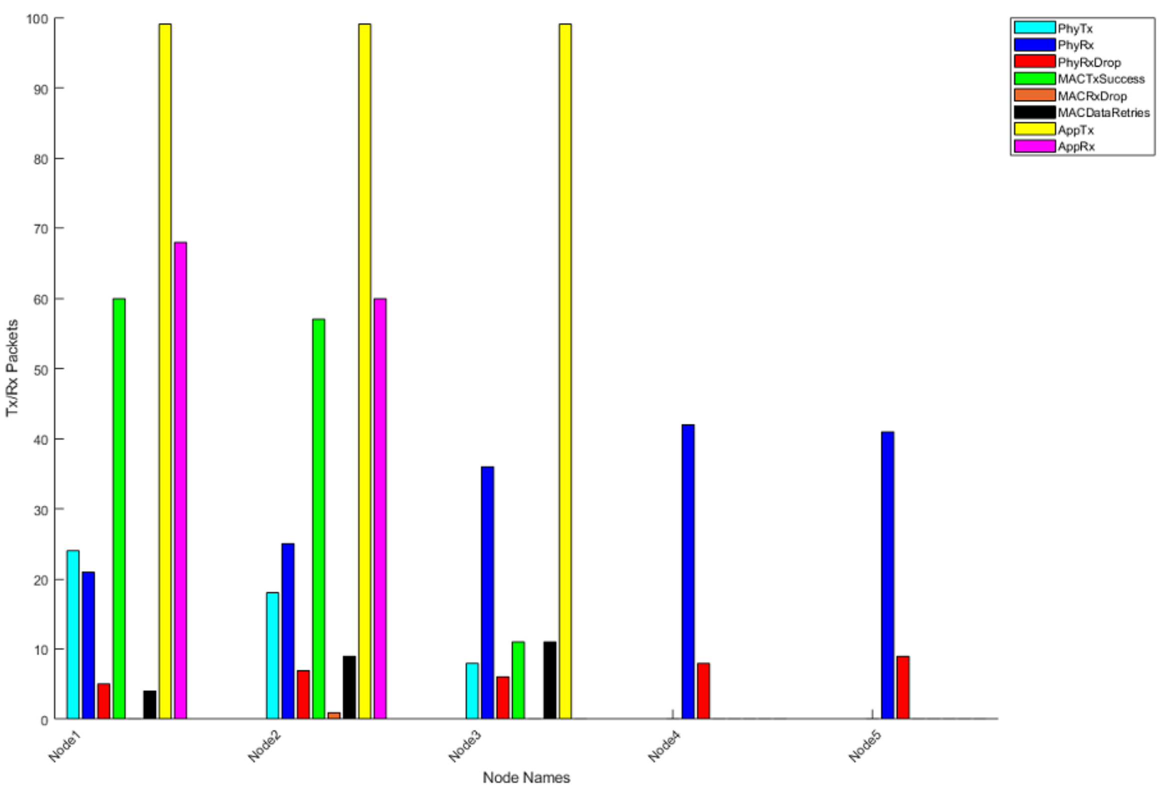
This figure shows MAC state transitions with respect to simulation time.




You can also observe the live state of the MAC layer transmission buffers using the 'Observe MAC queue lengths' button in the above visualization.




This figure shows the network statistics at the end of simulation.




#### Validating Application Layer Throughput with TGax Calibration Results

The TGax Task Group [ 4 ] published application throughput results for different scenarios. You can observe the Layer 3 (above MAC layer) throughput of each node in the network in 'Throughput' column in 'statisticsTable' stored in 'statistics.mat'. The TGax calibration scenarios for MAC simulator published results of application throughput for a User Datagram Protocol (UDP) with Logical Link Control (LLC) layers overhead.

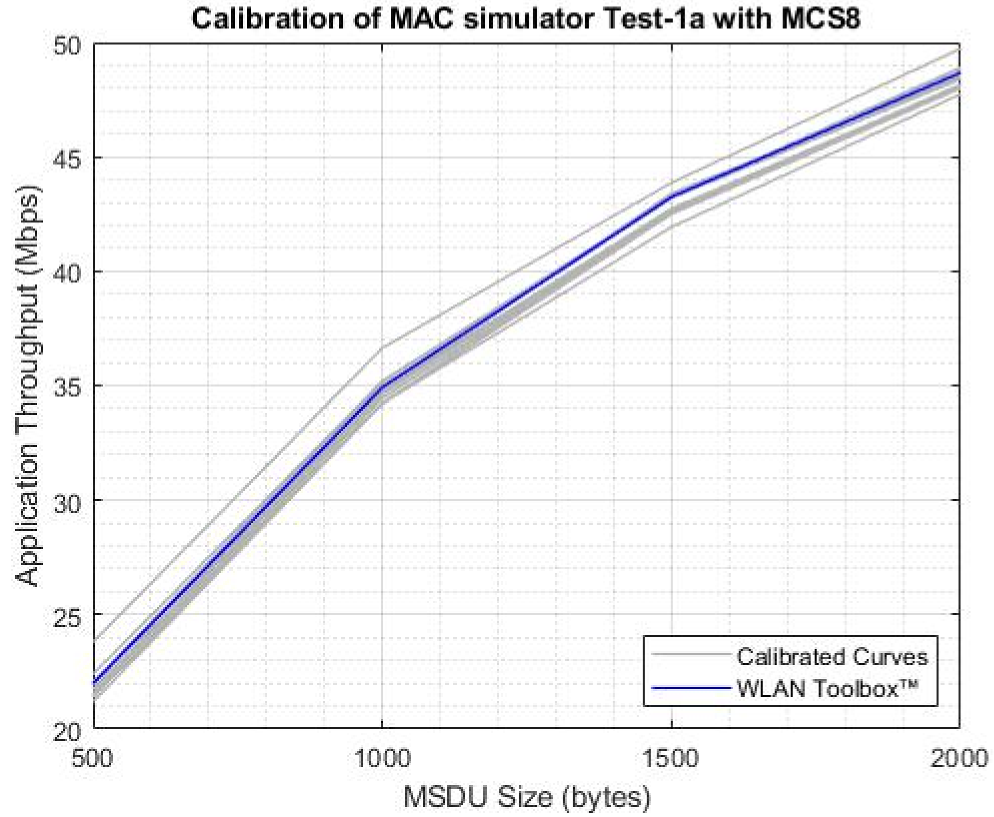
To calculate application throughput from simulation results use the code below:

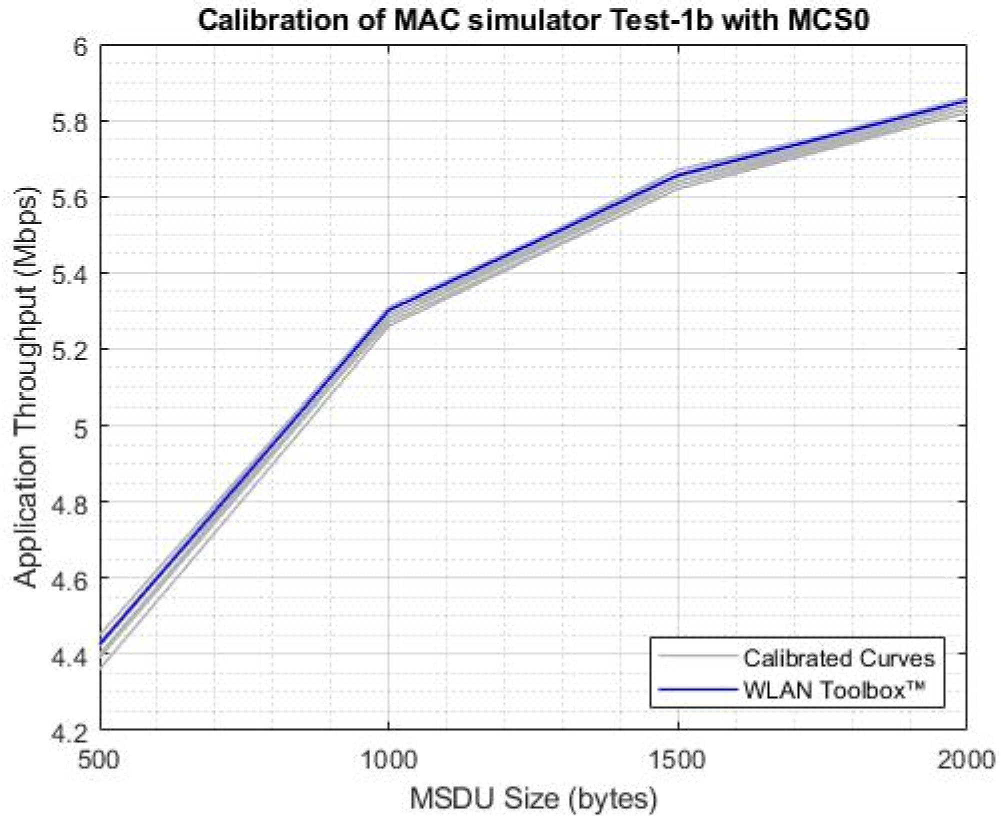
```
% Load statistics.mat (Output of the simulation) file
simulationResults = load('statistics', 'statisticsTable');
% Statistics
stats = simulationResults.statisticsTable;

% Successfully transmitted MAC layer bytes in the network
totalMACTxBytes = sum(stats.MACTxBytes);

% UDP & LLC overheads (bytes)
udpOverhead = 36;
llcOverhead = 8;

% UDP & LLC overhead (bytes) in the network
udpAndLLCOverhead = sum(stats.MACTxSuccess)*(udpOverhead + llcOverhead);

% Successfully transmitted application bytes
```

```
totalAppTxBytes = totalMACTxBytes - udpAndLLCOverhead;

% Time at which last transmission is completed in the network (Microseconds)
simulationTime = max(stats.MACRecentFrameStatusTimestamp);

% Application throughput (Mbps)
applicationThroughput = (totalAppTxBytes*8)/simulationTime;
disp(['Application Throughput = ' num2str(applicationThroughput) ' Mbps']);
```

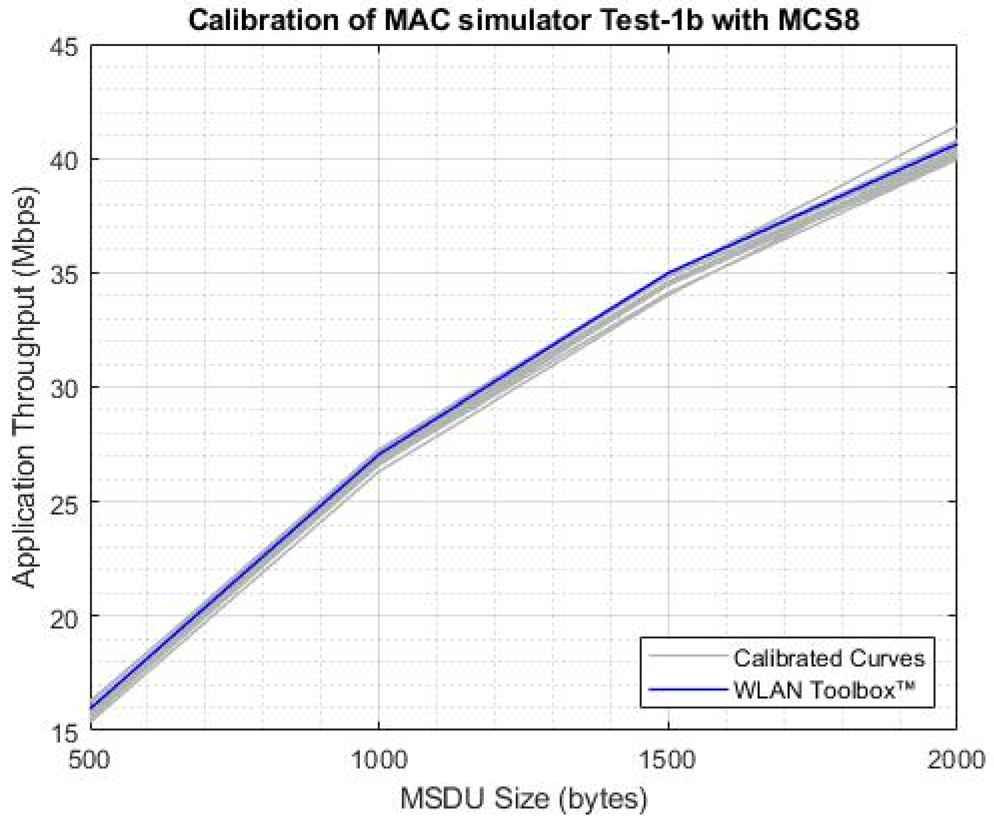
```
Application Throughput = 4.7276 Mbps
```

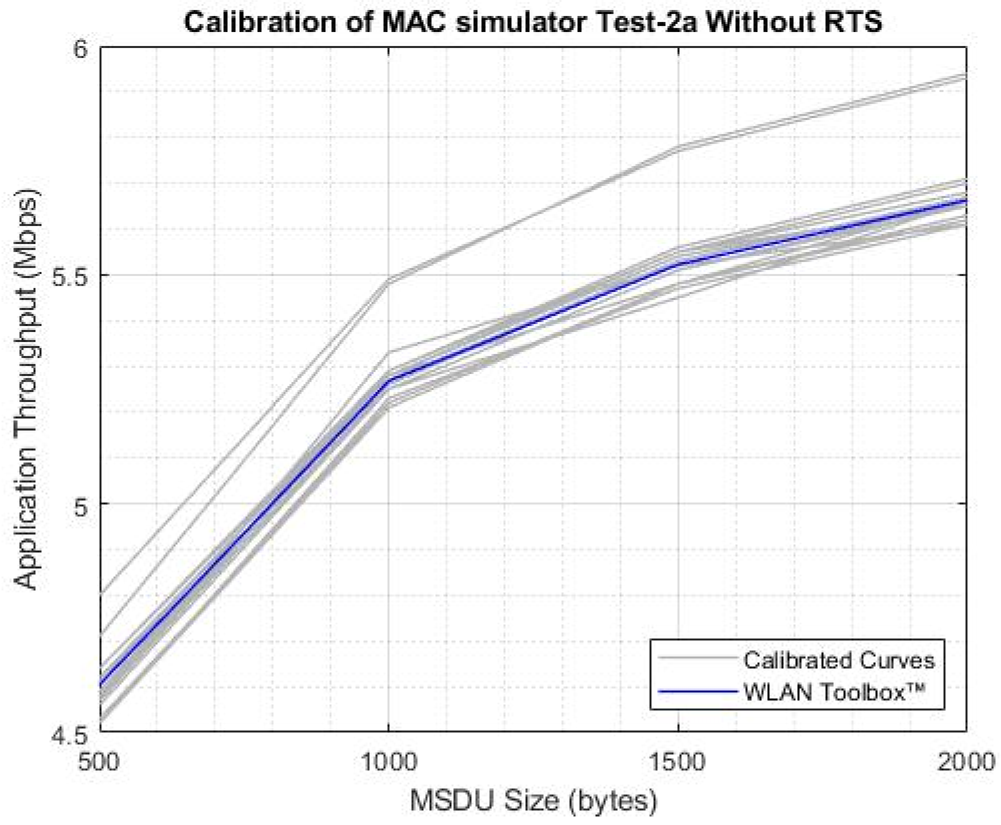
The application throughput for different TGax calibration scenarios is plotted against different MAC service data unit (MSDU) sizes for a simulation time of 30 seconds as shown below:

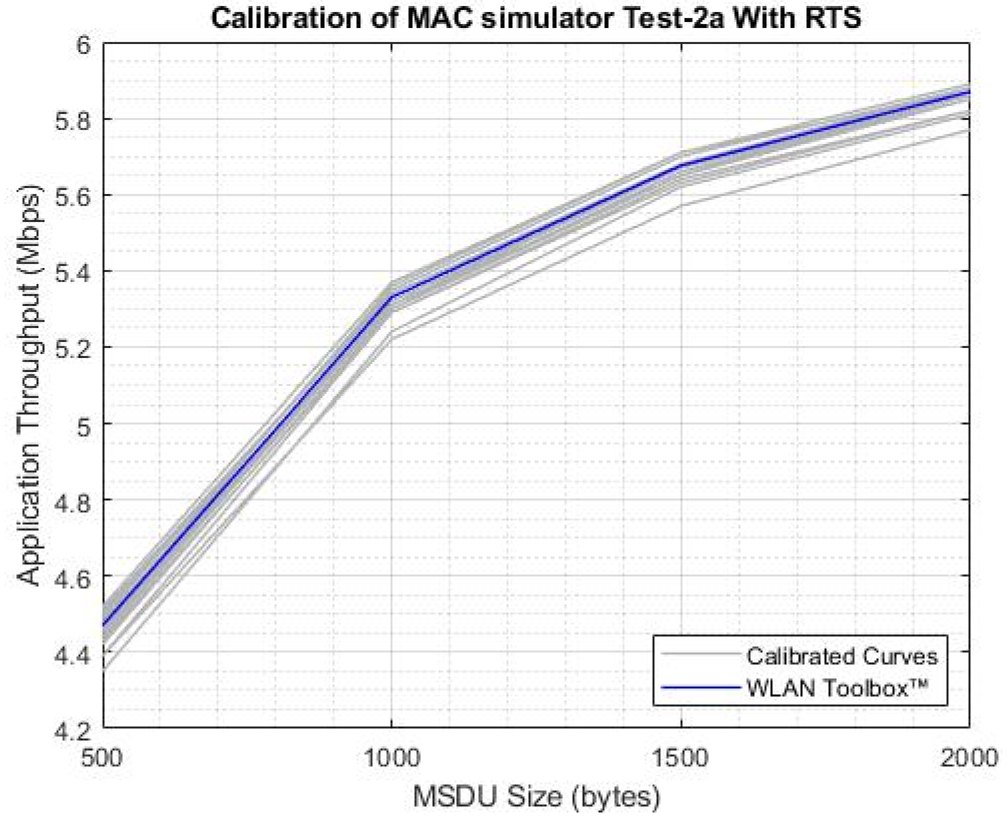




Calibration of MAC simulator Test-1b with MCS0
Application Throughput (Mbps)
6
5.8
5.6
5.4
5.2
5
4.8
4.6
4.4
4.2
500
1000
1500
2000
MSDU Size (bytes)
Calibrated Curves
WLAN Toolbox™


Calibration of MAC simulator Test-1b with MCS8
Application Throughput (Mbps)
MSDU Size (bytes)
45
40
35
30
25
20
15
500
1000
1500
2000
Calibrated Curves
WLAN Toolbox™


Calibration of MAC simulator Test-2a Without RTS
Application Throughput (Mbps)
MSDU Size (bytes)
Calibrated Curves
WLAN Toolbox™
6
5.5
5
4.5
500
1000
1500
2000




**Further Exploration**

**Configuration options**

You can change these configuration parameters to further explore this example:

- Application layer: Access category and packet interval
- MAC layer: RTS threshold, Tx queue size, data rate, short retry limit, long retry limit, transmitting frame format, MPDU aggregation, ack policy, number of transmit chains and the rate adaptation algorithms
- PHY: PHY Tx gain, PHY Rx gain, and Rx noise figure
- Channel modeling: Rayleigh fading, free space pathloss, range propagation loss and packet receive range
- Node positions using node position allocator
- The state of each node can be visualized during the run-time through the configuration available in the Visualizer block
- By default, the PHY transmitter and the receiver blocks run in the `Interpreted execution` mode. For longer simulation time, configure all the blocks to `Code generation` mode for better performance.

**Related examples**

Refer these examples for further exploration:

- To simulate the MAC Quality of Service (QoS) traffic scheduling in 802.11a/n/ac/ax networks, refer "802.11 MAC QoS Traffic Scheduling" (WLAN Toolbox) example.
- To model a multi-node IEEE 802.11ax network with abstracted PHY, refer "802.11ax System-Level Simulation with Physical Layer Abstraction" (WLAN Toolbox) example.
- To model a multi-node network using Distributed Coordination Function (DCF) MAC and 802.11a PHY, refer "Multi-Node 802.11a Network Modeling with PHY and MAC" (WLAN Toolbox) example.

This example enables you to create and configure a multi-node 802.11 network using a Simulink model for analyzing the MAC and application layer throughput. In this model, the MAC throughput obtained through the simulation results is used to calculate the application layer throughput. This model is validated using the Box 3 scenarios (Tests 1a, 1b, and 2a) specified in TGax evaluation methodology [ 3 ] to confirm that it complies with the IEEE 802.11 [ 1 ]. This example concludes that the calculated application layer throughput is within the range of minimum and maximum throughput specified in published calibration results [ 4 ].

### Appendix

The helper functions and objects used in this example are:

1. edcaFrameFormats.m: Create an enumeration for PHY frame formats.
2. edcaNodeInfo.m: Return MAC address of a node.
3. edcaPlotQueueLengths.m: Plot MAC queue lengths in the simulation.
4. edcaPlotStats.m: Plot MAC state transitions with respect to simulation times.
5. edcaStats.m: Create an enumeration for simulation statistics.
6. edcaUpdateStats.m: Update statistics of the simulation.
7. helperAggregateMPDUs.m: Generate an A-MPDU, by creating and appending the MPDUs containing the MSDUs in the MSDULIST.
8. helperSubframeBoundaries.m: Return subframes information of an A-MPDU.
9. phyRx.m: Model PHY operations related to packet reception.
10. phyTx.m: Model PHY operations related to packet transmission.
11. edcaApplyFading.m: Apply Rayleigh fading effect on the waveform.
12. heSIGBUserFieldDecode.m: Decode HE-SIG-B user field.
13. heCPECorrection.m: Estimate and correct common phase error.
14. heSIGBCommonFieldDecode.m: Decode HE-SIG-B common field.
15. heSIGBMergeSubchannels.m: Merge 20MHz HE-SIG-B subchannels.
16. addMUPadding.m: Add multiuser PSDU padding.
17. macQueueManagement.m: Create a WLAN MAC queue management object.
18. roundRobinScheduler.m: Create a round-robin scheduler object.
19. calculateSubframesCount.m: Return number of subframes to be aggregated.
20. interpretVHTSIGABitsFailCheck.m: Interprets the bits in VHT-SIG-A field
21. rateAdaptationARF.m: Create an auto rate fallback (ARF) algorithm object.
22. rateAdaptationMinstrelNonHT.m: Create a minstrel algorithm object.

### References


1. IEEE Std 802.11™. "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications." IEEE Standard for Information technology-Telecommunications and information exchange between systems, Local and metropolitan area networks-Specific requirements.
2. IEEE P802.11ax™/D4.1. "Amendment 6: Enhancements for High Efficiency WLAN.." Draft Standard for Information technology - Telecommunications and information exchange between systems Local and metropolitan area networks - Specific requirements -Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.
3. IEEE 802.11-14/0571r12. "11ax Evaluation Methodology." IEEE P802.11P: Wireless LANs.
4. Baron, Stephane., Nezou, Patrice., Guignard, Romain., and Viger, Pascal. "MAC Calibration Results." Presentation at the IEEE P802.11 - Task Group AX, September 2015.

# 802.11ax System-Level Simulation with Physical Layer Abstraction

This example demonstrates how to model a multi-node IEEE® 802.11ax™ [ 1 ] network with abstracted physical layer (PHY) using SimEvents®, Stateflow®, and WLAN Toolbox™. A PHY abstraction model largely reduces the complexity and the duration of system-level simulations by replacing the actual physical layer computations. This makes it possible to evaluate systems consisting of large number of nodes, resulting in increased scalability. Abstracted PHY models signal-power, gain, delay, loss and interference on each packet without generating physical layer packets, as specified by the TGax Evaluation Methodology [ 3 ].

### Physical Layer Abstraction

This example shows how to model an 802.11ax network with abstracted PHY. The example presents a variation of the system model used in the example "802.11 MAC and Application Throughput Measurement" (WLAN Toolbox). In "802.11 MAC and Application Throughput Measurement" (WLAN Toolbox) example, full PHY processing is modeled where waveforms are generated and decoded at the physical layer. However, this example models an abstracted PHY where no waveforms are generated or decoded. Abstracting the physical layer reduces the time taken for simulation at the cost of fidelity. Fidelity refers to the degree of exactness with which the PHY is modeled in the simulation. Simulations that tolerate low fidelity at the physical layer can use the abstracted PHY model.

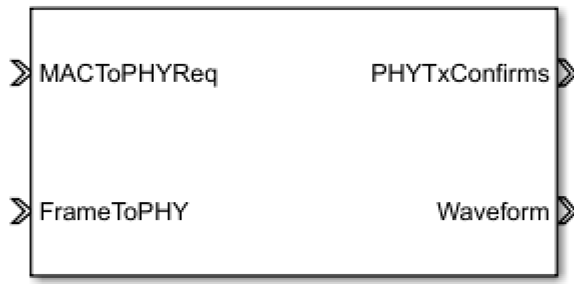
The abstracted PHY operates on pre-computed packet error rate (PER) tables and equations. These tables and equations are used to estimate the corrupted packet without any actual modulation or demodulation of packets, resulting in a low fidelity model. Refer the example "Physical Layer Abstraction for System-Level Simulation" (WLAN Toolbox) for more details related to the PHY abstraction.

### Abstracted Physical Layer Blocks

This section explains the blocks used for modeling the abstracted PHY and how it fits into the 802.11 [ 2 ] network model. Full PHY modeling involves operations related to waveform transmission and reception through a fading channel. Abstracted PHY models signal-power, gain, delay, loss and interference on each packet without generating physical layer packets. This example provides a `PHY Transmitter`, a `Statistical Channel`, and a `PHY Receiver` for modeling an abstracted PHY. These blocks are available in the library wlanAbstractedPHYLib.

**Abstracted PHY Transmitter:**

The `Abstracted PHY Transmitter` block models the transmit chain of the physical layer. This block consumes the frame and corresponding transmission parameters from the MAC layer. Parameters like transmit power, preamble duration, header duration and payload duration are calculated in the block. This information is passed along with the MAC frame as the metadata to simulate the transmission of a waveform.

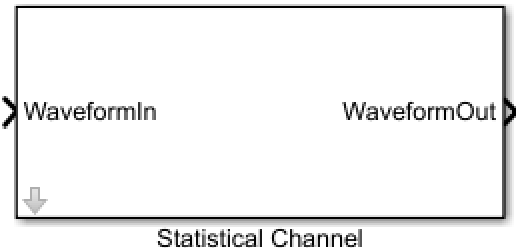



Interfaces to the `Abstracted PHY Transmitter` block are:

- MACToPHYReq: Triggers for indicating transmission start/end requests from MAC layer
- FrameToPHY: MAC frame to be transmitted
- PhyTxConfirms: Confirmation triggers to MAC layer for indicating completion of MAC layer requests
- Waveform: Abstract waveform transmitted into the channel (MAC frame and the metadata)

**Statistical Channel:**

The `Statistical Channel` block models pathloss, propagation delay, and reception range of the packet. To enable the estimation of loss, delay, and range at each receiver, the `Statistical Channel` block must be modeled inside every node coupled with the `Abstracted PHY Receiver`. Propagation delay is applied on each received packet, and the signal strength of each packet is degraded with optional pathloss. If the receiving node is within the range, the packet is forwarded to the `Abstracted PHY Receiver` with the effective signal strength. The packet is dropped if the receiving node is outside the range of the transmitter.




Interfaces to the `Statistical Channel` are:

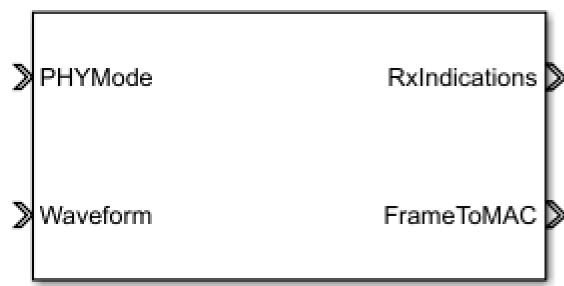
- WaveformIn: Input packet received from a PHY transmitter
- WaveformOut: Output packet intended for PHY receivers after applying channel loss

**Abstracted PHY Receiver:**

The `Abstracted PHY Receiver` block models the receive chain of the physical layer. This block receives and processes the packet based on the received metadata. The `Abstracted PHY Receiver` block models interference based on the packets received at overlapping timelines. The received packets are processed only at these checkpoints: (a) End of the preamble duration (b) End of

each subframe duration in the payload for aggregated frames (or) end of the payload duration for non-aggregated frames.

This block also provides an option for configuring the level of abstraction through the `PHY Abstraction` mask parameter. You can configure it to `'TGax Evaluation Methodology Appendix 1'` [ 3 ] to predict the performance of a link with a TGax channel model using effective SINR mapping. Details of this procedure can be found in the example “Physical Layer Abstraction for System-Level Simulation” (WLAN Toolbox). Alternatively, you can configure it to `'TGax Simulation Scenarios MAC Calibration'` [ 4 ] to assume a packet failure on interference, without actually calculating the link performance. Note that the option `'TGax Evaluation Methodology Appendix 1'` works for only MCS values in the range [0-9], as the TGax Evaluation Methodology [ 3 ] is defined only for these values.

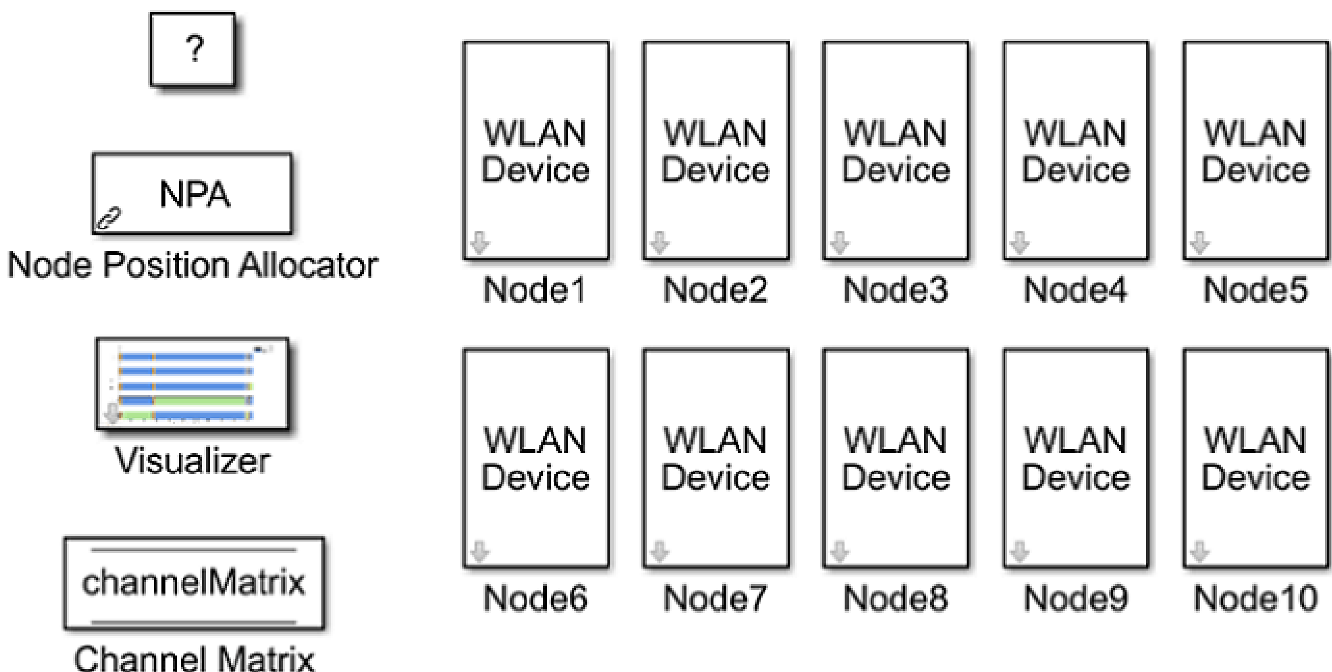



Interfaces to the `Abstracted PHY Receiver` block are:

- PHYMode: Trigger for switching off the receiver function when transmission is in progress
- Waveform: Abstract waveform received from the channel (MAC frame and the metadata)
- RxIndications: Triggers to MAC for indicating channel state shift (busy/idle) events or receive (start/end) events
- FrameToMAC: Received MAC frame

#### System-Level Simulation

This example simulates a network with 10 nodes in the model, WLANMultiNodeAbstractedPHYModel, as shown in this figure. These nodes implement carrier-sense multiple access with collision avoidance (CSMA/CA) with physical carrier sense and virtual carrier sense. The physical carrier sensing uses the clear channel assessment (CCA) mechanism to determine whether the medium is busy before transmitting. The virtual carrier sensing uses the RTS/CTS handshake to prevent the hidden node problem.

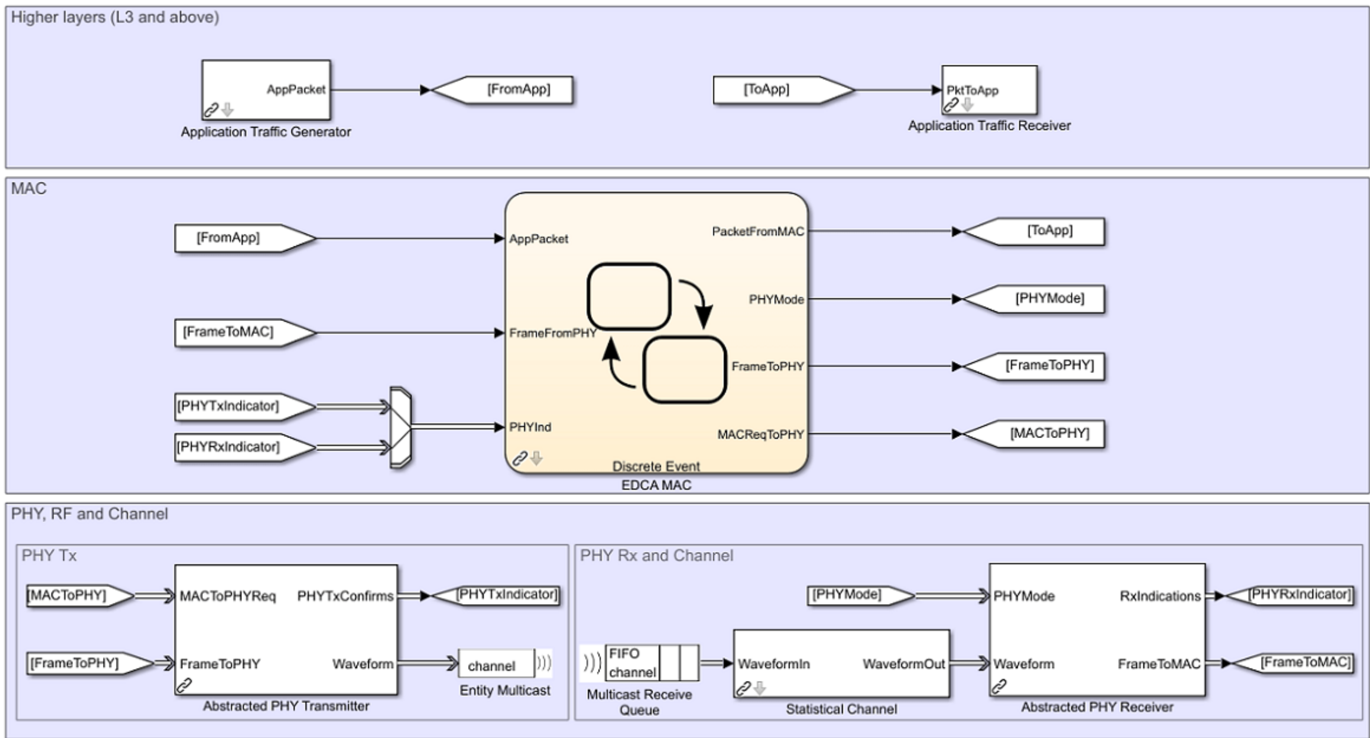



The positions for all the nodes in the network are configured through the node position allocator (`NPA`) block in the model. The state of each node can be visualized during run-time through the configuration available in the `Visualizer` block. The `Channel Matrix` block is a Data Store Memory. On initialization, a TGax channel realization is generated between each pair of nodes in the network and the resulting channel matrix per subcarrier is stored in the block. During the simulation, each receiver node accesses the memory to obtain the channel matrix between itself and a transmitting node to determine the link quality. In this model, nodes 1, 2, 3, 6, 7, and 8 act as both the transmitters and receivers, while nodes 4, 5, 9, and 10 are just passive receivers.

**Node Subsystem**

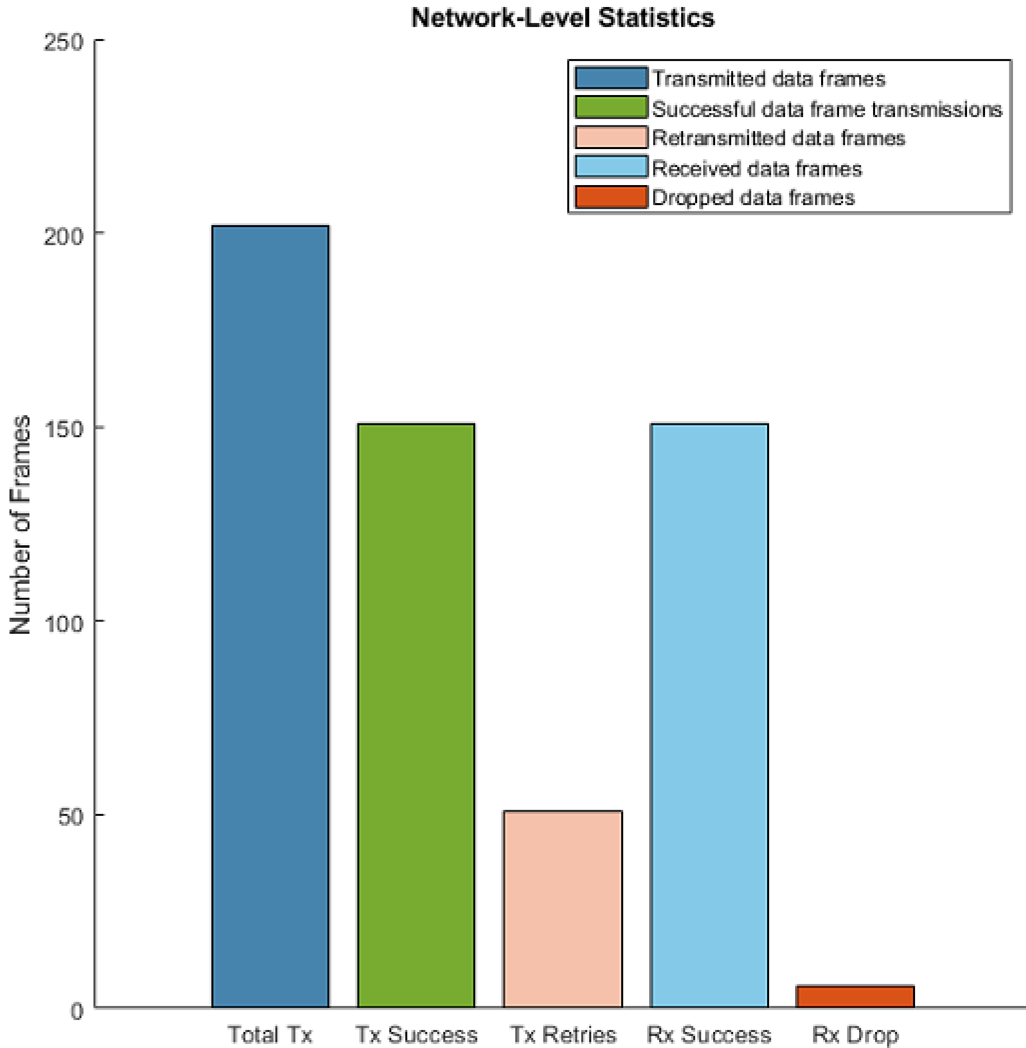
Each node in the above model is a subsystem representing a WLAN device. Each node contains an application layer, a MAC layer and a physical layer. The physical layer is modeled using the abstracted PHY blocks described in the previous section. You can configure a node to transmit and receive packets on a specific channel (frequency) by changing the `Multicast tag` parameter of the `Entity Multicast` and the `Multicast Receive Queue` blocks. By default, all nodes operate on the same channel. You can also configure the receive range for a specific node using the `Packet Receive Range` parameter of the `Statistical Channel` block.

You can easily switch between abstracted PHY blocks available in the wlanAbstractedPHYLib and full PHY processing blocks available in the `wlanFullPHYLib.slx` library of the example “802.11 MAC and Application Throughput Measurement” (WLAN Toolbox). The interfaces to the transmitter, receiver and channel blocks remain the same. By default, the abstracted PHY blocks run in the `Interpreted execution` mode. For longer simulation time, configure all the blocks to `Code generation` mode for better performance.




### Simulation results

Running the model simulates the WLAN network for the specified simulation time. A plot with network-level statistics (corresponding to MAC layer) is generated at the end of simulation. Detailed node-level statistics (corresponding to application, MAC, and physical layers) are collected during the simulation and saved to a base workspace file `statistics.mat`. You can also enable an optional live visualization, to see the state of each node during run-time, through the mask configuration of the `Visualizer` block.

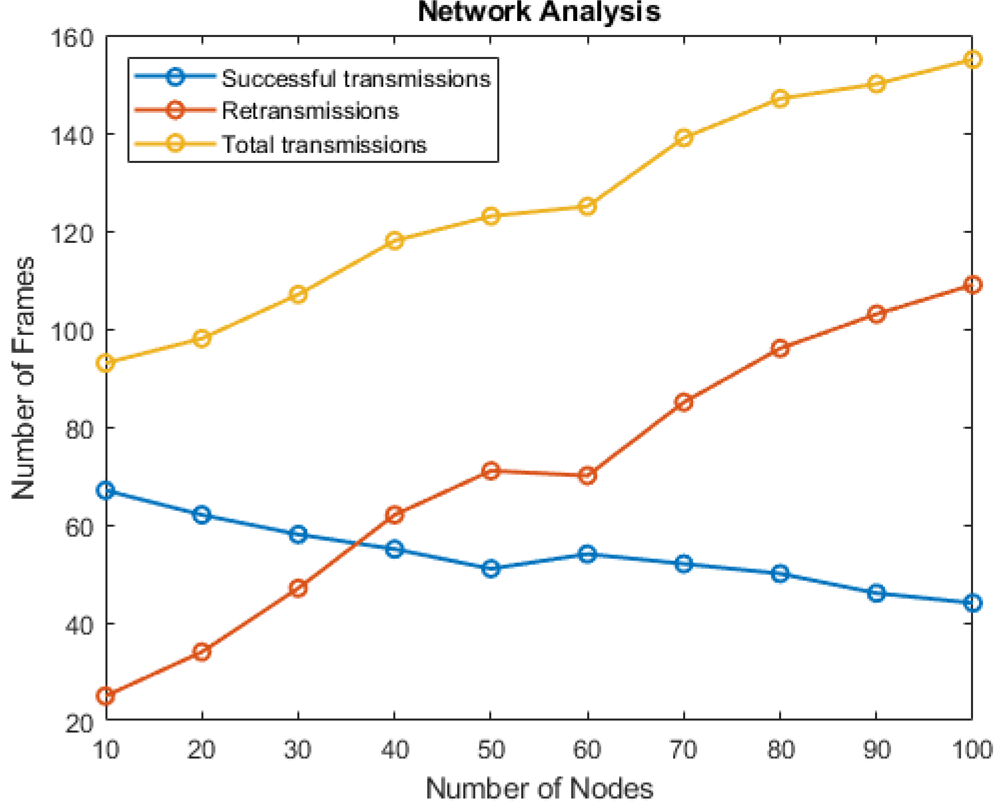



#### Scalability

The above model shows a network of 10 nodes. You can create a network with a large number of nodes by using the hCreateWLANNetworkModel function. This helper function uses the node subsystem from this example and creates a network of WLAN nodes positioned linearly 10 meters apart from each other. You can create different simulation scenarios and analyze the node-level or network-level statistics with varying number of nodes. For example, the plot below shows the retransmissions and successful transmissions relative to the total transmissions, as the number of nodes in the network increase. The configuration parameters used for collecting the results are:

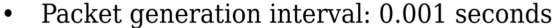
- Format: HE-SU
- Modulation and coding scheme (MCS) index: 0
- Number of subframes in A-MPDU: 1

- Distance between nodes: 10 meters
- Path loss: Not applied
- PHY abstraction type: "TGax Evaluation Methodology Appendix 1"
- Range propagation: All the nodes are within range of each other
- Operating frequency: All the nodes operate in the same frequency

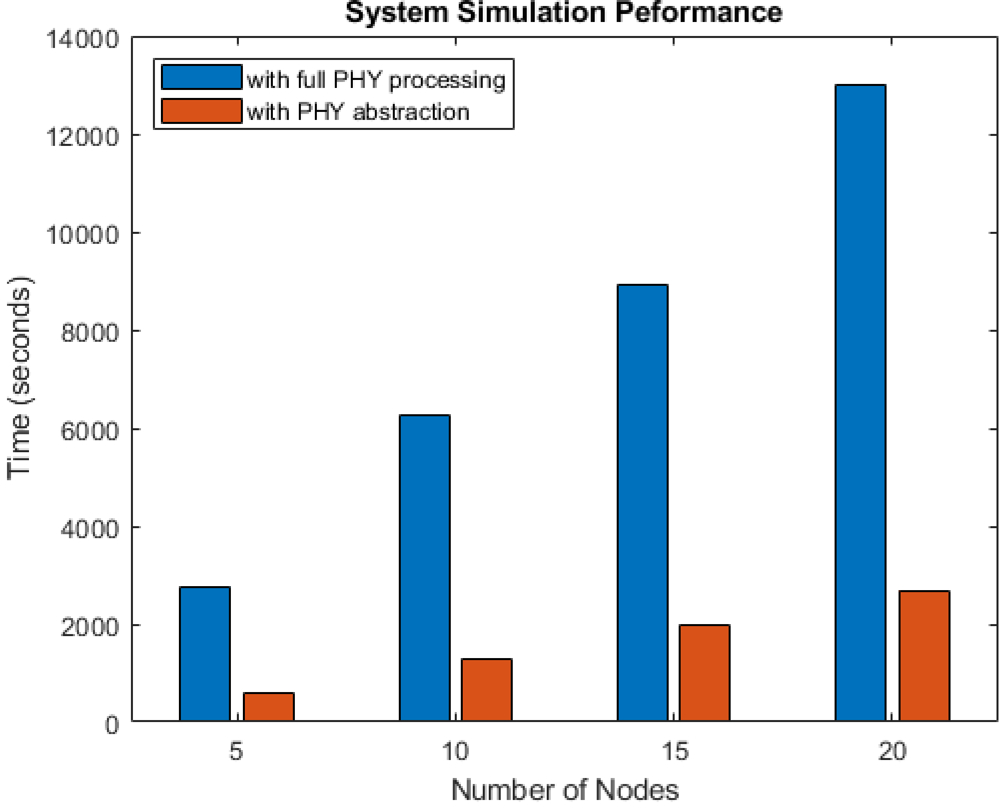



The plot below shows that the simulation runs faster with abstracted PHY as compared to full PHY processing, thus making it more scalable. The configuration parameters used for collecting the performance results are:

- Format: HE-SU
- Modulation and coding scheme (MCS) index: 0
- Number of subframes in A-MPDU: 2
- Distance between nodes: 1 meter
- Path loss: Not applied
- PHY abstraction type: "TGax Evaluation Methodology Appendix 1"
- Range propagation: All nodes are within range of each other
- Operating frequency: All the nodes operate in the same frequency
- Simulation mode: `Code generation` mode for all the blocks
- Simulation time: 5 seconds



- Packet generation interval: 0.001 seconds




This example explained the physical layer abstraction and demonstrated a 10-node WLAN network with abstracted PHY. This example shows that a network simulation with abstracted PHY is faster and more scalable compared to using full PHY processing.

### Further Exploration

In this example, the A-MPDUs exchanged between the nodes are deaggregated to MPDUs at the receiving node. These MPDUs are exported to packet capture (PCAP) and packet capture next generation (PCAPNG) format file using the `pcapDump` DES block. To use the `pcapDump` DES block, go to wlanSystemLevelComponentsLib

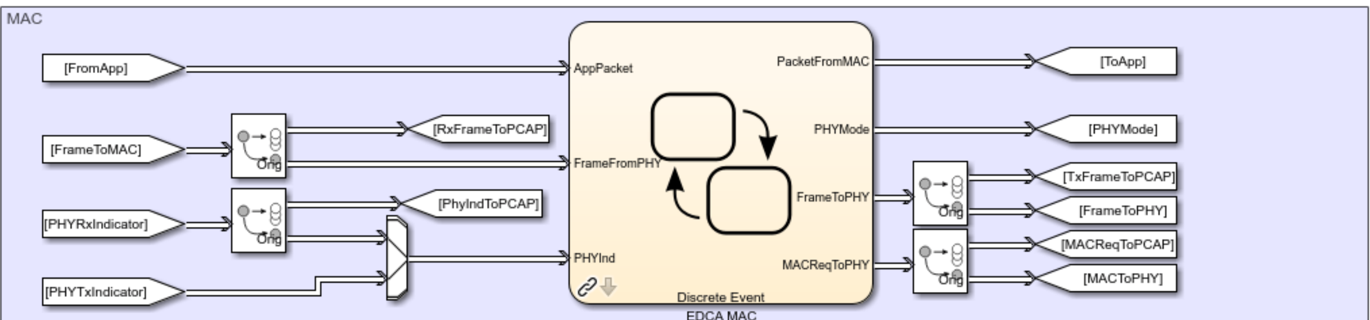
### Export to PCAP/PCAPNG Format File

The PCAP/PCAPNG format files contain the packet data of the network. These files are mainly associated with network analyzers like Wireshark [ 5 ], a third party tool used to visualize and analyze PCAP/PCAPNG files. The main advantages of using PCAP/PCAPNG files during system level simulations are:

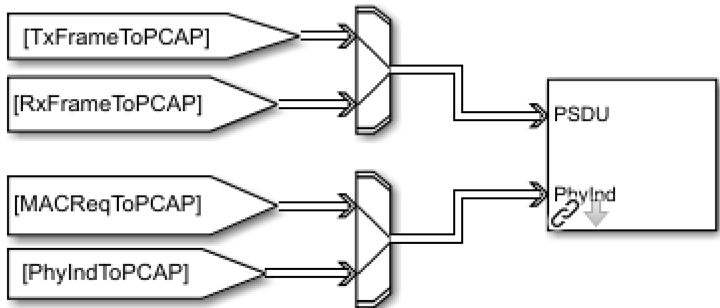
- Monitor the network traffic.
- Visualize and analyze the network characteristics of the data.

To duplicate the MAC layer input entities (received A-MPDUs, `FrameToMAC`, and `PhyRxIndicator` vector) and output entities (transmitted A-MPDUs, `FrameToPHY`, and `MACReqToPHY` vector), use the

Entity Replicator blocks. The MAC layer provides RxFrameToPCAP, PhyIndToPCAP, TxFrameToPCAP, and MACReqToPCAP as inputs to the pcapDump DES block.

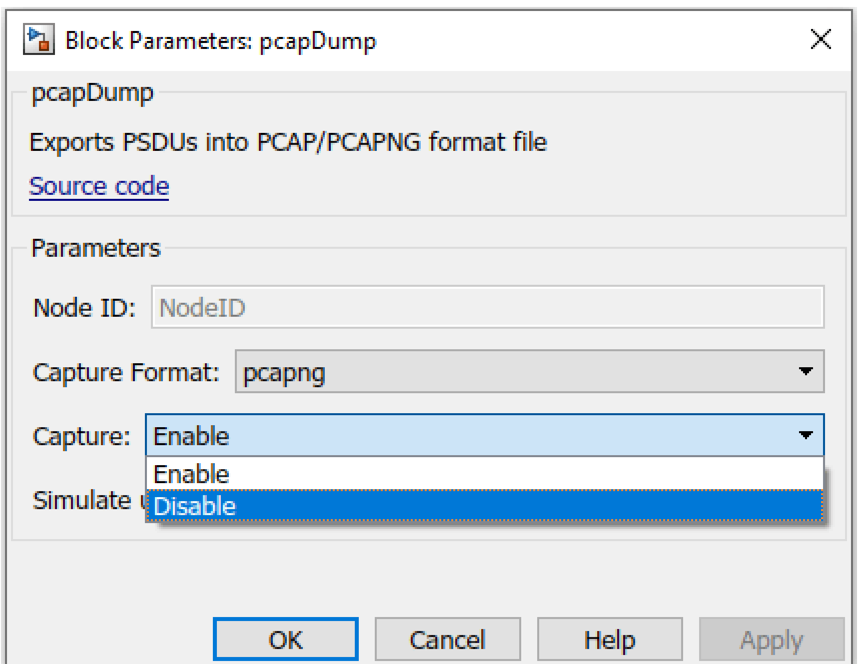



The pcapDump DES block contains two input ports, one for Tx/Rx A-MPDUs and other for Tx/Rx information.




Select the capture format as pcap or pcapng. As the simulation starts, the packets exchanged between the nodes are logged into the selected capture format file.

To capture the packet, double click the pcapDump DES block and select the parameter Capture as Enable.




A new capture file (PCAP/PCAPNG format) is created for every node. The file name corresponds to name of the node. If name of the node is Node1, the captured file name is Node1.pcap or Node1.pcapng.

**Appendix**

The example uses these helpers:

1. edcaFrameFormats.m: Create an enumeration for PHY frame formats.
2. edcaNodeInfo.m: Return MAC address of a node.
3. edcaPlotQueueLengths.m: Plot MAC queue lengths in the simulation.
4. edcaPlotStats.m: Plot MAC state transitions with respect to simulation times.
5. edcaStats.m: Create an enumeration for simulation statistics.
6. edcaUpdateStats.m: Update statistics of the simulation.
7. helperSubframeBoundaries.m : Return subframe boundaries of an A-MPDU. * phyTxAbstracted: Model PHY operations related to packet transmission * phyRxAbstracted: Model PHY operations related to packet reception * channelBlock: Model the channel for a node
8. addMUPadding.m: Add or remove the padding difference between an HE-SU and HE-MU PSDU
9. macQueueManagement.m: Create a WLAN MAC queue management object
10. roundRobinScheduler.m: Create round-robin scheduler object
11. calculateSubframesCount.m: Calculate the number of subframes required to form MU-PSDU * hCreateWLANNetworkModel: Create a WLAN network with given number of nodes * hDisplayNetworkStats: Display network level statistics * hSetupAbstractChannel: TGax channel setup * HelperPCAPNGWriter: Create a PCAPNG file writer handle object * HelperPCAPWriter: Create a PCAP file writer handle object * HelperWLANPacketWriter: Create a file writer handle object that writes WLAN packets into PCAP/PCAPNG format file * HelperPCAPUtils: Provide methods that are commonly used in PCAP helpers * createRadiotapHeader: Create a radiotap header

12. rateAdaptationARF.m: Create an auto rate fallback (ARF) algorithm object.
13. rateAdaptationMinstrelNonHT.m: Create a minstrel algorithm object.

### References

1. IEEE P802.11ax™/D4.1 Draft Standard for Information technology - Telecommunications and information exchange between systems Local and metropolitan area networks - Specific requirements -Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications - Amendment 6: Enhancements for High Efficiency WLAN.
2. IEEE Std 802.11™ - 2016 IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.
3. IEEE 802.11-14/0571r12 - 11ax Evaluation Methodology.
4. IEEE 802.11-14/0980r16 - TGax Simulation Scenarios.
5. Wireshark - Go Deep. https://www.wireshark.org/. Accessed 9 Dec. 2019.

# Use SimEvents with Simulink


- “Working with SimEvents and Simulink” on page 7-2
- “Solvers for Discrete-Event Systems” on page 7-5
- “Model Simple Order Fulfilment Using Autonomous Robots” on page 7-7

# Working with SimEvents and Simulink

You can exchange data between SimEvents and Simulink environments. However, time-based signals and SimEvents signals have different characteristics.

## Exchange Data Between SimEvents and Simulink

Use Simulink Function blocks in SimEvents models:

- To read or write attributes of entities.
- To send messages that trigger other events.
- To exchange data between event and time domain sections of a model.

Use Message Send and Receive blocks to send and receive messages between Simulink and SimEvents blocks.

## Time-Based Signals and SimEvents Block Transitions

Time-based signals and SimEvents signals have different characteristics. Here are some indications that a time-based signal is automatically converted into a SimEvents signal, or conversely:

- You want to connect a time-based signal to an input port of a SimEvents block.
- You are using data from a SimEvents block to affect time-based dynamics.
- You want to perform a computation involving both time-based signals and SimEvents output.

When the transition occurs, a capital **E** appears on the line.

## SimEvents Support for Simulink Subsystems

You can use SimEvents blocks (discrete-event blocks) without restriction in Simulink Virtual Subsystems, and in Simulink Nonvirtual Subsystems, observing some specific guidelines.

For more information about Simulink subsystems, see Subsystem, Atomic Subsystem, Nonvirtual Subsystem, CodeReuse Subsystem.

### Discrete-Event Blocks in Virtual Subsystems

You can use discrete-event blocks without restriction in a virtual subsystem.

### Discrete-Event Blocks in Nonvirtual Subsystems

When you use discrete-event blocks in an atomic subsystem, follow these guidelines:

- The entire discrete-event subsystem, which includes all discrete-event blocks, must reside entirely within the atomic subsystem. You cannot route entities into, or out of, the atomic subsystem.
- If you want to connect two or more atomic subsystems that contain discrete-event blocks, each atomic subsystem must meet all the preceding conditions.

For more information about atomic subsystems, see Subsystem, Atomic Subsystem, Nonvirtual Subsystem, CodeReuse Subsystem.

### Discrete-Event Blocks in Variant Subsystems

You can use discrete-event blocks in a variant subsystem. The software permits both entities and time-based signals to enter and depart a virtual variant.

However, if you use an atomic subsystem as a variant, or within a variant, then that atomic subsystem must obey the rules for using discrete-event blocks in nonvirtual subsystems. These rules are described in “Discrete-Event Blocks in Nonvirtual Subsystems” on page 7-2. An atomic subsystem is the only type of nonvirtual subsystem that can contain discrete-event blocks, even when the nonvirtual subsystem is contained within a variant subsystem.

The SimEvents software does not support setting the **Variant activation time** parameter to `code compile` for these blocks:

- Variant Subsystem
- Variant Sink
- Variant Source

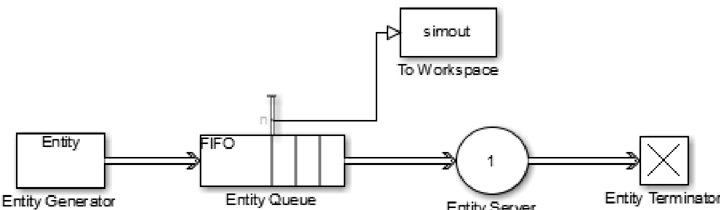
## Save Simulation Data

### Behavior of the To Workspace Block

The To Workspace block writes event-based signals to the MATLAB workspace when the simulation stops or pauses. One-way to pause a running simulation is to select **Pause** under the **Debug** tab.

### Send Queue Length to the Workspace

The example shows one way to write the times and values of signals to the MATLAB workspace. In this case, the signal is the **n** output from an Entity Queue block, which indicates how many entities the queue holds.




You can use different time formats in the To Workspace block to display the data.

To record entities and their attributes passing along an entity line, consider connecting a To Workspace block to that entity line.

### Data Logging

You can log data from your SimEvents model using Simulink. For more information, see “Save Run-Time Data from Simulation”.

## See Also

Message Receive | Message Send | Simulink Function

## Related Examples

- “Create a Hybrid Model with Time-Based and Event-Based Components”
- “Events and Event Actions” on page 1-2
- “Generate Entities When Events Occur” on page 1-13

## More About

- “Solvers for Discrete-Event Systems” on page 7-5

# Solvers for Discrete-Event Systems

| In this section... |
| --- |
| "Variable-Step Solvers for Discrete-Event Systems" on page 7-5 |
| "Fixed-Step Solvers for Discrete-Event Systems" on page 7-5 |

Depending on your configuration, you can use both variable-step and fixed-step solvers with discrete-event systems. To choose solver settings for your model, navigate to the **Solver** pane of the model Configuration Parameters dialog box.

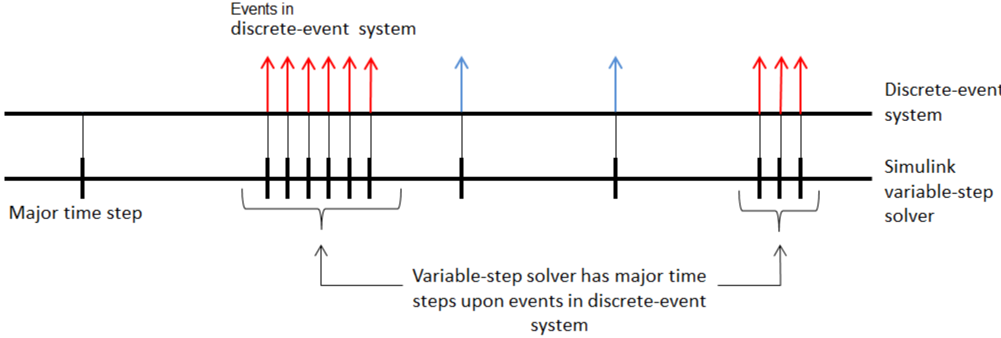
When choosing a solver type for your model, use the following guidelines:

- If your model contains only event-based computation and excludes continuous and discrete time-based computation, choose the variable-step, discrete solver. In this case, if you select a variable-step continuous solver, the software detects that your model does not contain any blocks with continuous states (Simulink blocks) and automatically switches the solver to `discrete (no continuous states)`. When the software makes this change, it notifies you with a message in the MATLAB command window.
- If your discrete-event system is within a Simulink model that also contains time-based modeling, choose either a variable-step or fixed-step solver, depending on your simulation requirements. For each solver type, the following sections describe the behavior of discrete-event systems when contained within such models.

## Variable-Step Solvers for Discrete-Event Systems

If your discrete-event system is within a Simulink model that contains time-based modeling, and you choose a variable-step solver for the model, the Simulink solver has a major time step each time the discrete-event system processes events.

The following graphic illustrates the behavior of the variable-step solver when used with a discrete-event system contained within a Simulink model.

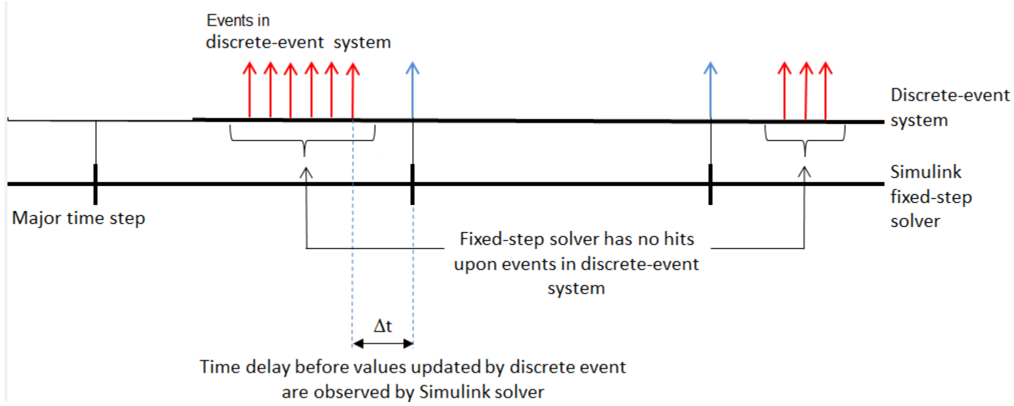



## Fixed-Step Solvers for Discrete-Event Systems

If you have a discrete-event system within a Simulink model that includes time-based modeling, you can choose a fixed-step solver for the model.

When you use a fixed-step solver, the simulation still executes events in the discrete-event system at the times at which they occur. However, these events do not cause the Simulink solver to have sample hits at those times. The software insulates the discrete-event system from the time-based portions of the Simulink model.

The following graphic illustrates the behavior of the fixed-step solver when used with a discrete-event system.




## See Also

## More About

- “Compare Solvers”
- “Working with SimEvents and Simulink” on page 7-2

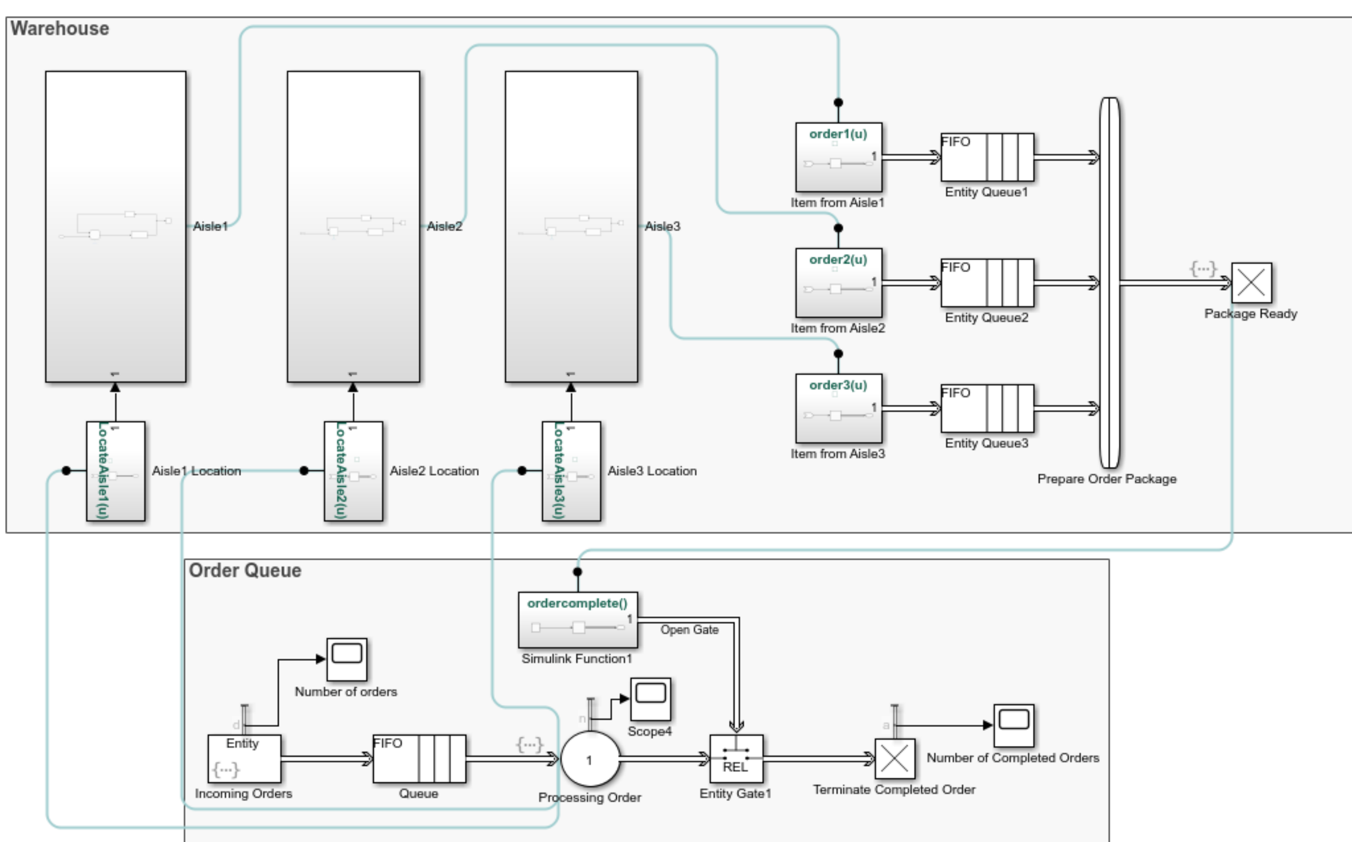
# Model Simple Order Fulfilment Using Autonomous Robots

This example models a warehouse with autonomous robots for order management. The goal of the example is to show how to facilitate complex models created with Simulink, Stateflow, and SimEvents components and their communication via messages. See “View Differences Between Stateflow Messages, Events, and Data” (Stateflow) for more information about messages.

## Order Fulfilment Model

Order fulfilment model has two major components

- The Order Queue component represents an online order queue with the blocks from the SimEvents® library.
- The Warehouse component represents delivery of order items by autonomous robots. It uses blocks from Simulink® and SimEvents® libraries and a Stateflow® chart. The chart requires a Stateflow® license.

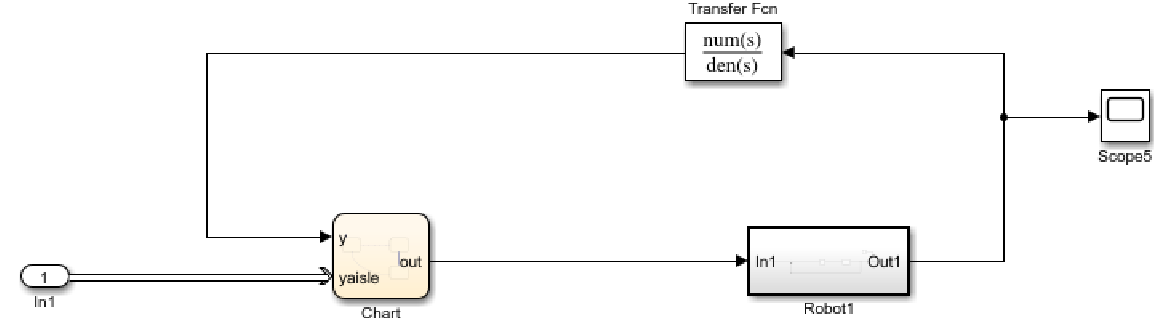



In this model, an online order for multiple items arrives at the Order Queue component. The locations of the ordered items are communicated from the Processing Order block to the autonomous robots in the Warehouse component. Three robots are assigned to three aisles. A robot picks up an item from its aisle location and returns it to its initial location for delivery. An order can have one, two, or three

items. When all ordered items are delivered by the robots, the order is complete and a new order arrives. Until an order is complete, no new orders are received to the Order Queue component.

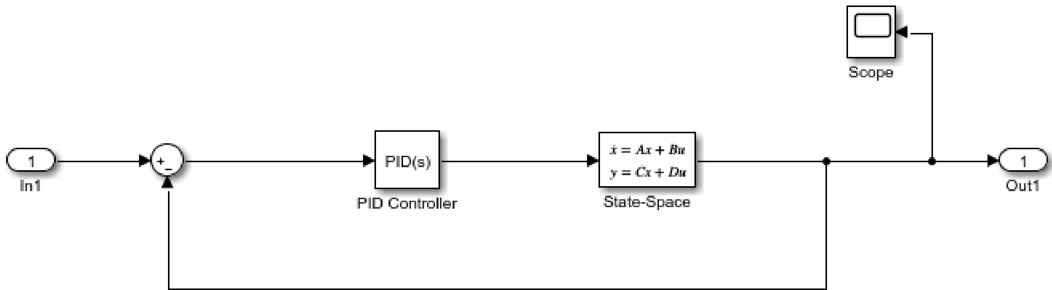
## Warehouse Component

The warehouse has three aisles. The first aisle contains clothing items, the second aisle contains toys, and the third aisle contains electronics. Three delivery robots are identical and their dynamics are driven by a linear time-invariant system that is controlled by a tuned PID controller. For instance, the Aisle1 subsystem block consists of a Robot1 subsystem and a Discrete-Event Chart block as a scheduler.



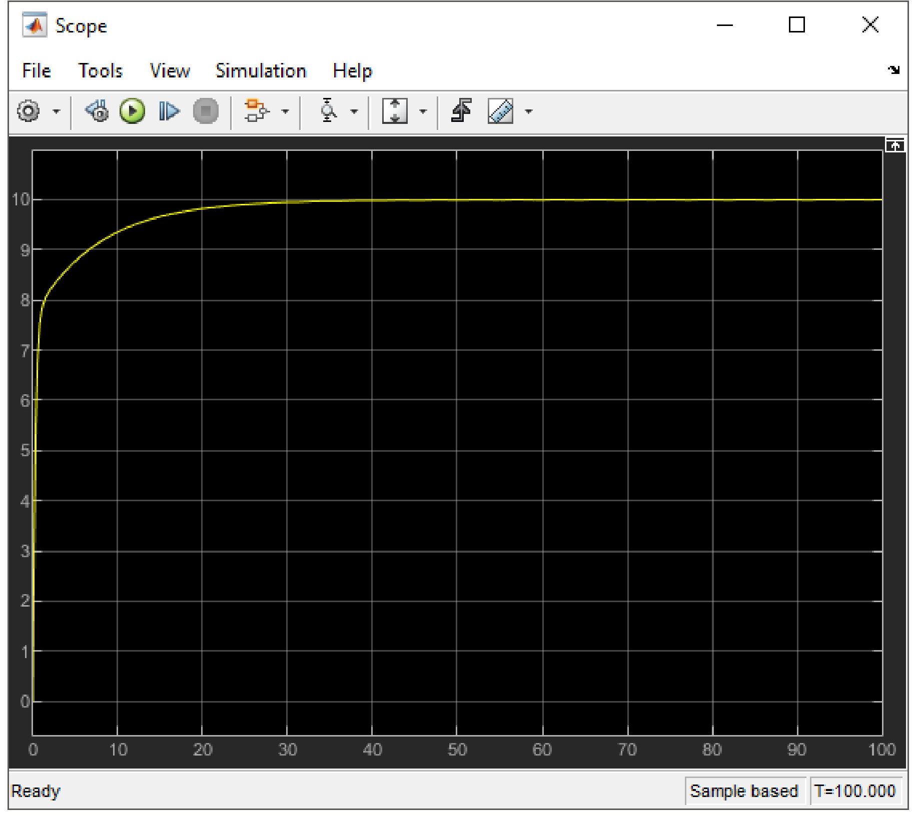

### Robot1 Subsystem

The Robot1 subsystem has a generic feedback control loop with the dynamics of the robot represented by the State-Space block and the PID controller.




The Robot1 subsystem is designed to track a reference signal from the In1 block, which is the `out` signal from the Discrete-Event Chart block. The system compares the input value with the output from the State-Space block and the difference between signals is fed to the PID Controller block.

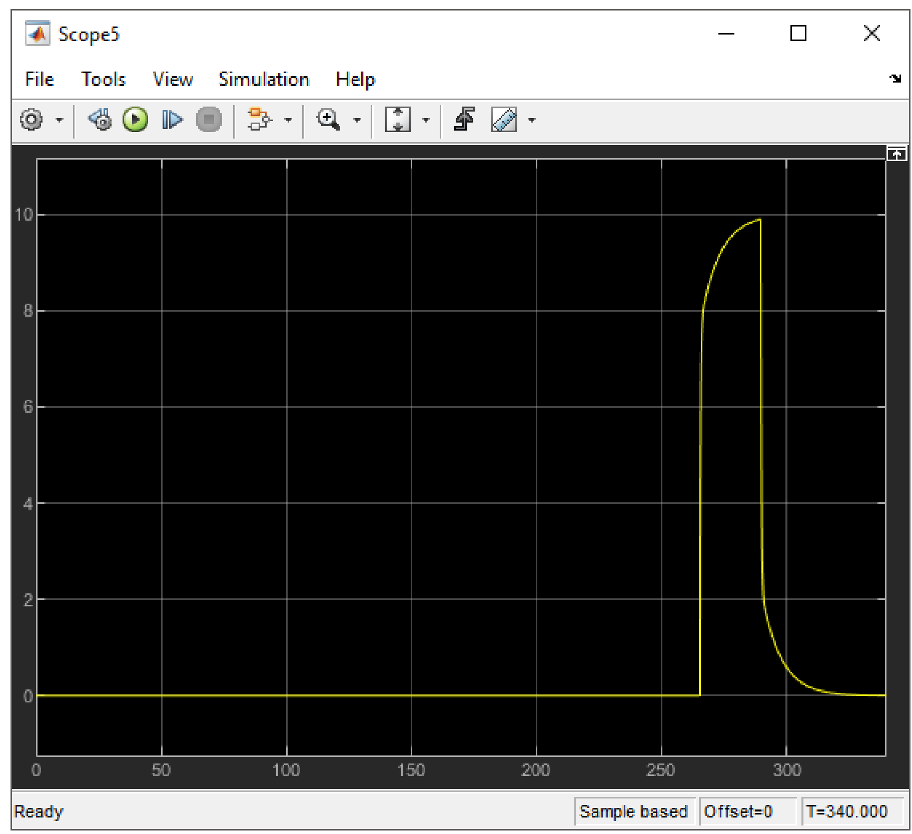
For instance, if the signal from the In1 block is a constant with value `10`, starting from the initial state `0`, the output of the system converges to `10`.




In the *x*-axis and *y*-axis, Robot1 moves as follows.

- Robot1 is initially at *x1* and *y1* = 0 coordinate. For item pickup and delivery, it moves only on the *y*-axis and its *x1* coordinate remains the same.
- Each order item in Aisle1 has a *yaisle* coordinate on the *y*-axis. *yaisle* becomes the constant input reference signal to be tracked by Robot1 subsystem.
- When Robot1 subsystem reaches *yaisle*, it picks up the order item and autonomously reruns back to *y1* = 0 location for delivery.

The scope displays an example trajectory for Robot1 subsystem, which receives a *yaisle* value `10` as the constant reference input at simulation time `265`. When the distance between the robot's location and $y = 10$ is `0.1`, reference input signal is `0` and the robot returns to its initial location for delivery.

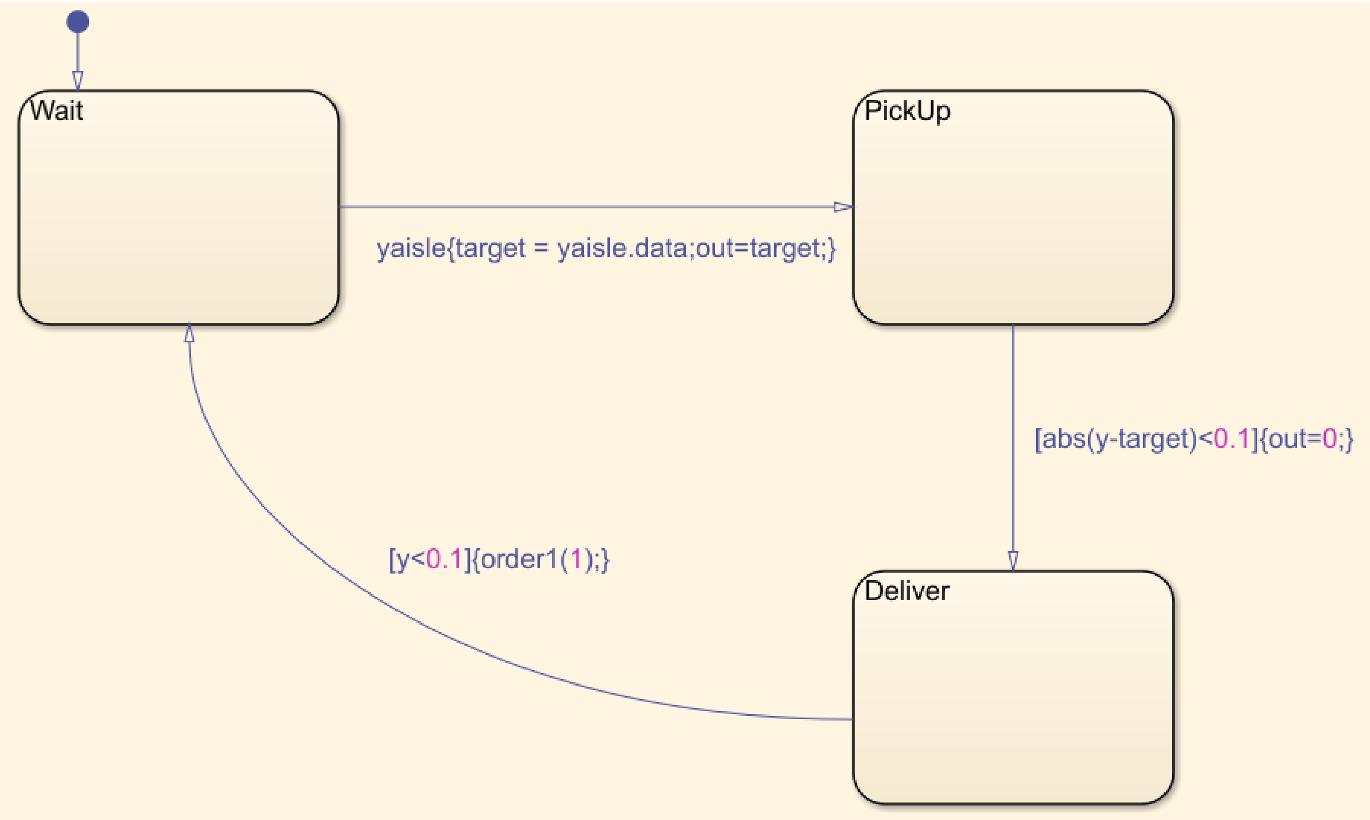



Robot2 subsystem and Robot3 subsystem have identical dynamics and behavior for the item delivery in Aisle2 subsystem and Aisle3 subsystem. Their *x* coordinates are *x2* and *x3* and they also move on the vertical *y*-axis.

### Scheduler

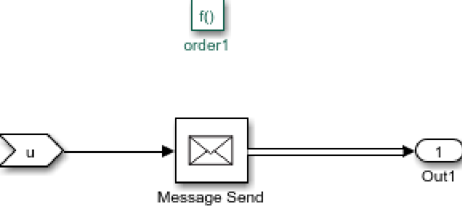
In the previous example trajectory, Robot1 has three states. The Discrete-Event Chart block is used to schedule the transitions between these robot states.

- A robot waits in the `Wait` state, until it receives a *yaisle* item coordinate. Robot1 subsystem is in the `Wait` state, until the simulation time is `265`.
- A robot transitions to the `PickUp` state, when there is an incoming message carrying the *yaisle* value of an item to the Discrete-Event Chart block. This value is assigned to `out`, which is the output signal from the Discrete-Event Chart block. The `out` signal is fed to the Robot1 subsystem as the input signal In1 to be tracked and the robot moves towards the *yaisle* item location. Robot1 subsystem transitions to the `PickUp` state at time `265`.
- When a robot is `0.1` units away from *yaisle*, it picks up the item. Then, the robot transitions to a `Deliver` state. The `out` signal becomes `0` and the robot returns back to $y = 0$ for delivery. At the simulation time `290`, Robot1 subsystem is `0.1` unit away from $y = 10$ and transitions to the `Deliver` state.
- When a robot returns and it is `0.1` units away from $y = 0$, it transitions to the `Wait` state. At around `320`, Robot1 subsystem delivers the item and transitions back to the `Wait` state.




**Order Package Preparation**

1. When a robot delivers its item, the item is sent to generate the order package. This behavior is represented by the Message Send block that generates a message inside the Item from Aisle Simulink Function block. Then, the generated message enters the Entity Queue block.




2. A Composite Entity Creator block waits for all three items from the three Entity Queue blocks to create a composite entity that represents the order.

   To complete the order, all of the items from the three aisles are required to be delivered.
3. When all the items are delivered, the order is complete and it arrives at the Package Ready block.
4. The entry of the order to the Package Ready block triggers the Simulink Function1 block to generate a message and to open the gate for order termination.
5. When the order is terminated, a new order arrives at the Processing Order block which restarts the delivery process.

Until an order is complete, no new orders are received, so the robots that deliver their items wait for the order to be completed.

## Order Queue Component

The order queue block is a simple queuing system composed of an Entity Generator, Entity Queue, Entity Server, Entity gate, and Entity Terminator block. For more information about creating a simple queuing system, see “Manage Entities Using Event Actions”.

1. Entity Generator block randomly generates orders. The intergeneration time is drawn from an exponential distribution with mean `100`.
2. Each generated entity has three randomly generated attributes `aisle1`, `aisle2`, and `aisle3` that represent the *yaisle* coordinates of the items in Aisle1, Aisle2, and Aisle3 subsystems.

   ```
   entity.Aisle1 = randi([1,30]);
   entity.Aisle2 = randi([1,30]);
   entity.Aisle3 = randi([1,30]);
   ```

   It is assumed that the items are located vertically between $y = 1$ and $y = 30$.
3. The arrival of the order to the Entity Server block activates the robots by communicating the items' *yaisle* coordinates. Entering this MATLAB code in the **Entry action** field.

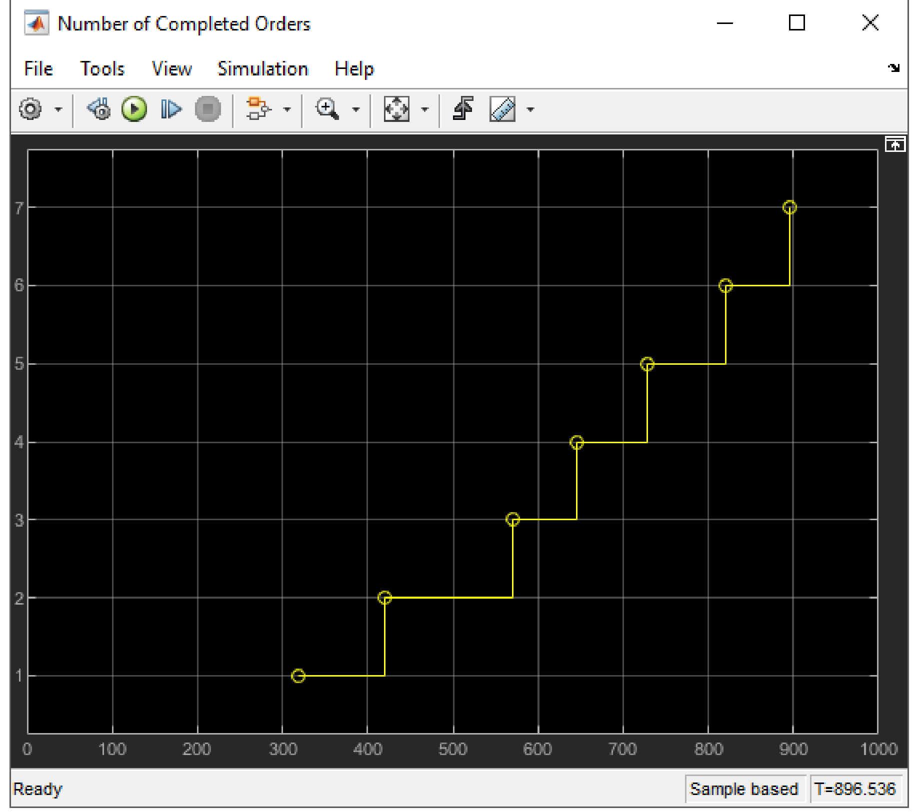
   ```
   LocateAisle1(entity.Aisle1);
   LocateAisle2(entity.Aisle2);
   LocateAisle3(entity.Aisle3);
   ```

   Calling the `LocateIsle()` function communicates the *yaisle* coordinate of an item to the corresponding robot.
4. The order waits in the Entity Server block until the Entity Gate block opens.
5. When all items are delivered, the order package enters the Package Ready block and its entry calls the Simulink Function1 block through the function `ordercomplete()`. The Simulink Function1 block generates a message to open the gate.
6. When the gate opens, the order is terminated and a new order arrives at the Entity Server block.

## Results

Inspect the order throughput from the Order Queue.

1. Increase the simulation time to `1000`.
2. Simulate the model and observe that the scope displays `7` as the total number of completed orders.




## See Also

Discrete-Event Chart | Entity Generator | Entity Queue | Entity Server | Entity Terminator

## Related Examples

- "Create a Hybrid Model with Time-Based and Event-Based Components"

## More About

- "Working with SimEvents and Simulink" on page 7-2
- "Solvers for Discrete-Event Systems" on page 7-5

# Build Discrete-Event Systems Using Charts


- “Create Custom Queuing Systems Using Discrete-Event Stateflow Charts” on page 8-2
- “Discrete-Event Chart Precise Timing” on page 8-6
- “Trigger a Discrete-Event Chart Block on Message Arrival” on page 8-9
- “Dynamic Scheduling of Discrete-Event Chart Block” on page 8-18

# Create Custom Queuing Systems Using Discrete-Event Stateflow Charts

The Discrete-Event Chart block is similar to a Stateflow chart but is used for discrete events. You can use the block to receive, process, and send SimEvents entities. The Discrete-Event Chart block provides graphical state transitions and MATLAB action language to create custom SimEvents models.

The distinguishing characteristic of the Discrete-Event Chart block is that it executes in an event-based rather than time-based fashion. To model custom discrete-event systems, use these Discrete-Event Chart block behaviors:

- Precise timing — The time resolution for occurrence of events can be arbitrarily precise and is not limited by the sample time of the model.

  For more information, see "Discrete-Event Chart Precise Timing" on page 8-6.
- Trigger on arrival — The block executes immediately on message arrival. It does not wait for the next sample time hit.

  For more information, see "Trigger a Discrete-Event Chart Block on Message Arrival" on page 8-9.
- Variable execution order — The block does not have a fixed sorted execution order. The order of execution depends on the run-time conditions of the model.

  For more information, see "Dynamic Scheduling of Discrete-Event Chart Block" on page 8-18.
- Multiple executions per time step — The block can execute zero or multiple times in a single time step.

  For more information, see "Dynamic Scheduling of Discrete-Event Chart Block" on page 8-18.

---

**Note** With SimEvents software, you can view, edit, and simulate your Discrete Event Chart custom block within a SimEvents example model. However, to save the model you must have a Stateflow license.

For new models, without a Stateflow license, you can view and edit the model, but cannot simulate or save it.

The entities you use with discrete-event charts can be bus objects or anonymous entities.

---

## Properties of Discrete-Event Chart

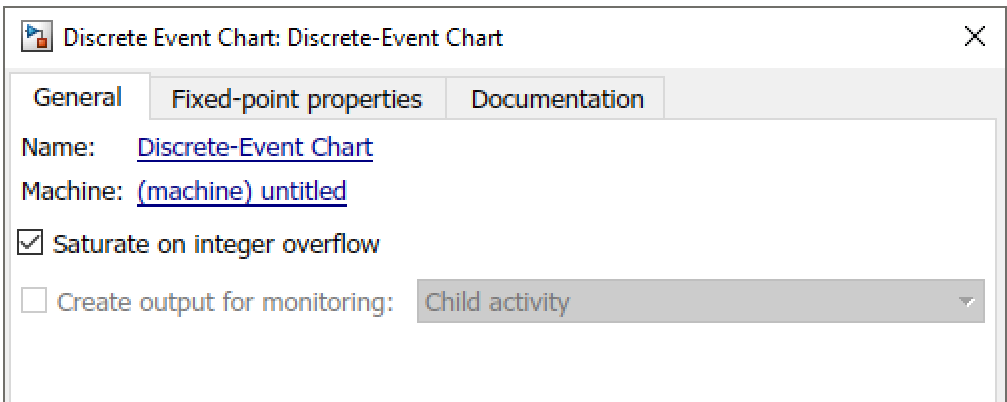
Discrete-event chart properties allow you to specify how your chart interfaces with Simulink and SimEvents. These properties are a subset of the Stateflow chart properties.

To specify properties for a single chart:

1. Double-click a chart.

2. Right-click an open area of the chart and select **Properties**.

   All charts provide general and documentation properties.
3. Observe that the chart allows the configuration of only these properties on the **General** tab. It also supports the **Fixed-point properties** and **Documentation** tabs.


   For more information about chart properties, see “Specify Properties for Stateflow Charts” (Stateflow).

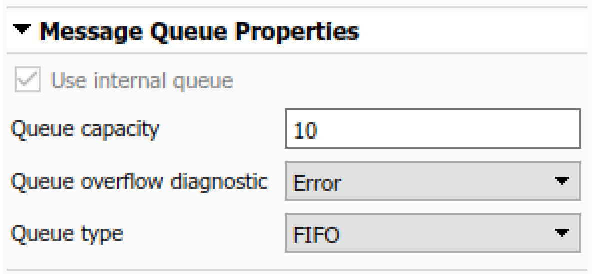
**Note** SimEvents supports only MATLAB action language and always supports variable-size arrays.

## Define Local Messages

Similar to the Stateflow chart, you can define local messages for the discrete-event chart using the Stateflow Editor or Model Explorer.

To add a local message for the discrete-event chart:

1. Select **Symbols Pane** and **Create Message**.
2. Select `Local Message` and rename it to EntityOut.
3. To specify local message queue properties, such as capacity, sorting policy, and overflow behavior right-click EntityLocal and select `Inspect` to open Property Inspector.

## Specify Message Properties

Discrete-event charts have additional properties for output messages and local messages.

| Message Input Port Properties | Description |
| --- | --- |
| **Priority** | If two message events occur at the same time, to decide which to process first, the discrete-event chart uses this priority. A smaller numeric value indicates a higher priority. |

## Event Triggering

An event in Stateflow is an object that trigger actions. For more information, see "Synchronize Model Components by Broadcasting Events" (Stateflow).

SimEvents Discrete-Event Chart support a subset of these events:

- Message
- Temporal
- Local
- Implicit Events, `enter`, `exit`, `on`, `change`

SimEvents Discrete-Event Chart does nor support these events:

- Conditions without an event
- `during`, `tick`
- Event input from Simulink
- Event output to Simulink

**Note** The SimEvents event calendar displays and prioritizes message, and temporal events. Events of these types execute according to the event calendar schedule.

The event calendar does not display or prioritize local and implicit events. In the SimEvents environment, these events execute as dependent events of message or temporal events. For parallel states, local and implicit events execute in the state execution order.

## Message Triggering

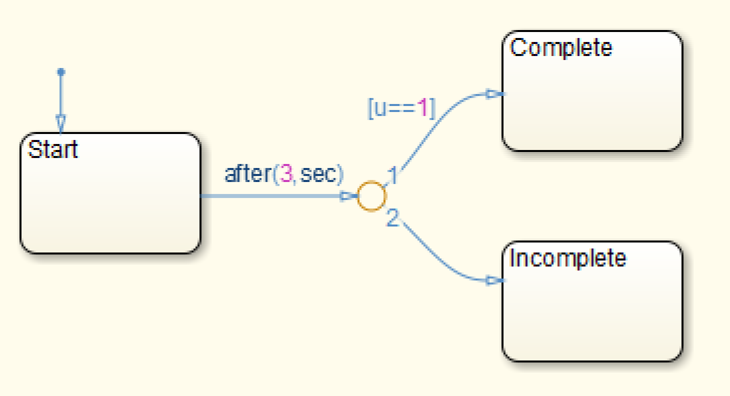
When a message or entity arrives at a message input or local queue, the discrete-event chart responds to the message as follows:

- If the discrete-event chart is in a state of waiting for a message, the discrete-event chart wakes up and makes possible transitions. The chart immediately wakes up in order of message priority, processing the message with the highest priority first. For an example, see "Trigger a Discrete-Event Chart Block on Message Arrival" on page 8-9.
- If the discrete-event chart does not need to respond to the arriving message, the discrete-event chart does not wake up and the message is queued.

## Temporal Triggering

In a discrete-event chart, you can use both event-based and absolute time-based temporal logic operators. When using absolute time-based temporal logic operators, the SimEvents software uses the specified time delay value exactly. For an example, see "Discrete-Event Chart Precise Timing" on page 8-6.

For example, the activation of the temporal logic `'after(3,sec)'` causes the chart to wake up after three seconds of simulation clock time.




When using absolute-time temporal logic operators, observe these differences from the Stateflow environment.

| Operator | Description |
| --- | --- |
| `after` | You can use as event notation in both state actions and transitions. |
| `before` | When you use as event notation of a transition, you cannot use additional condition notations on this transition. You can apply a connective junction to check additional conditions, as long as the connective junction has one unconditional transition. |

In conditional notation, Discrete-Event Chart supports both `after` and `before`.

## See Also

Discrete Event Chart

## Related Examples

- "Discrete-Event Chart Precise Timing" on page 8-6
- "Specify Properties for Stateflow Charts" (Stateflow)

# Discrete-Event Chart Precise Timing

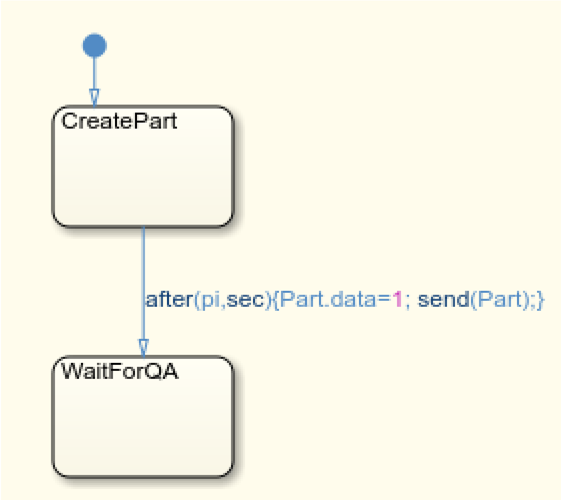
This example shows the precise timing that a Discrete-Event Chart block executes as it generates parts in a facility. The behavior of the Discrete-Event Chart and the Stateflow® blocks are compared. Both blocks require a Stateflow® license. Using a Discrete-Event Chart block, the example shows that the temporal resolution of events can be arbitrarily precise and independent from the model sample time.

In this example, an entity represents a part generated in `pi` seconds. The solver is set to `Fixed-step` with step size `1`, and for the Stateflow® Chart block, the `Enable Super Step Semantics` check box is selected. For more information, see “Super Step Semantics” (Stateflow).

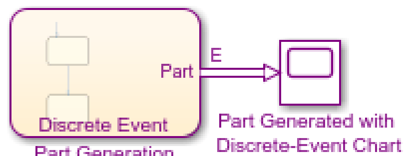
### Model Description

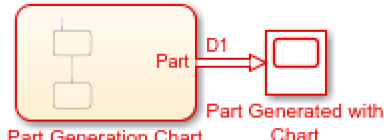
In this model, the Part Generation block is created using a Discrete-Event Chart block and the Part Generation Chart is created using a Stateflow® Chart block. Both blocks contain the same state transition model, including two states, `CreatePart` and `WaitForQA`.

- The `CreatePart` state represents the production of a `Part` in `pi` seconds.
- The `WaitForQA` state represents the wait for the quality control department for `Part`'s validation.




Enable the sample time annotation and simulate the model. Observe that the sample time for the Discrete-Event Chart block reflects the event-based sampling.

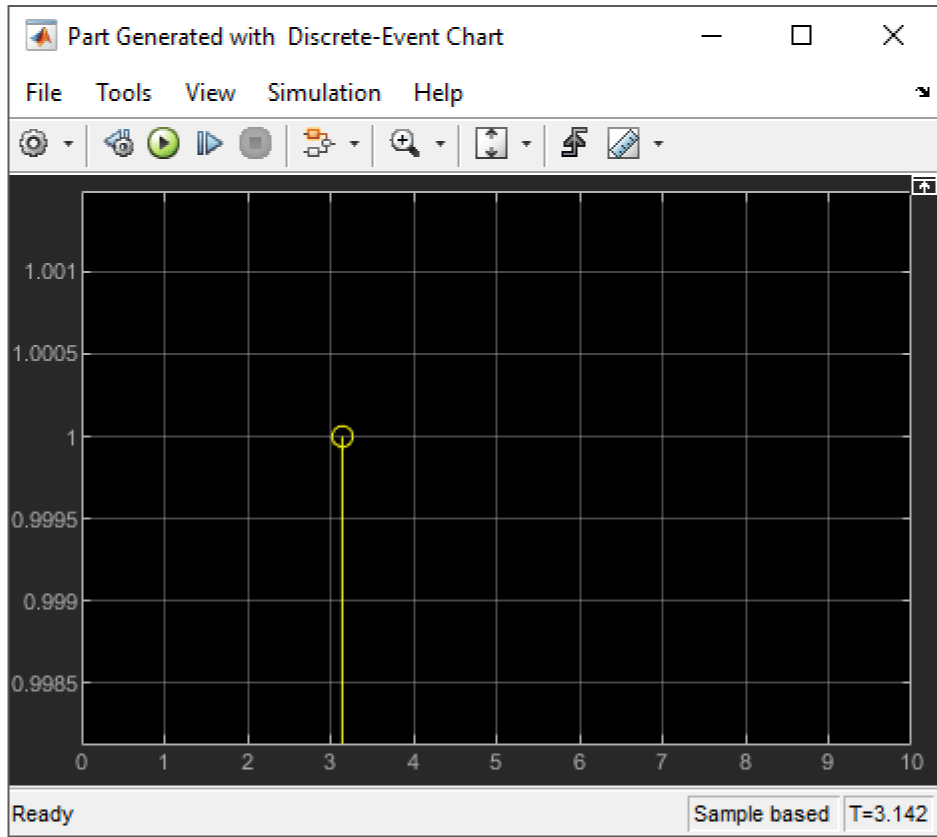


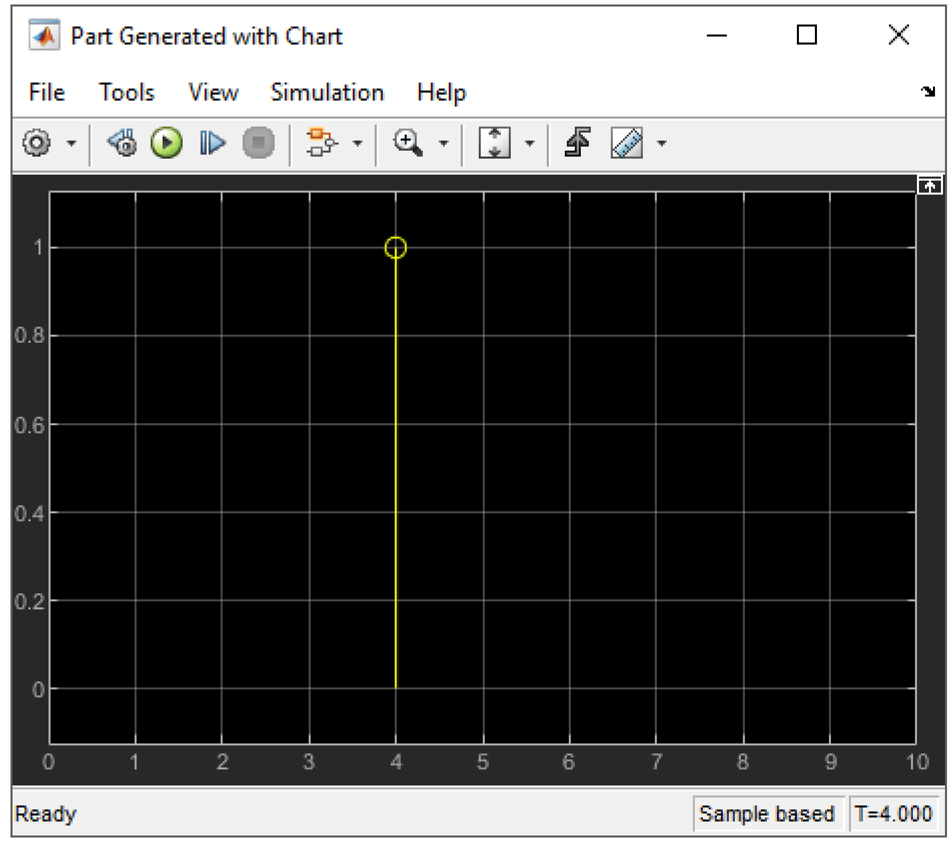
Copyright 2019 The MathWorks, Inc.

### Simulation Results

Observe that `Part` is generated by the Discrete-Event chart after precisely `3.14` seconds, independent from the simulation step size.




Observe that `Part` is generated by the Stateflow® Chart after `4` seconds. This is due to the fixed step size `1` , which causes the Stateflow® Chart block to wait until the next simulation step.




## See Also

Discrete-Event Chart

## More About

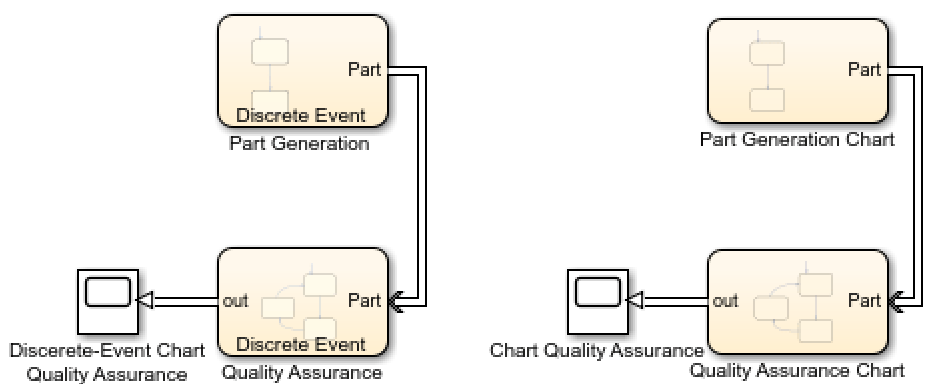

- "Create Custom Queuing Systems Using Discrete-Event Stateflow Charts" on page 8-2
- "Trigger a Discrete-Event Chart Block on Message Arrival" on page 8-9
- "Dynamic Scheduling of Discrete-Event Chart Block" on page 8-18

# Trigger a Discrete-Event Chart Block on Message Arrival

This example shows how to trigger a Discrete-Event Chart Block on the message arrival when generating parts in a facility and performing quality assurance. In the example, behaviors of a Discrete-Event Chart and Stateflow® Chart blocks are compared. Both blocks require a Stateflow® license. The example shows that, a Discrete-Event Chart block executes immediately upon the arrival of a message and does not wait for the next sample time hit.

In this example, a part is generated in the Part Generation block and it is sent to the Quality Assurance block for the Part's quality control. After the evaluation, the Quality Assurance block outputs the validated part.

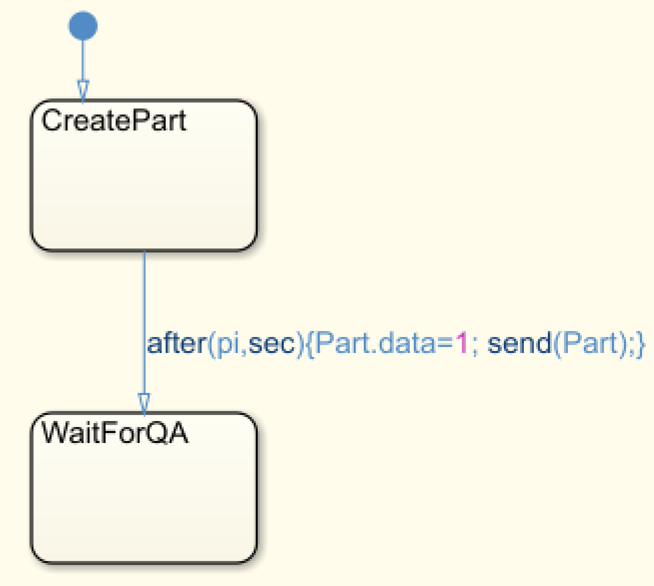
The model is further modified to send the validated part back to the Part Generation block from which it is shipped to the customer. For both models in this example, the solver is set to `Fixed-step` with step size `1`, and for all the Stateflow® Chart blocks, the `Enable Super Step Semantics` option is selected. For more information, see “Super Step Semantics” (Stateflow).




### Model Description

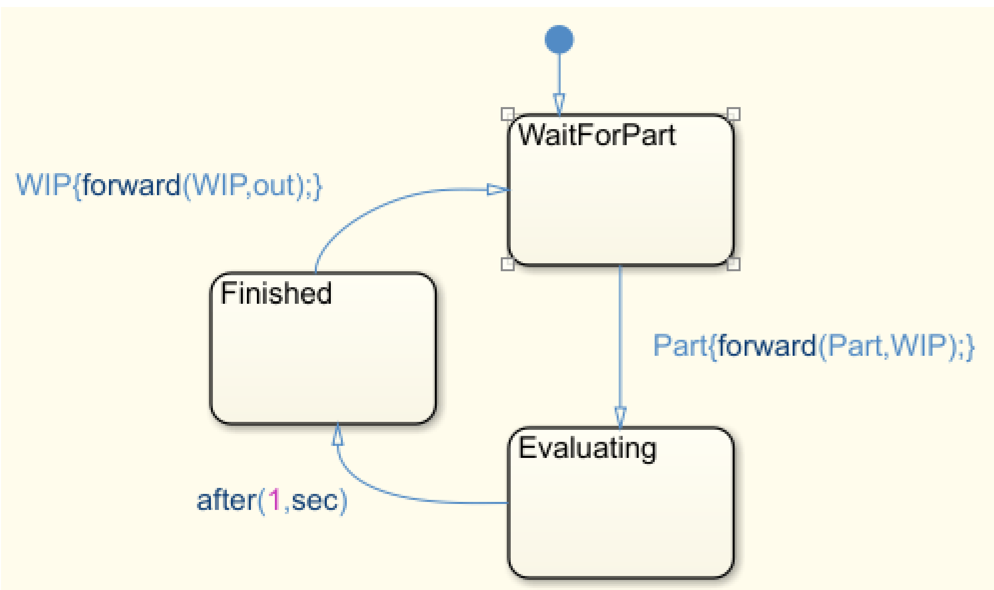
In the `PartQualityEvaluationModel` model, the Part Generation is modeled by a Discrete-Event Chart block, and the Part Generation Chart is modeled by a Stateflow® Chart block. Both blocks contain the same state transition logic including two states, `CreatePart` and `WaitForQA`.

- The `CreatePart` state represents the production of a `Part` in `pi` seconds.
- The `WaitForQA` state represents the wait for the quality control department for the `Part`'s validation.



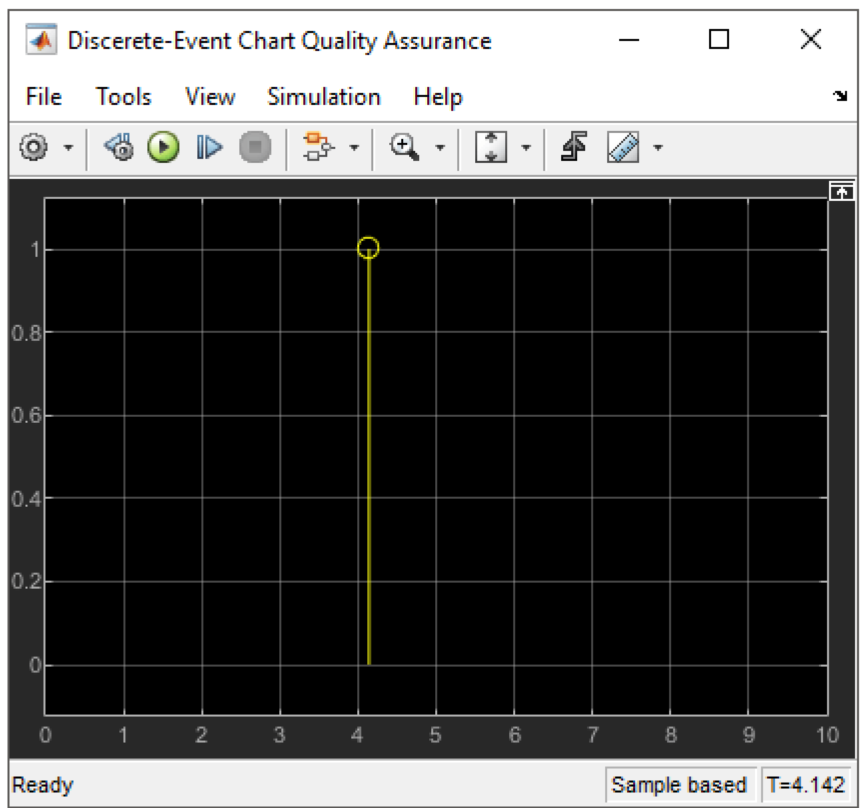

Similarly, Quality Assurance is modeled by a Discrete-Event Chart block and Quality Assurance Chart is modeled by using a Stateflow® Chart block. Both blocks contain the same state transition logic including three states, `WaitForPart`, `Evaluating`, and `Finished`.

- The `WaitForPart` state represents the wait for the generated Part.
- When the Part arrives, the block transitions to the `Evaluating` state to represent the start of the evaluation process.
- After `1` second, the evaluation is complete and the block transitions to `Finished` state.
- The Part departs the block and the block transitions back to the `WaitForPart` state.

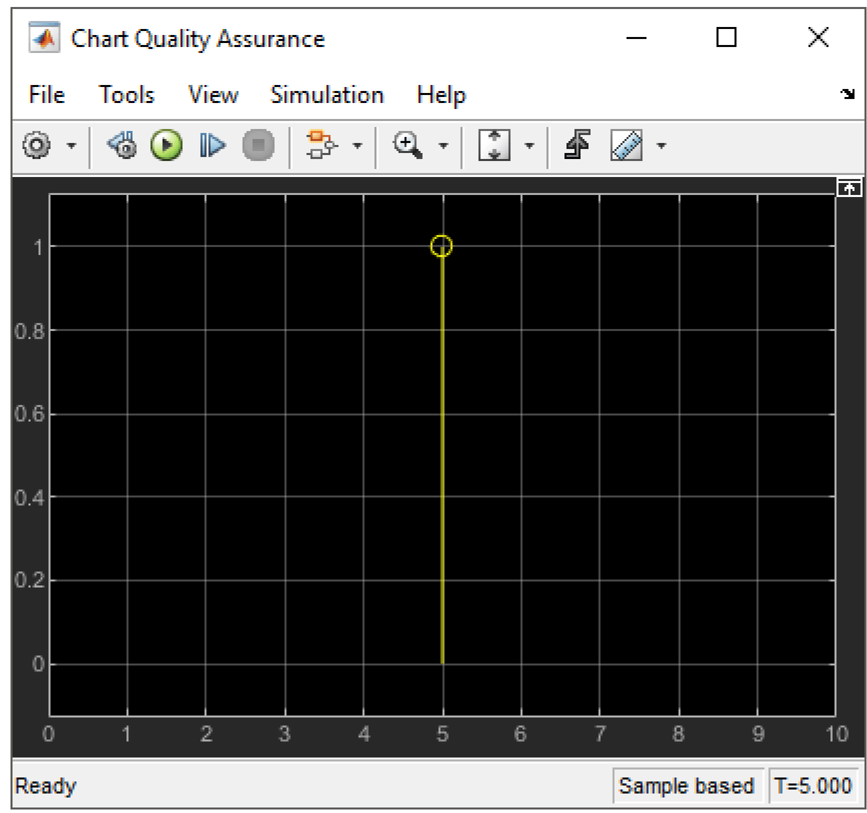


### Simulation Results

Simulate the model. Observe the Scope block connected to the Quality Assurance block. The block outputs the Part after `4.14` seconds, which is the sum of `3.14` seconds required for the Part's generation and 1 s for its quality control.

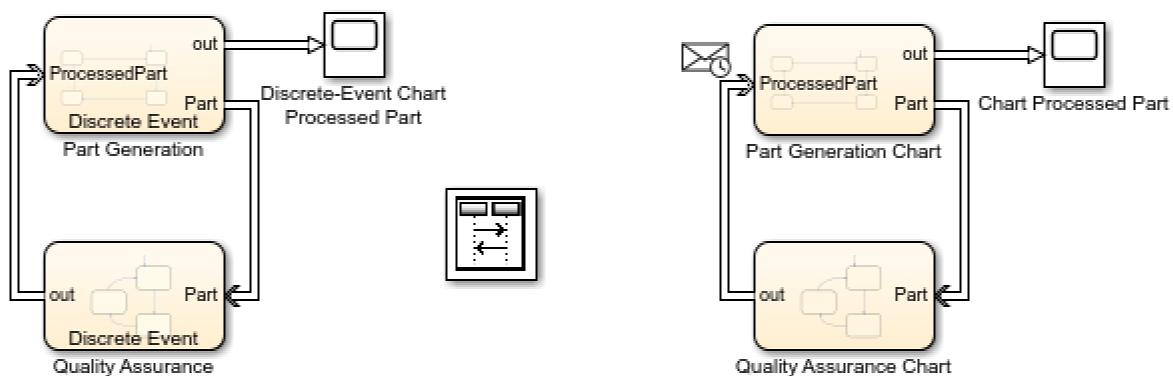



Observe the Scope block that is connected to the Quality Assurance Chart block. The block outputs the Part after 5 seconds, which is the sum of 4 seconds for the Part's generation and 1 second for its quality control as a result of fixed step size 1. This difference is based on the precise timing property of the Discrete-Event chart. For more information, see “Discrete-Event Chart Precise Timing” on page 8-6.




#### Further Modify the Model

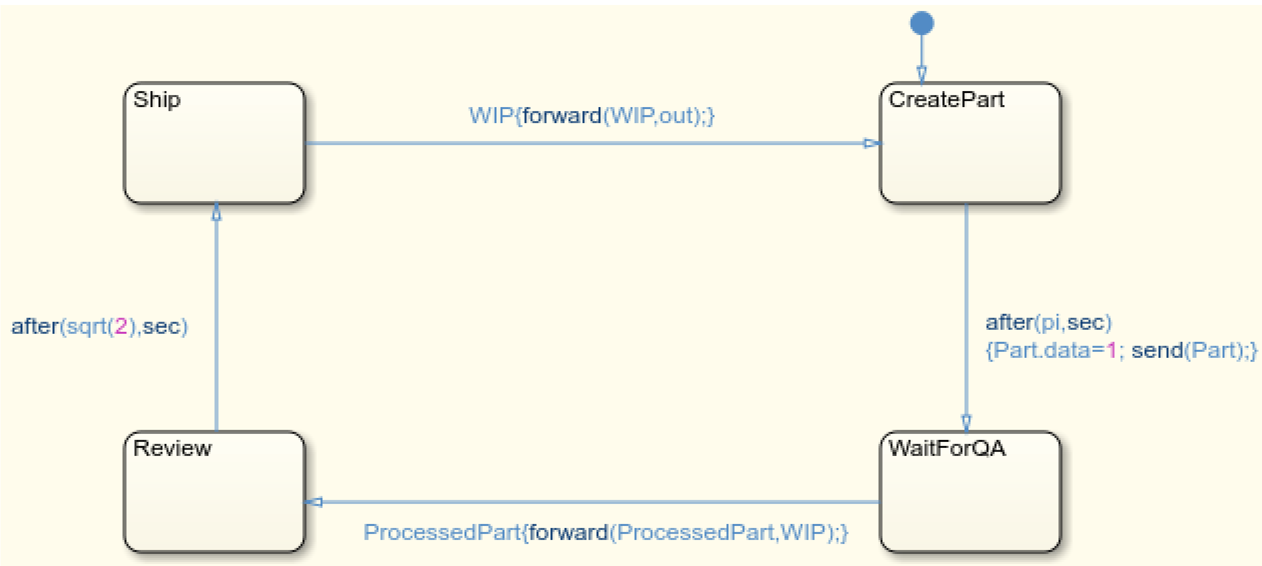
Open PartQualityControlShip which is the modified the model that sends the processed Part back to the Part Generation block for shipment. In the `PartQualityControlShip` model, the modified Part Generation and Part Generation Chart blocks contain the same set of additional states and transitions.




In the Part Generation and Part Generation Chart Blocks:

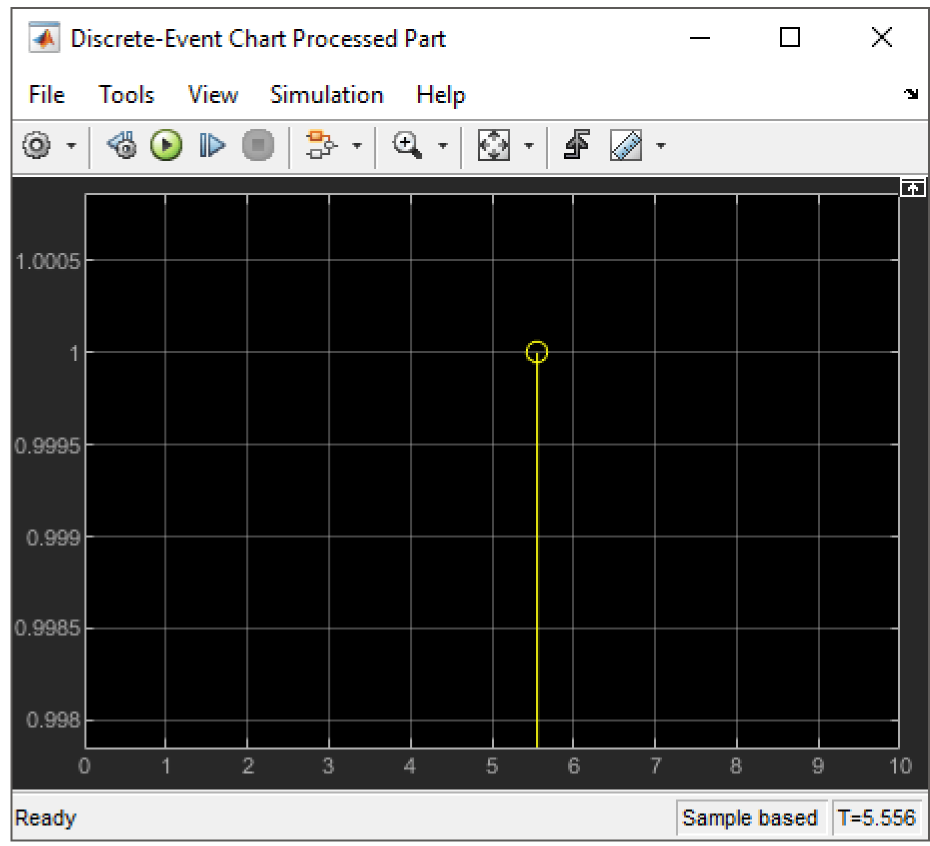
- The `Review` state represents the review of the quality control report for the `ProcessedPart`. When the `ProcessedPart` returns, the block transitions to the `Review` state.

- When the review is complete after `sqrt(2)` seconds, the block transitions to the `Ship` state.
- When the processed Part is shipped to the customer, the block transitions back to the `CreatePart` state to generate a new part.




#### Simulation Results

Simulate the modified model. Observe that the processed Part departs the Part Generation block after `5.55` seconds, which is the sum of `4.14` required for part generation and quality control and 1.41 for the review before shipment.

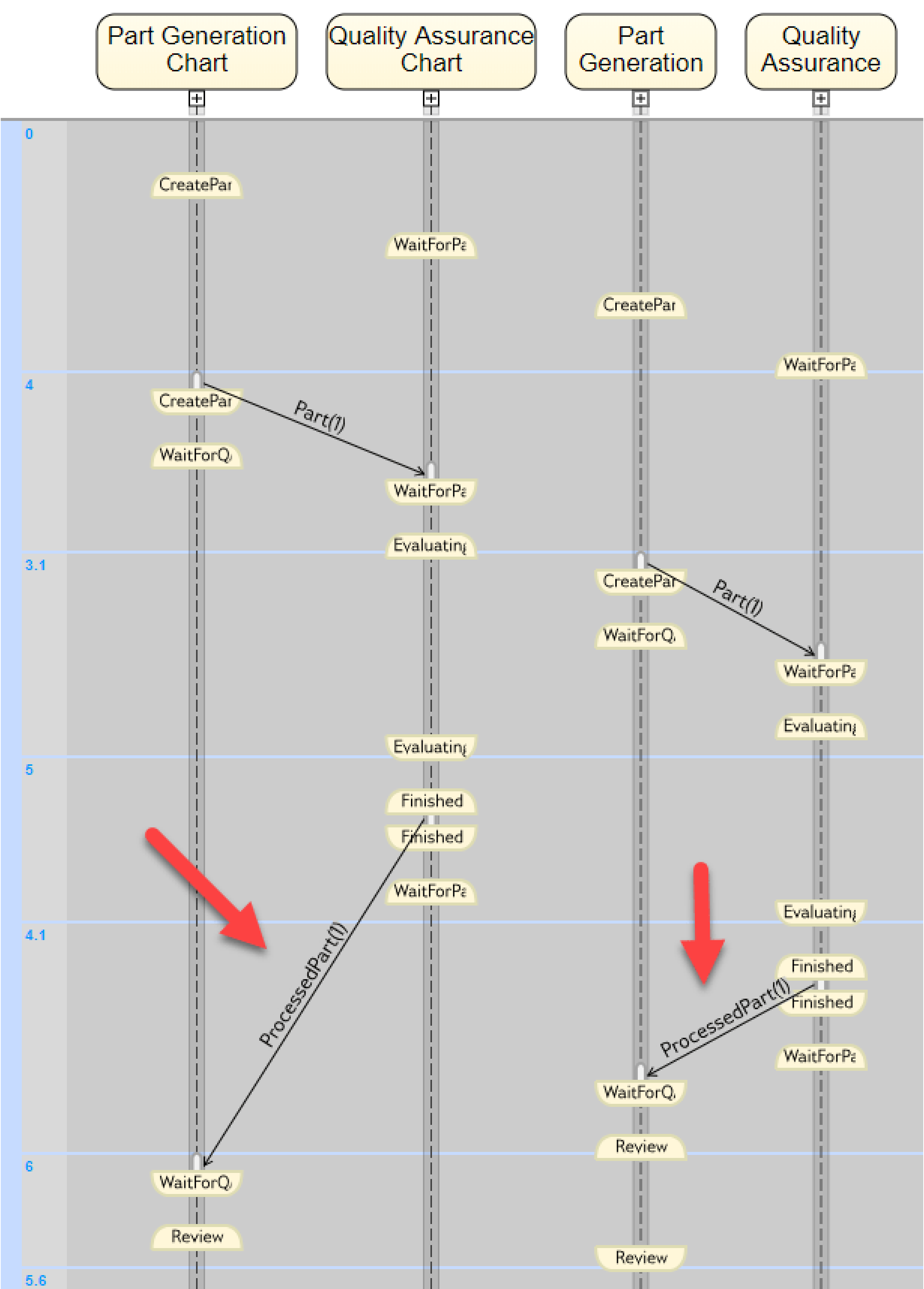



Observe that the processed Part departs the Part Generation Chart after 8 seconds, which is the sum of 5 required for part generation and quality control, 2 for the review before shipment, and 1 for the block's static scheduling.

Observe the Sequence Viewer block. Each time grid row bordered by two blue lines contains events that occur at the same simulation time. The Sequence Viewer window shows events vertically, ordered in time, and uses a combination of linear and nonlinear displays. For more information, see "Use the Sequence Viewer to Visualize Messages, Events, and Entities" on page 5-24.

The `ProcessedPart` is sent from Quality Assurance block to Part Generation at `4.1` and the Part's arrival triggers the Discrete-Event Chart block immediately. At time `5`, the `ProcessedPart` is sent from the Quality Assurance Chart to the Part Generation Chart. However, the Part Generation Chart waits for the next sample time hit at `6` after the message arrival to execute.

In the order, Part Generation Chart executes first and Quality Assurance Chart executes second in one sample time hit. That is the reason why Part Generation Chart block waits for the next sample time hit to execute as the first block in the order.






## See Also

Discrete-Event Chart

## More About

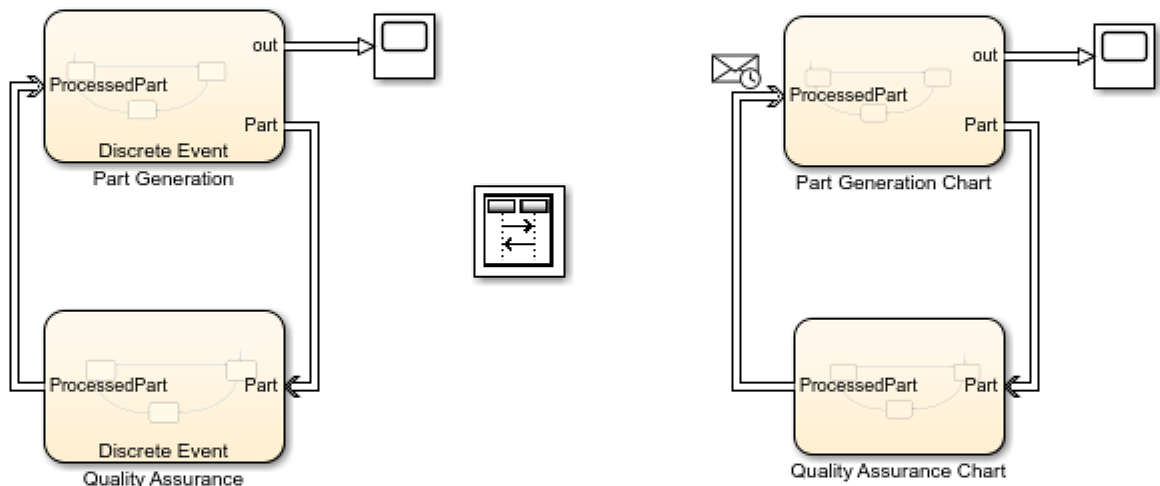

- "Create Custom Queuing Systems Using Discrete-Event Stateflow Charts" on page 8-2
- "Discrete-Event Chart Precise Timing" on page 8-6
- "Dynamic Scheduling of Discrete-Event Chart Block" on page 8-18

# Dynamic Scheduling of Discrete-Event Chart Block

This example shows how to use the dynamic scheduling that the Discrete-Event Chart block provides. A Discrete Event Chart block can execute zero or multiple times in a time step. The example compares the behaviors of the Discrete-Event Chart and Stateflow® Chart blocks. Both blocks require a Stateflow® license.

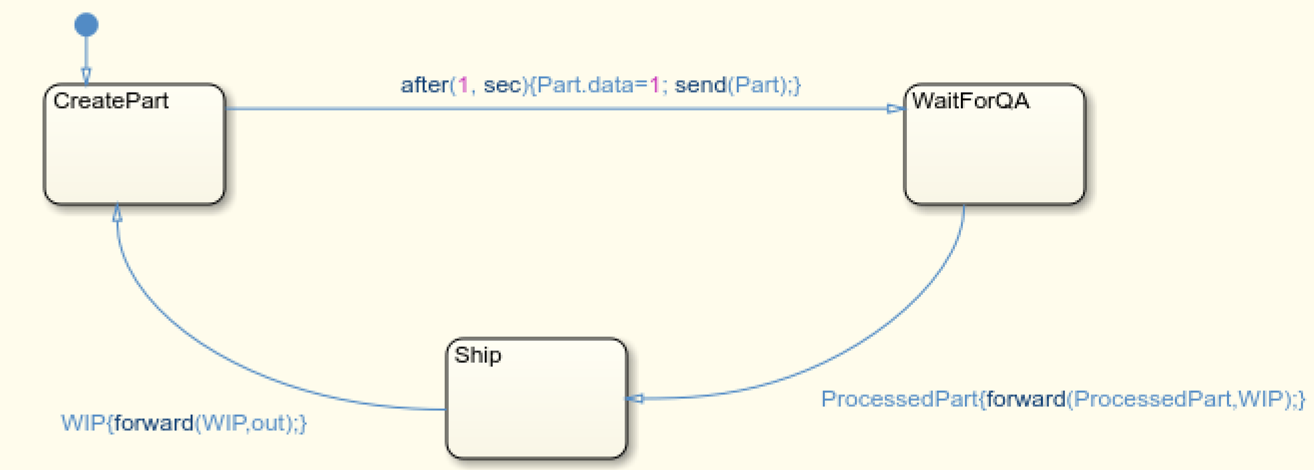
In this example, a bicycle part is generated every second by the Part Generation block. Its quality control is simultaneously performed when the part is in the assembly line. The quality control process takes `1` s to restart. This process is modeled by the Quality Assurance block.

The solver is set to `Fixed-step` with step size `1`, and for all the Stateflow® Chart blocks, the `Enable Super Step Semantics` option is selected. For more information, see “Super Step Semantics” (Stateflow).

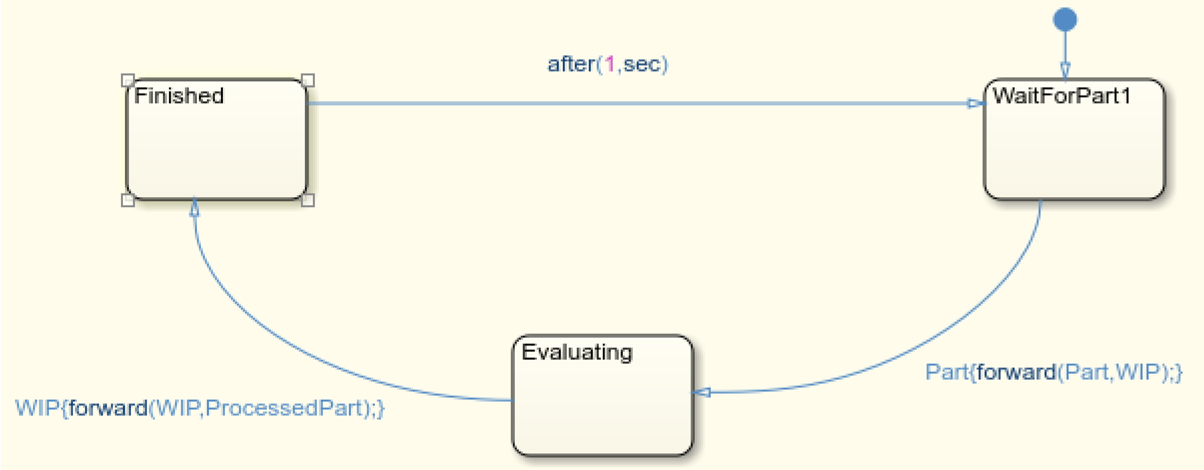



### Model Description

In the model, Part Generation is modeled by a Discrete-Event Chart block and Part Generation Chart is modeled by a Stateflow® Chart block. Both blocks contain the same state transition logic including three states, `CreatePart`, `WaitForQA`, and `Ship`.




- After 1 s, a Part is generated and the Chart transitions from the `CreatePart` to `WaitForQA`.
- The quality control is simultaneous and the `ProcessedPart` returns back immediately. The block transitions to the `Ship` state and after the `ProcessedPart` is shipped to the `CreatePart` state.

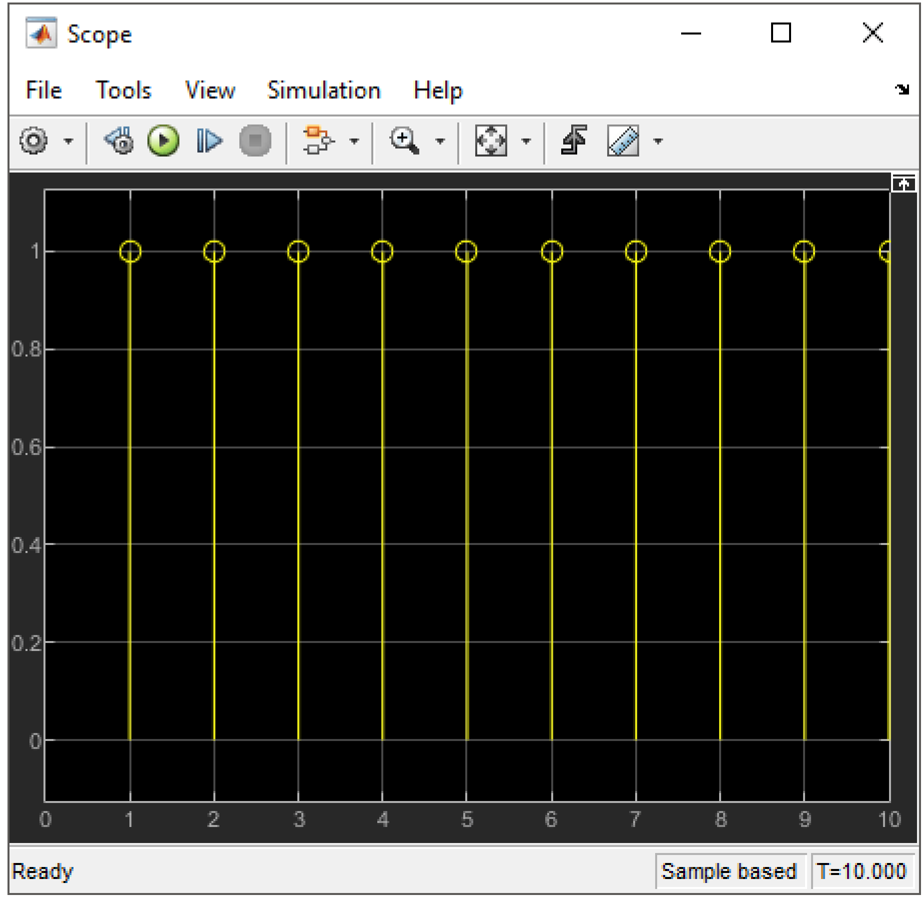



Similarly, the Quality Assurance is modeled by a Discrete-Event Chart while the Quality Assurance Chart is modeled by a Stateflow® Chart block. Both blocks contain the same state transition logic including three states, `WaitForPart`, `Evaluating`, and `Finished`.

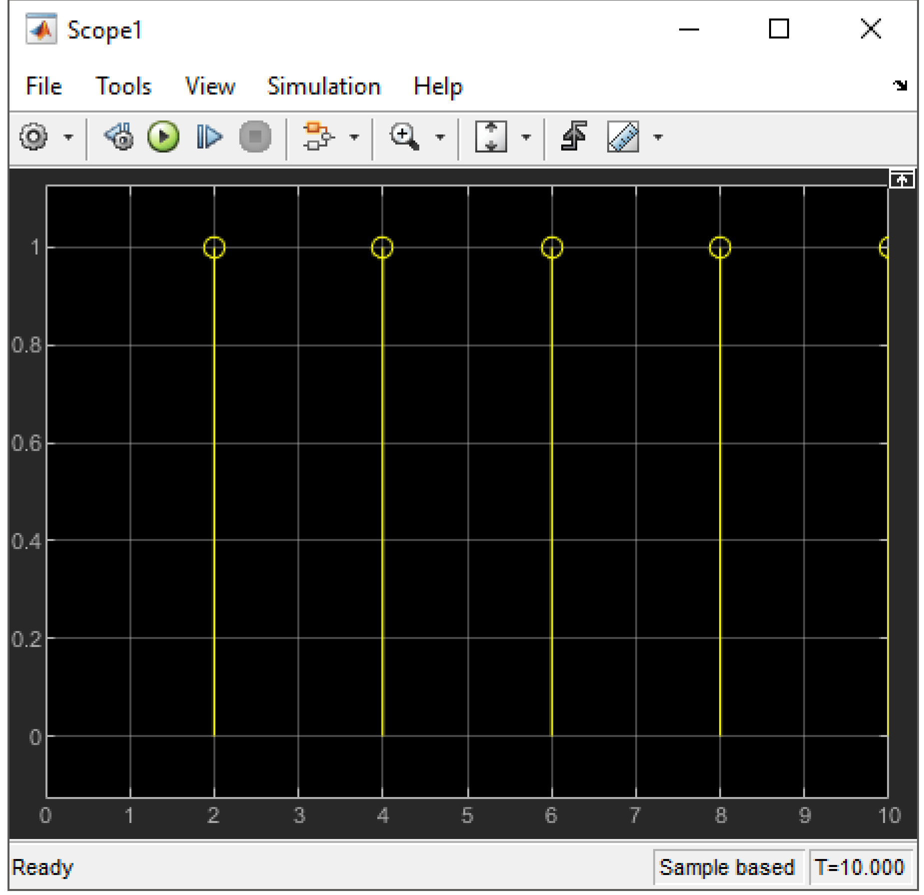
- The `WaitForPart` state represents the wait for the generated Part. When the Part arrives, the block transitions to the `Evaluating` state.
- Then the `ProcessedPart` is immediately sent back to Part Generation and the block transitions to the `Finished` state.
- After 1 s, the block returns to the `WaitForPart` state.

#### Simulation Results

- Simulate the model. Observe the Scope block connected to the Part Generation block. The Parts depart the facility every second.



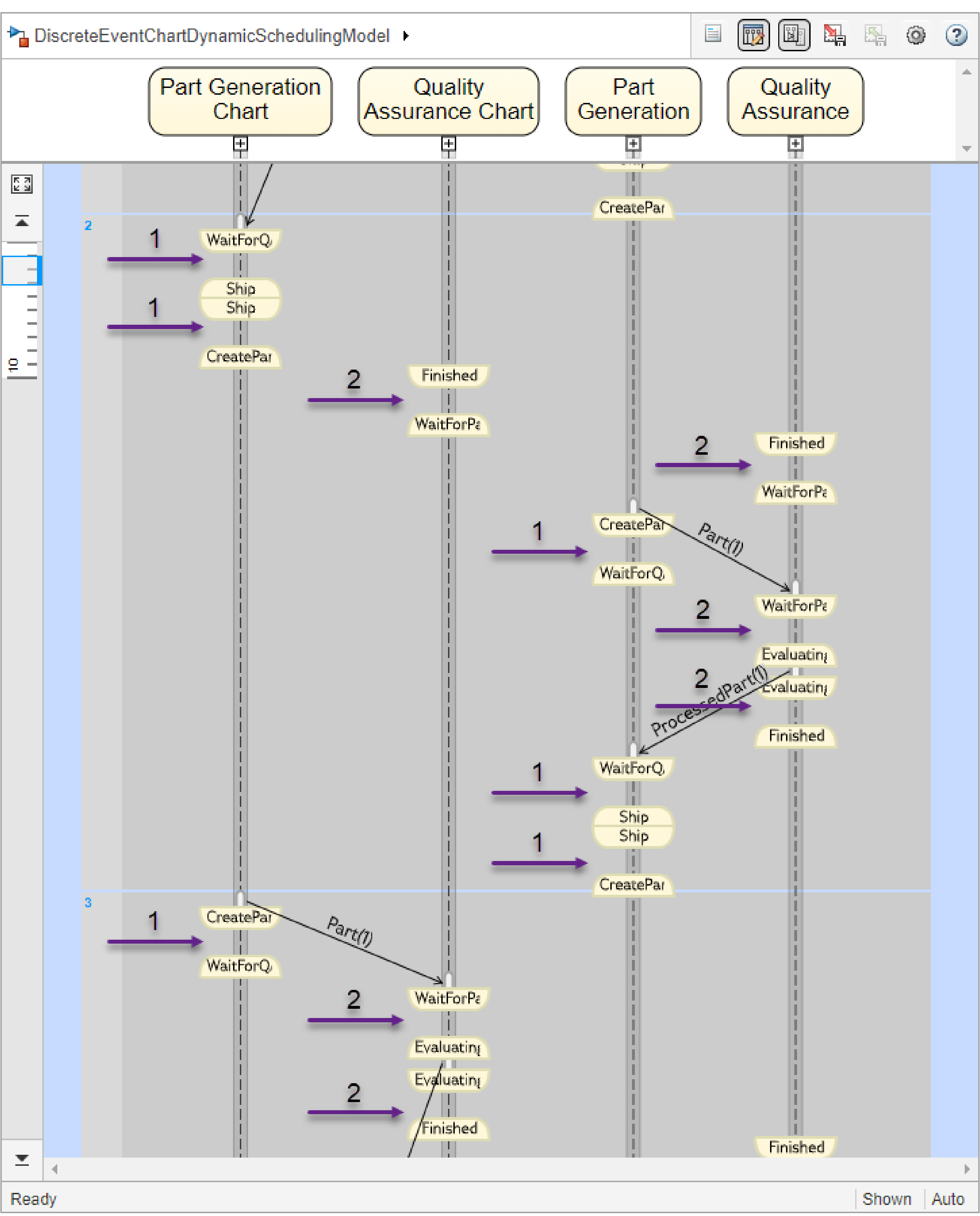

Observe the Scope block connected to the Part Generation Chart block, which displays that the parts are generated in every two seconds.




The difference is due to the dynamic scheduling property of the Discrete-Event Chart block. For instance, observe the Sequence Viewer block. Each time grid row, bordered by two blue lines, contains events that occur at the same simulation time. For more information, see "Use the Sequence Viewer to Visualize Messages, Events, and Entities" on page 5-24.

In the second and third simulation time step, the static scheduling of the Stateflow® Chart blocks causes their execution with a fixed order, in which the Part Generation Chart labeled `1` is executed first and the Quality Assurance Chart labeled `2` is executed second for each time step. The sequence is `1,1,2` for the second time step and `1,2,2` for the third time step.

The dynamic scheduling property of the Discrete-Event Chart allows multiple executions of the Part Generation and Quality Assurance blocks at each time step with the changing order. For example, in the second time step, the order becomes `2,1,2,2,1,1`.






## See Also

Discrete-Event Chart

## More About


- “Create Custom Queuing Systems Using Discrete-Event Stateflow Charts” on page 8-2
- “Discrete-Event Chart Precise Timing” on page 8-6
- “Trigger a Discrete-Event Chart Block on Message Arrival” on page 8-9

# Build Discrete-Event Systems Using System Objects


- “Create Custom Blocks Using MATLAB Discrete-Event System Block” on page 9-2
- “Delay Entities with a Custom Entity Storage Block” on page 9-9
- “Create a Custom Entity Storage Block with Iteration Event” on page 9-14
- “Custom Entity Storage Block with Multiple Timer Events” on page 9-19
- “Custom Entity Generator Block with Signal Input and Signal Output” on page 9-24
- “Build a Custom Block with Multiple Storages” on page 9-31
- “Create a Custom Resource Acquirer Block” on page 9-38
- “Create a Discrete-Event System Object” on page 9-44
- “Generate Code for MATLAB Discrete-Event System Blocks” on page 9-48
- “Customize Discrete-Event System Behavior Using Events and Event Actions” on page 9-51
- “Call Simulink Function from a MATLAB Discrete-Event System Block” on page 9-55
- “Resource Scheduling Using MATLAB Discrete-Event System and Data Store Memory Blocks” on page 9-58

# Create Custom Blocks Using MATLAB Discrete-Event System Block

**In this section...**


"Entity Types, Ports, and Storage in a Discrete-Event System Framework" on page 9-2

"Events" on page 9-5

"Implement a Discrete-Event System Object with MATLAB Discrete-Event System Block" on page 9-6


Discrete-Event System objects let you implement custom event-driven entity-flow systems using the MATLAB language. The MATLAB Discrete-Event System block enables you to use discrete-event System objects to create a custom block in your SimEvents model. You can author such discrete-event System objects via a set of MATLAB methods.

You can create a custom discrete-event System object from scratch that:

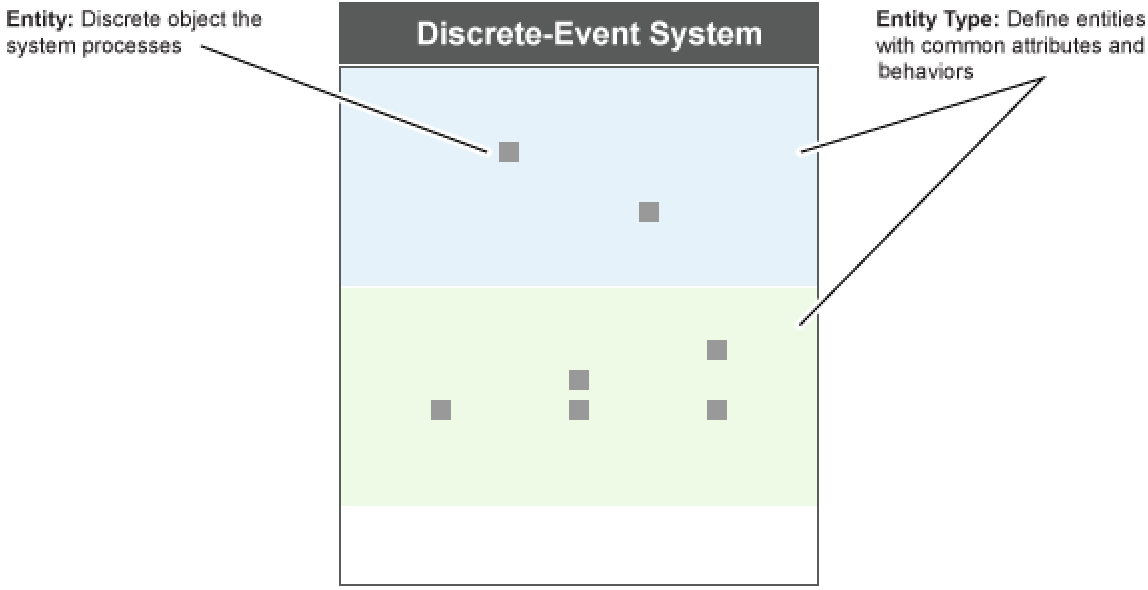
- Contains multiple entity storage elements, with each storage element containing multiple SimEvents entities, and configure it to sort entities in a particular order.
- Has an entity or a storage element that can schedule and execute multiple types of events. These events can model activities such as entity creation, consumption, search, transmission, and temporal delay.
- Can accept entity/signal as input/output, produce entity and signal as outputs, and support both built-in data types and structured/bus data types.
- Use MATLAB toolboxes for computation and scaling of complex systems.

The MATLAB Discrete-Event System block is similar to the MATLAB System block with these differences:

- The resulting discrete-event System object is an instantiation of the `matlab.DiscreteEventSystem` class rather than the `matlab.System` class.
- The `matlab.DiscreteEventSystem` has its own set of System object methods particular to discrete-event systems.
- The `matlab.DiscreteEventSystem` also inherits a subset of the MATLAB System methods. For a complete list of this subset, see "Create a Discrete-Event System Object" on page 9-44.

## Entity Types, Ports, and Storage in a Discrete-Event System Framework

An entity is a discrete object that the system processes. An entity has a type and the entity type defines a class of entities that share a common set of data specifications and run-time methods. Examples of data specifications include dimensions, data type, and complexity.



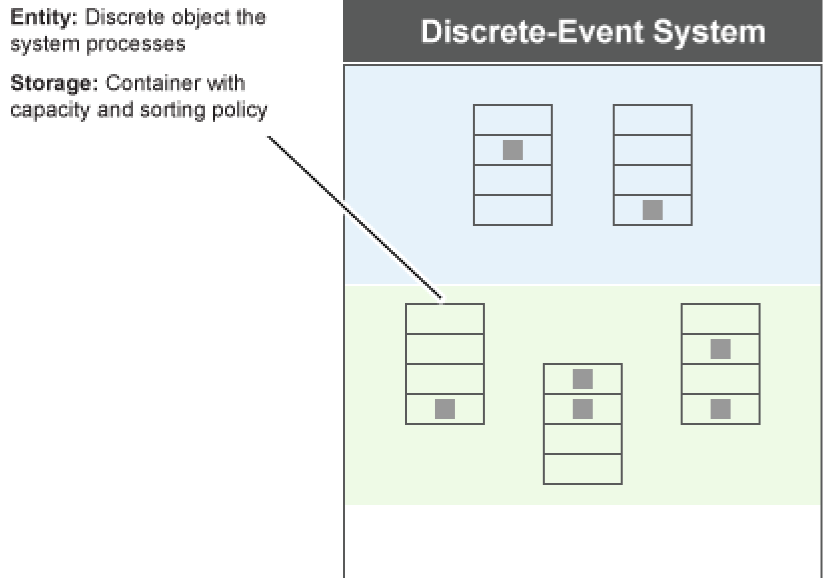

Consider these guidelines when defining custom entity types using the `getEntityTypesImpl` method:

- You can specify multiple entity types. Each type must have a unique name.
- An entity storage element, input port, and output port must specify the entity type they work with.
- Specify or resolve common data specifications for an entity type. For example, an input port and an output port with the same entity type must have the same data type.
- When forwarding an entity, the source and destination data specifications must be same in these instances:
  - From an input port to a storage element
  - Between storage elements
  - From a storage element to an output port
- Each entity type can share a common set of event action methods. When naming these methods, to distinguish the entity type use this convention:

  *`entitytypeAction`*

  For example, if there are two entity types, `car` and `truck`, use method names such as:

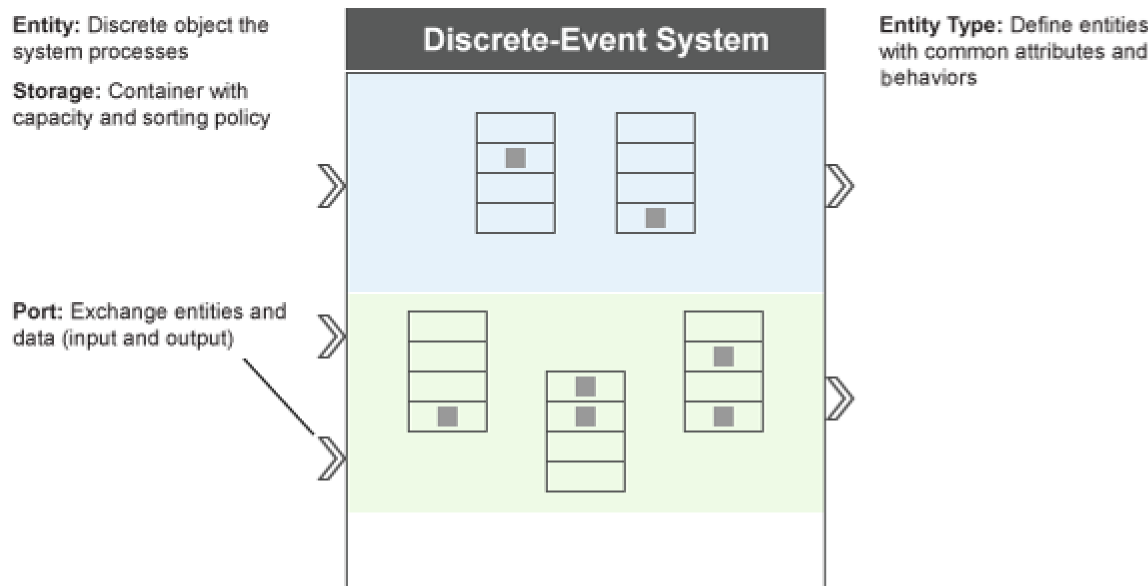
  ```
  carEntry
  truckEntry
  ```




During simulation, an entity always occupies a unit of storage space. Such storage spaces are provided by entity storage elements. A discrete-event System object can contain multiple entity storage elements. Use the `getEntityStorageImpl` method to specify storage elements. A storage space is a container with these properties:

- Entity type — Entity type this storage is handling.
- Capacity — Maximum number of entities that the storage can contain.
- Storage type — Criteria to sort storage entities (`FIFO`, `LIFO`, and `priority`).
- Key name — An attribute name used as key name for sorting. This property is applicable only when the storage type is `priority`.
- Sorting direction — Ascending or descending priority queues. This property is applicable only when the storage type is `priority`.

You can access any entity at an arbitrary location of a storage and specify events.



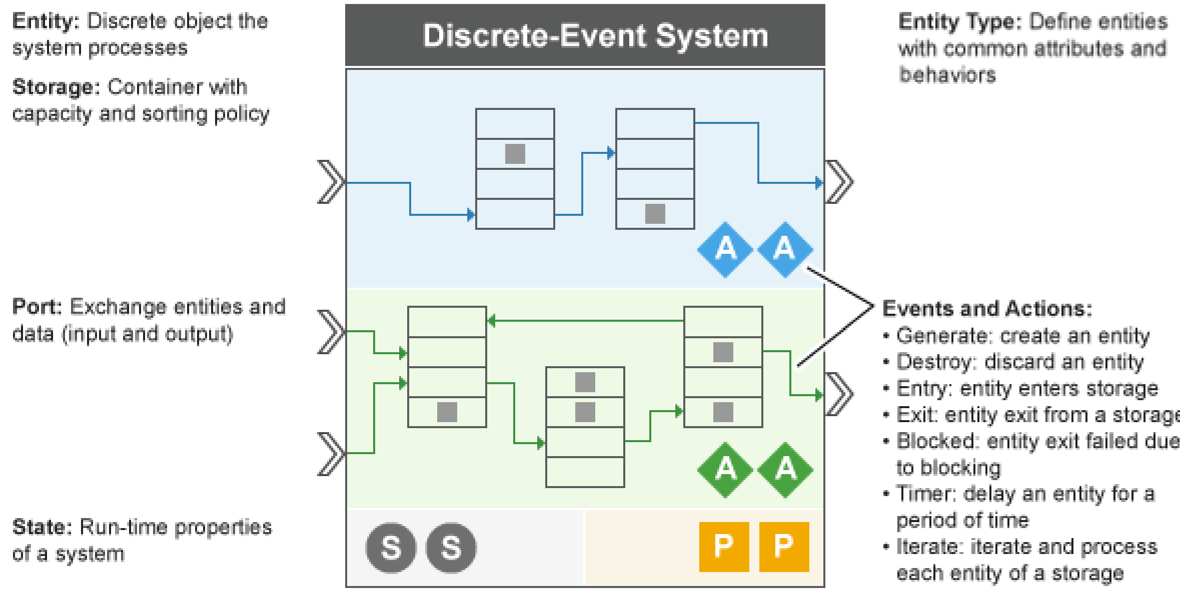

Ports enable a discrete-event System object to exchange entities and data with other blocks or model components. You can specify a variable number of input and output ports using the `getNumInputsImpl` and `getNumOutputsImpl` methods. You can also specify which ports are entity ports and the entity types for these ports. Use the `getEntityPortsImpl` method to specify these port properties.

## Events

You can schedule events for a discrete-event System object to execute. Events are associated with user-defined actions. An event action defines a custom behavior by changing state or entity values, and executing the next set of events.

You can use methods and functions to:

- Schedule events
- Define event actions in response to events
- Initialize events
- Cancel events




A MATLAB discrete-event System object can have these types of events:

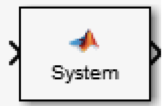
- Storage events — You can schedule these events on a storage element. The actor is a storage element.
  - You can generate a new entity inside a storage element.
  - You can iterate each entity of a storage element.
- Entity events — You can schedule these events on an entity. Actor is an entity.
  - You can delay an entity.
  - You can forward an entity from its current storage to another storage or output port.
  - You can destroy the existing entity of a storage element.

For more information on using events and event actions, see “Customize Discrete-Event System Behavior Using Events and Event Actions” on page 9-51.

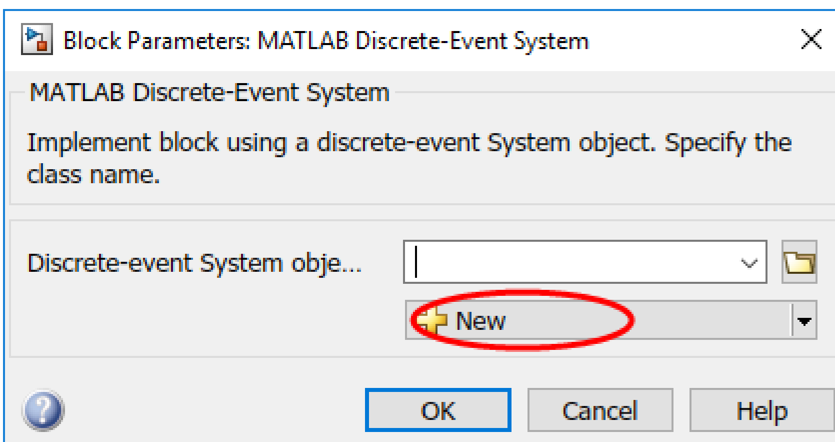
## Implement a Discrete-Event System Object with MATLAB Discrete-Event System Block

To Implement a custom block by assigning a discrete-event System object, follow these steps.

1. Open a new model and add the MATLAB Discrete-Event System block from the SimEvents library.




2. In the block dialog box, from the **New** list, select `Basic` to create a System object from a template.

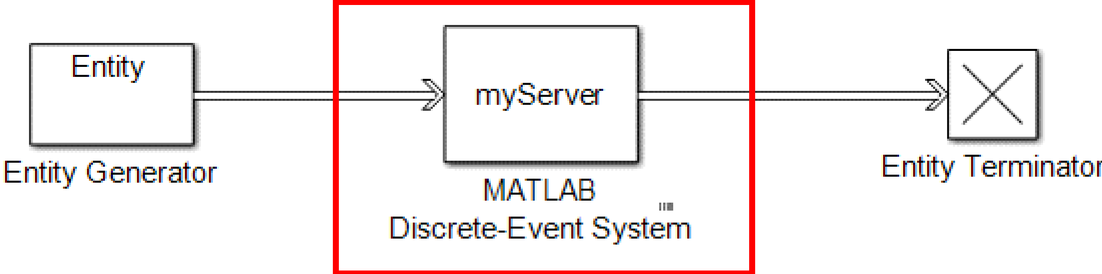



Modify the template as needed and save the System object.

You can also modify the template and define Discrete-Event System objects from the MATLAB Editor using code insertion options. By selecting **Insert Property** or **Insert Method**, the MATLAB Editor adds predefined properties, methods, states, inputs, or outputs to your System object. Use these tools to create and modify System objects faster, and to increase accuracy by reducing typing errors.

3 If the System object exists, in the block dialog box, enter its name for the **Discrete-event System object name** parameter. Click the list arrow to see the available discrete-event System objects in the current folder.

The MATLAB Discrete-Event System block icon and port labels update to the icons and labels of the corresponding System object. Suppose that you select a System object named `myServer` in your current folder and generate a custom entity server block that serves entities and outputs each entity through the output port. Then, the block updates as shown in the model.




Many different MATLAB System object functions allow you to capture the properties and implement custom behaviors. The provided template is simplified, but you can add complexity by editing event actions, introducing actions, and modifying parameters. The object-oriented programming features of MATLAB System object enable you to scale your model, and interface it with the graphical programming features of SimEvents.

These topics walk you through a complete workflow for creating custom blocks with distinct functionalities.

1 "Delay Entities with a Custom Entity Storage Block" on page 9-9
2 "Create a Custom Entity Storage Block with Iteration Event" on page 9-14

3. “Custom Entity Storage Block with Multiple Timer Events” on page 9-19
4. “Custom Entity Generator Block with Signal Input and Signal Output” on page 9-24
5. “Build a Custom Block with Multiple Storages” on page 9-31
6. “Create a Custom Resource Acquirer Block” on page 9-38

For other examples of MATLAB Discrete-Event System block and discrete-event System objects, see SimEvents Examples in the Help browser.

To use provided custom blocks, in the SimEvents library, double-click the Design Patterns block. The **MATLAB Discrete-Event System** category contains these design patterns:

| Example | Application |
| --- | --- |
| Custom Generator | Implement a more complicated entity generator. |
| Custom Server | Use a custom server. |
| Selection Queue | Select a specific entity to output from a queue. |

For more information, see “SimEvents Common Design Patterns”.

## See Also

`matlab.DiscreteEventSystem` | `matlab.System`

## More About

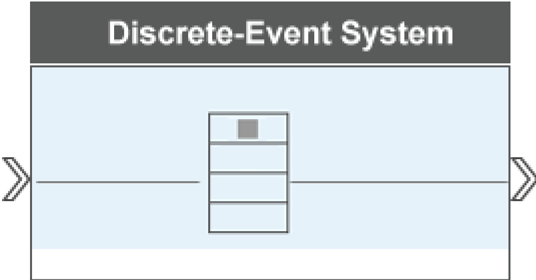
- “Delay Entities with a Custom Entity Storage Block” on page 9-9
- “Integrate System Objects Using MATLAB System Block”
- “Create a Discrete-Event System Object” on page 9-44
- “Customize Discrete-Event System Behavior Using Events and Event Actions” on page 9-51

# Delay Entities with a Custom Entity Storage Block

This example shows how to use discrete-event System object methods to create a custom entity storage block that has one input port, one output port, and one storage element. The discrete-event System object is the instantiation of the `matlab.DiscreteEventSystem` class, which allows you to use the implementation and service methods provided by this class. Then, you use the MATLAB Discrete-Event System block to integrate the System object into a SimEvents model.

The custom MATLAB Discrete-Event System block accepts an entity from its input port and forwards it to its output port with a specified delay. The figure visualizes the block using the discrete-event system framework.

To open the model and to observe the behavior of the custom block, see `CustomEntityStorageBlockExample`.




## Create the Discrete-Event System Object

1 Create a new script and inherit the `matlab.DiscreteEventSystem` class.

```
classdef CustomEntityStorageBlock < matlab.DiscreteEventSystem
```

2 Add a custom description to the block.

```
% A custom entity storage block with one input, one output, and one storage.
```

3 Declare two nontunable parameters `Capacity` and `Delay` to represent the storage capacity and the entity departure delay from the storage.

```
% Nontunable properties
    properties (Nontunable)
    % Capacity
        Capacity = 1;
    % Delay
        Delay = 4;
    end
```

The parameters capture the properties of the block.

- Tunable parameters can be tuned during run time.
- Non-tunable parameters cannot be tuned during run time.

4 Specify these methods and set access to protected.

```
    methods (Access = protected)

        % Specify the number of input ports.
        function num = getNumInputsImpl(~)
            num = 1;
        end
```

```
        % Specify the number of output ports.
        function num = getNumOutputsImpl(~)
            num = 1;
        end
        % Specify a new entity type Car.
        function entityTypes = getEntityTypesImpl(obj)
            entityTypes = obj.entityType('Car');
        end
        % Specify Car as the entity type that is used in
        % input and output ports.
        function [inputTypes,outputTypes] = getEntityPortsImpl(obj)
            inputTypes = {'Car'};
            outputTypes = {'Car'};
        end
        % Specify the storage type, capacity, and connection to
        % the input and output ports.
        function [storageSpecs, I, O] = getEntityStorageImpl(obj)
            storageSpecs = obj.queueFIFO('Car', obj.Capacity);
            % First element of I indicates the entity storage index 1 that is
            % connected to input 1.
            I = 1;
            % First element of O indicates the entity storage index 1 that is
            % connected to output 1.
            O = 1;
        end

    end
```

Only one storage sorts cars in a first-in-first-out (FIFO) manner. The `Capacity` parameter of the object defines the server capacity.

The method `getEntityStorageImpl()` also specifies the connections between the ports and the storage, `I` and `O`.

- The return value `I` is a vector of elements $i = 1, \dots n$ where its length $n$ is equal to the number of input ports.

  In this example, `n` is `1` because only one input port is declared.
- The $i^{th}$ element indicates the entity storage index that the $i^{th}$ input port connects to.

  In this example, input port `1` is connected to storage `1`.
- If an input port is a signal port, the corresponding element is `0`.

Similarly the return value `O` is used to define the connections between the storage and the output port.

**5** Specify an `eventForward` event to forward an entity of type `Car` to the output when it enters the storage.

```
    function [entity,event] = CarEntry(obj,storage,entity,source)
        % Specify event actions when entity enters storage.
        event = obj.eventForward('output', 1, obj.Delay);
    end
```

A `Car` entry to the storage invokes an event action and the event `obj.eventForward` forwards `Car` to the output with index `1` with a delay specified by `obj.Delay`.

You can use the input arguments of this method to create custom behavior. The argument `obj` is the discrete-event System object inherited by the method. The argument `storage` is the index of the storage element that the entity enters. The argument `entity` is the entity that enters the storage and it has two fields, `entity.sys` and `entity.data`. The argument `source` is the source location of the entity that enters the storage.

**Note** You cannot manipulate entity data within an exit action.

**6** Name your discrete-event System object `CustomEntityStorageBlock` and save it as `CustomEntityStorageBlock.m`.

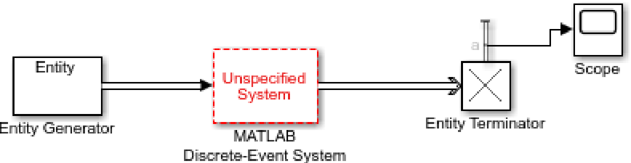
The custom block represents a simplified gas station that can serve one car at a time. A car arrives at the gas station and is serviced for 4 minutes before departing the station.

### See the Code to Generate Custom Entity Storage Block

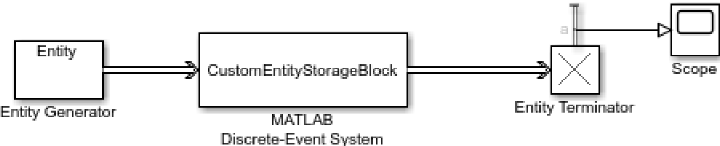
```
classdef CustomEntityStorageBlock < matlab.DiscreteEventSystem

    % A custom entity storage block with one input, one output, and one storage.

    % Nontunable properties
    properties (Nontunable)
    % Capacity
        Capacity = 1;
    % Delay
        Delay = 4;
    end

    methods (Access = protected)

        function num = getNumInputsImpl(~)
            num = 1;
        end

        function num = getNumOutputsImpl(~)
            num = 1;
        end

        function entityTypes = getEntityTypesImpl(obj)
            entityTypes = obj.entityType('Car');
        end

        function [inputTypes,outputTypes] = getEntityPortsImpl(obj)
            inputTypes = {'Car'};
            outputTypes = {'Car'};
        end

        function [storageSpecs, I, O] = getEntityStorageImpl(obj)
            storageSpecs = obj.queueFIFO('Car', obj.Capacity);
            I = 1;
            O = 1;
        end

    end

   methods

        function [entity,event] = CarEntry(obj,storage,entity,source)
            % Specify event actions when entity enters storage.
             event = obj.eventForward('output', 1, obj.Delay);
        end

    end

end
```

## Implementing the Custom Entity Storage Block

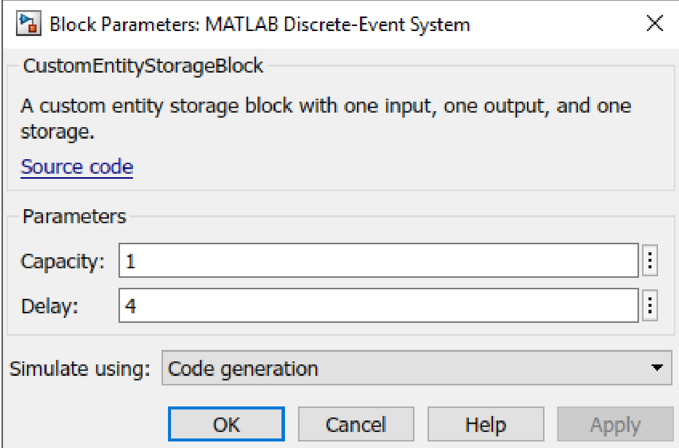
**1** Create a model using an Entity Generator block, MATLAB Discrete-Event System block, and an Entity Terminator block.



**2** Open the MATLAB Discrete-Event System block, and set the **Discrete-event System object name** to `CustomEntityStorageBlock`.

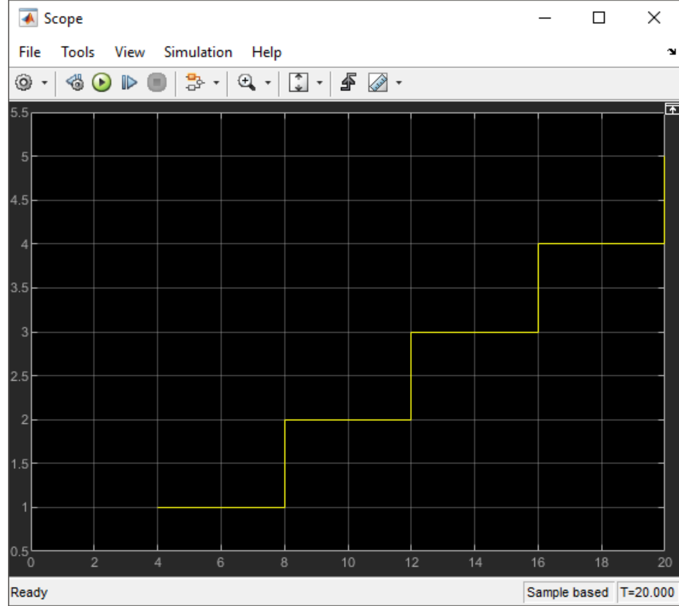



**3** Double-click the MATLAB Discrete-Event System block to observe its capacity and delay.




**4** Output the **Number of entities arrived, a** statistic from the Entity Terminator block and connect it to a scope

**5** Increase the simulation time to `20` and run the simulation. Observe the entities arriving at the Entity Terminator block with a delay of `4`.



## See Also

`entry` | `getEntityPortsImpl` | `getEntityStorageImpl` | `getEntityTypesImpl` | `matlab.DiscreteEventSystem` | `matlab.System`

## More About

- “Integrate System Objects Using MATLAB System Block”
- “Create a Discrete-Event System Object” on page 9-44
- “Generate Code for MATLAB Discrete-Event System Blocks” on page 9-48
- “Call Simulink Function from a MATLAB Discrete-Event System Block” on page 9-55

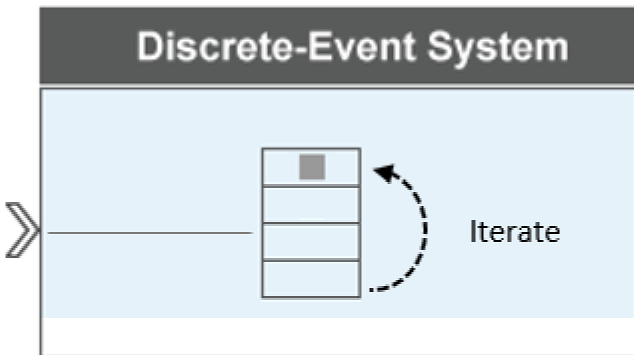
# Create a Custom Entity Storage Block with Iteration Event

A discrete-event System object can contain multiple event types for manipulating entities, acting on the storages, and resource management. When an event is due for execution, a discrete-event system can respond to that event by invoking event actions. The goal of this example is to show how to work with events and event actions when creating a custom block. To see the list of provided event and event actions, see “Customize Discrete-Event System Behavior Using Events and Event Actions” on page 9-51.

To open the model and to observe the behavior of the custom block, see `CustomEntityStorageBlockWithIterationEventExample`.

## Create the Discrete-Event System Object

In this example, a custom block allows entities to enter its storage element through its input port. The storage element sorts the entities based on their `Diameter` attribute in ascending order. Every entity entry to the block's storage invokes an iteration event to display the diameter and the position of each entity in the storage.




The storage element allows you to define its capacity to store and sort entities during which any entity can be accessed and manipulated. In this example, the storage with capacity `5` is used to store and sort car wheels based on their `Diameter` attribute in an ascending order. When a new wheel enters the storage, an iteration event `eventIterate` is invoked, which triggers an iteration event action `iterate` to display wheel positions in the storage and their diameter.

### See the Code to Generate the Custom Storage Block with Iteration Event

```
classdef CustomEntityStorageBlockIteration < matlab.DiscreteEventSystem

    % A custom entity storage block with one input port and one storage element.

    % Nontunable properties
    properties (Nontunable)
        % Capacity
        Capacity = 5;
    end
    % Create the storage element with one input and one storage.
    methods (Access=protected)

        function num = getNumInputsImpl(obj)
            num = 1;
        end

        function num = getNumOutputsImpl(obj)
            num = 0;
        end
```

```
        function entityTypes = getEntityTypesImpl(obj)
            entityType1 = obj.entityType('Wheel');
            entityTypes = entityType1;
        end

        function [inputTypes,outputTypes] = getEntityPortsImpl(obj)
            inputTypes = {'Wheel'};
            outputTypes={};

        end

        function [storageSpecs, I, O] = getEntityStorageImpl(obj)
            storageSpecs = obj.queuePriority('Wheel',obj.Capacity, 'Diameter','ascending');
            I = 1;
            O = [];

        end

    end
    % Entity entry event action
    methods

        function [entity, event] = WheelEntry(obj,storage,entity, source)
            % Entity entry invokes an iterate event.
            event = obj.eventIterate(1, '');
        end

        % The itarate event action
        function [entity,event,next] = WheelIterate(obj,storage,entity,tag,cur)
            % Display wheel id, position in the storage, and diameter.
            coder.extrinsic('fprintf');
            fprintf('Wheel id %d, Current position %d, Diameter %d\n', ...
                entity.sys.id, cur.position, entity.data.Diameter);
            if cur.size == cur.position
                fprintf('End of Iteration \n')
            end
            next = true;
            event=[];
        end

    end

end
```

## Define Custom Block Behavior

1. Define a storage with capacity `obj.Capacity`, which sorts wheels based in their priority value. The priority values are acquired from the `Diameter` attributes of the entities and are sorted in ascending order.

   ```
   function [storageSpecs, I, O] = getEntityStorageImpl(obj)
           storageSpecs = obj.queuePriority('Wheel',obj.Capacity, 'Diameter','ascending');
           I = 1;
           O = [];
   end
   ```

2. A wheel's entry into the storage invokes an iterate event.

   ```
   function [entity, event] = WheelEntry(obj,storage,entity, source)
           % Entity entry invokes an iterate event.
           event = obj.eventIterate(1, '');
   end
   ```

   Input argument `1` is the storage index for the iterate event, and `''` is the tag name.

3. The iterate event invokes an iterate event action.

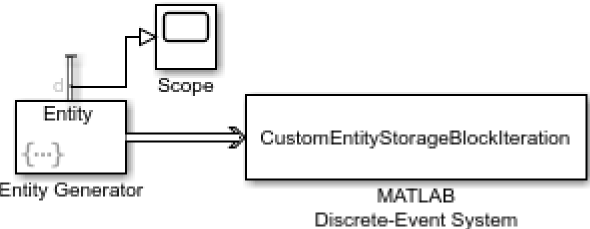
   ```
   % The itarate event action
   function [entity,event,next] = WheelIterate(obj,storage,entity,tag,cur)
       % Display wheel id, position in the storage, and diameter.
           coder.extrinsic('fprintf');
           fprintf('Wheel id %d, Current position %d, Diameter %d\n', ...
               entity.sys.id, cur.position, entity.data.Diameter);
           if cur.size == cur.position
               fprintf('End of Iteration \n')
           end
   ```

```
        next = true;
        event=[];
    end
```

In the code, `coder.extrinsic('fprintf')` declares the function `fprintf()` as extrinsic function for code generation. For each iteration, the code displays the new wheel ID, current position, and diameter, which is used as sorting attribute.

## Implement Custom Block

1. Save the `.m` file as `CustomEntityStorageBlockIteration`. Link the System object to a SimEvents model by using a MATLAB Discrete-Event System block. For more information about linking, see “Create Custom Blocks Using MATLAB Discrete-Event System Block” on page 9-2.
2. Create a SimEvents model including the MATLAB Discrete-Event System block, and an Entity Generator block.



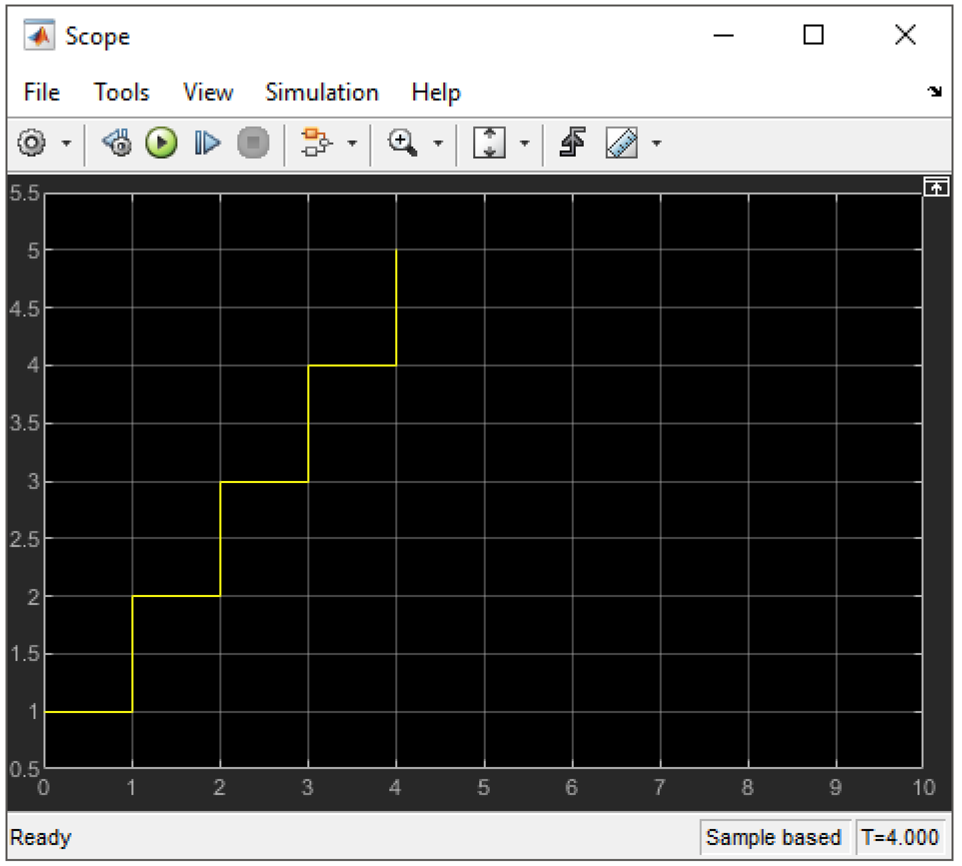

3. In the Entity Generator block:

   a. In the **Entity type** tab, set the **Attribute Name** as `Diameter`.

      The attribute `Diameter` is used to sort entities in the MATLAB Discrete-Event System block.
   b. In the **Event actions** tab, in the **Generate action** field, add this code to randomize the size of the incoming entities.

      ```
      entity.Diameter = randi([1 10]);
      ```
   c. In the **Statistics** tab, output the **Number of entities departed, d** statistic and connect to a scope.
4. Connect the blocks as shown and simulate the model.

   a. Observe that the Entity Generator block generates `5` entities since the capacity of the storage block is `5`.




**b** The Diagnostic Viewer displays the iteration event for each wheel entry to the storage. Each iteration displays ID, position, and diameter of the wheels. Observe how each wheel entry changes the order of the stored wheels. In the last iteration, 5 entities in the storage are sorted in ascending order.

```
Wheel id 1, Current position 1 and Diameter 9
End of Iteration
Wheel id 1, Current position 1 and Diameter 9
Wheel id 2, Current position 2 and Diameter 10
End of Iteration
Wheel id 3, Current position 1 and Diameter 2
Wheel id 1, Current position 2 and Diameter 9
Wheel id 2, Current position 3 and Diameter 10
End of Iteration
Wheel id 3, Current position 1 and Diameter 2
Wheel id 1, Current position 2 and Diameter 9
Wheel id 2, Current position 3 and Diameter 10
Wheel id 4, Current position 4 and Diameter 10
End of Iteration
Wheel id 3, Current position 1 and Diameter 2
Wheel id 5, Current position 2 and Diameter 7
Wheel id 1, Current position 3 and Diameter 9
Wheel id 2, Current position 4 and Diameter 10
Wheel id 4, Current position 5 and Diameter 10
End of Iteration
```

## See Also

`entry` | `eventIterate` | `getEntityPortsImpl` | `getEntityStorageImpl` | `getEntityTypesImpl` | `iterate` | `matlab.DiscreteEventSystem` | `matlab.System`

## More About

- "Integrate System Objects Using MATLAB System Block"
- "Create a Discrete-Event System Object" on page 9-44
- "Generate Code for MATLAB Discrete-Event System Blocks" on page 9-48
- "Call Simulink Function from a MATLAB Discrete-Event System Block" on page 9-55

# Custom Entity Storage Block with Multiple Timer Events

A discrete-event system allows the implementation of distinct event types for manipulating entities and storages. Sometimes, the desired behavior involves more than one event acting on the same storage or entity. This example shows how to handle multiple events acting on the same target in a discrete-event system framework. In this example, a custom entity storage block is generated to implement the `tag`, which is one of the identifiers, when multiple timer events are acting on the same entity. To see the list of event identifiers, see “Customize Discrete-Event System Behavior Using Events and Event Actions” on page 9-51.

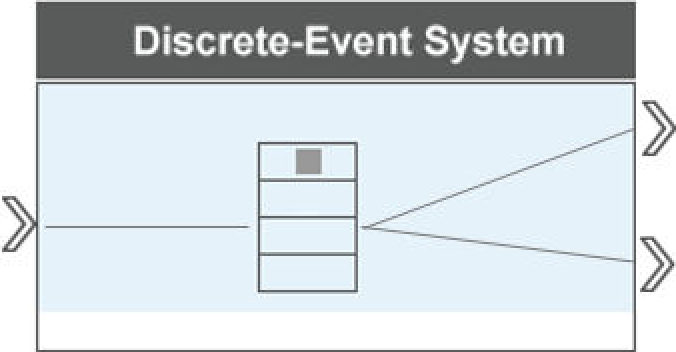
To open the model and observe the behavior of the custom block, see `CustomEntityStorageBlockWithTwoTimerEventsExample`.

## Create the Discrete-Event System Object with Multiple Timer Events

Suppose that the discrete-event System object is used to represent a facility that processes metal parts using an oven. The processing time varies based on the detected metal. For safety, the parts have a maximum allowed processing time.

- If the oven processing time is less than the allowed maximum time, the parts are processed and depart the oven and the facility.
- If there is an error in detected metal, the service time exceeds the maximum allowed processing time, the process stops and the parts are taken out of the oven to be rerouted for further processing.

To represent this behavior, this example uses a custom entity storage block with one input, two outputs, and a storage element. An entity of type `Part` with `TimeOut` attribute enters the storage of the custom block to be processed. `TimeOut` determines the maximum allowed processing time of the parts. When a part enters the storage, two timer events are activated. One timer tracks the processing time of the part in the oven. When this timer expires, the entity is forwarded to output `1`. Another timer acts as a fail-safe and tracks if the maximum allowed processing time is exceeded or not. When this timer expires, the process is terminated and the entity is forwarded to the output `2`.




This example that generates the custom block and uniquely identifies these two timer events targeting on the same entity using custom tags.

### See the Code to Generate the Custom Storage Block with Timer Events

```
classdef CustomEntityStorageBlockTimer < matlab.DiscreteEventSystem
```

```
    % A custom entity storage block with one input port, two output ports, and one storage.

    % Nontunable properties
    properties (Nontunable)
    % Capacity
        Capacity = 1;
    end

    methods (Access=protected)

        function num = getNumInputsImpl(~)
            num = 1;
        end

        function num = getNumOutputsImpl(~)
            num = 2;
        end

        function entityTypes = getEntityTypesImpl(obj)
            entityTypes = obj.entityType('Part');
        end

        function [inputTypes,outputTypes] = getEntityPortsImpl(obj)
            inputTypes = {'Part'};
            outputTypes = {'Part' 'Part'};
        end

        function [storageSpecs, I, O] = getEntityStorageImpl(obj)
            storageSpecs = obj.queueFIFO('Part', obj.Capacity);
            I = 1;
            O = [1 1];
        end

    end


   methods

        function [entity,event] = PartEntry(obj,storage,entity,source)
            % Specify event actions when entity enters storage.
             ProcessingTime=randi([1 15]);
             event1 = obj.eventTimer('TimeOut', entity.data.TimeOut);
             event2 = obj.eventTimer('ProcessComplete', ProcessingTime);
             event = [event1 event2];
        end

        function [entity, event] = timer(obj,storage,entity,tag)
            % Specify event actions for when scheduled timer completes.
            event = obj.initEventArray;
            switch tag
                case 'ProcessComplete'
                    event = obj.eventForward('output', 1, 0);
                case 'TimeOut'
                    event = obj.eventForward('output', 2, 0);
            end

        end

    end

end
```

## Custom Block Behavior

1 Generate a custom block with one input, two outputs, and a storage element. For more information about creating a basic storage element, see “Implement a Discrete-Event System Object with MATLAB Discrete-Event System Block” on page 9-6.

```
function num = getNumInputsImpl(~)
    num = 1;
end

function num = getNumOutputsImpl(~)
    num = 2;
end

function entityTypes = getEntityTypesImpl(obj)
    entityTypes = obj.entityType('Part');
```

```
    end

    function [inputTypes,outputTypes] = getEntityPortsImpl(obj)
        inputTypes = {'Part'};
        outputTypes = {'Part' 'Part'};
    end

    function [storageSpecs, I, O] = getEntityStorageImpl(obj)
        storageSpecs = obj.queueFIFO('entity1', obj.Capacity);
        I = 1;
        O = [1 1];
    end
```

**2** Invoke two timers with tags `'TimeOut'` and `'ProcessComplete'` when an entity enters the storage.

```
    function [entity,event] = PartEntry(obj,storage,entity,source)
        % Specify event actions when entity enters storage.
        ProcessingTime = randi([1 15]);
        % The TimeOut attribute specifies the expiration time of the timer with tag TimeOut
        event1 = obj.eventTimer('TimeOut', entity.data.TimeOut);
        % The expiration time of the timer ProcessComplete is a random integer between
        % 1 and 15.
        event2 = obj.eventTimer('ProcessComplete', ProcessingTime);
        event = [event1 event2];
    end
```

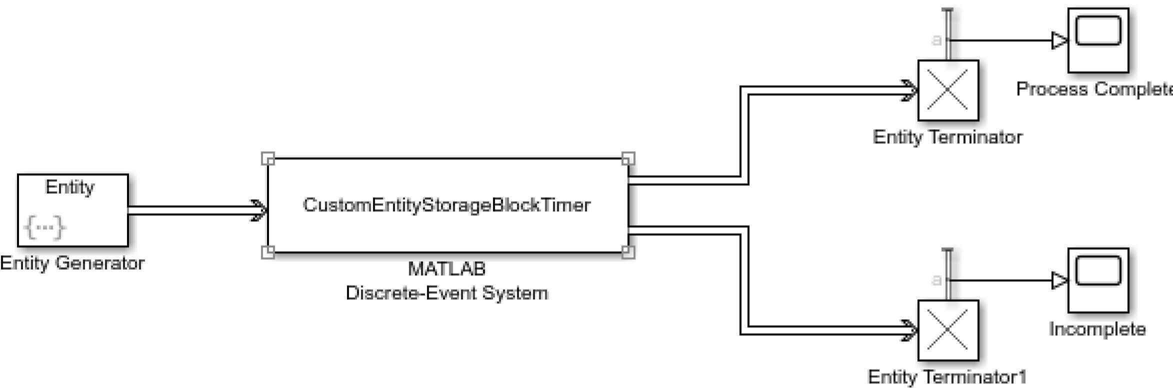
**3** The timer that expires the first determines the entity forward behavior.

```
    function [entity, event] = timer(obj,storage,entity,tag)
        % Specify event actions for when scheduled timer completes.
        event = obj.initEventArray;
            switch tag
                case 'ProcessComplete'
                    % If ProcessComplete expires first, entities are forwarded to output 1.
                    event = obj.eventForward('output', 1, 0);
                case 'TimeOut'
                    % If TimeOut expires first, entities are forwarded to output 2.
                    event = obj.eventForward('output', 2, 0);
            end
    end
```

## Implement Custom Block

**1** Save the `.m` file as `CustomEntityStorageBlockTimer`. Link the System object to a SimEvents model by using a MATLAB Discrete-Event System block. For more information about linking, see "Create Custom Blocks Using MATLAB Discrete-Event System Block" on page 9-2.

**2** Create a SimEvents model including the MATLAB Discrete-Event System block, an Entity Generator block, two Entity Terminator blocks. Connect the blocks as shown in the model.

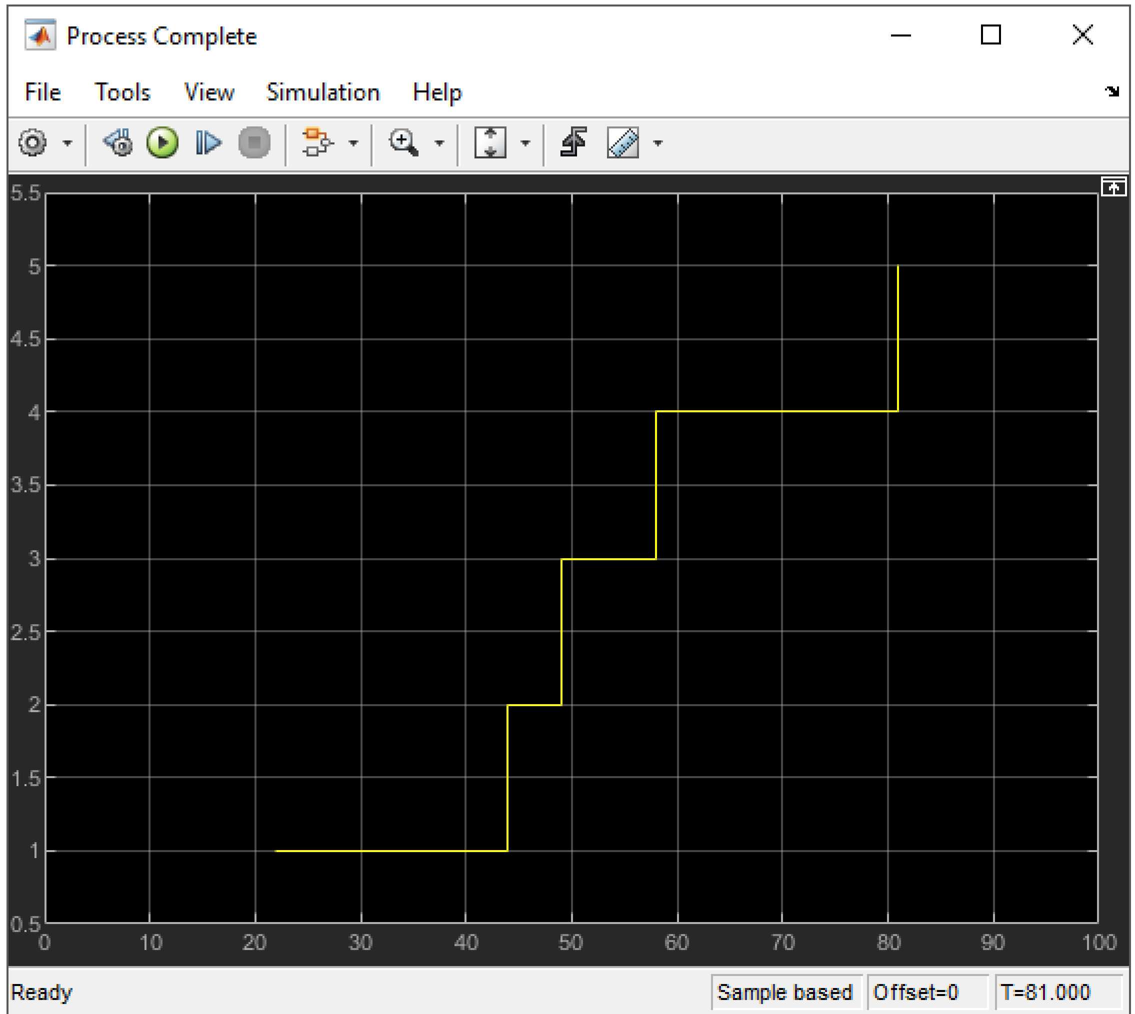


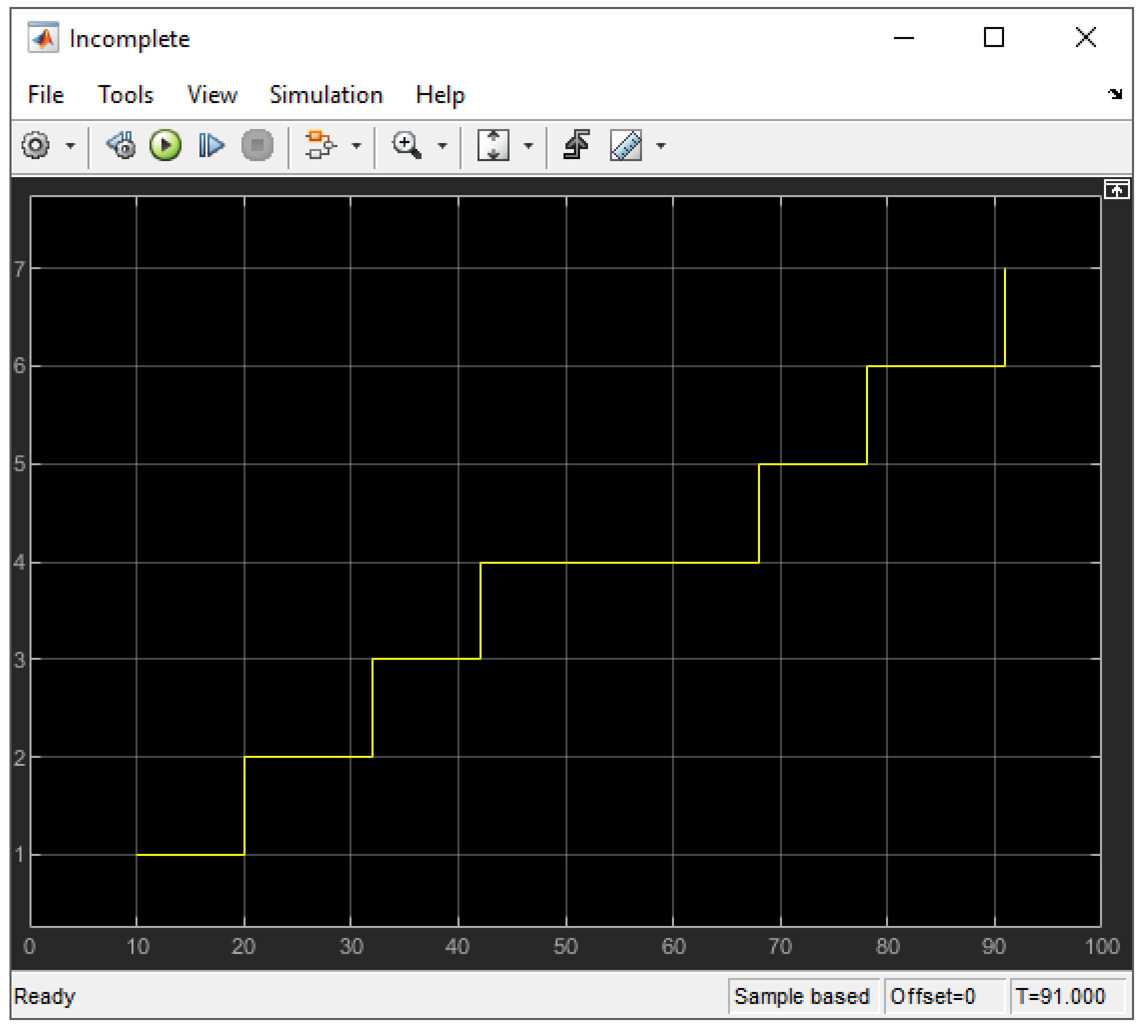

**3** In the Entity Generator block:

a. In the **Entity type** tab, set the **Attribute Name** as `TimeOut`.
b. In the **Event actions** tab, in the **Generate action** field:

```
entity.TimeOut = 10;
```

4. In the Entity Terminator and Entity Terminator1 blocks, output the **Number of entities arrived, a** statistic and connect them to scopes.
5. Increase simulation time to `100` and simulate the model. Observe that entities are forwarded to the corresponding output based on the corresponding timer expiration.






## See Also

`entry` | `eventTimer` | `getEntityPortsImpl` | `getEntityStorageImpl` | `getEntityTypesImpl` | `matlab.DiscreteEventSystem` | `matlab.System` | `timer`

## More About

- "Integrate System Objects Using MATLAB System Block"
- "Create a Discrete-Event System Object" on page 9-44
- "Generate Code for MATLAB Discrete-Event System Blocks" on page 9-48
- "Call Simulink Function from a MATLAB Discrete-Event System Block" on page 9-55

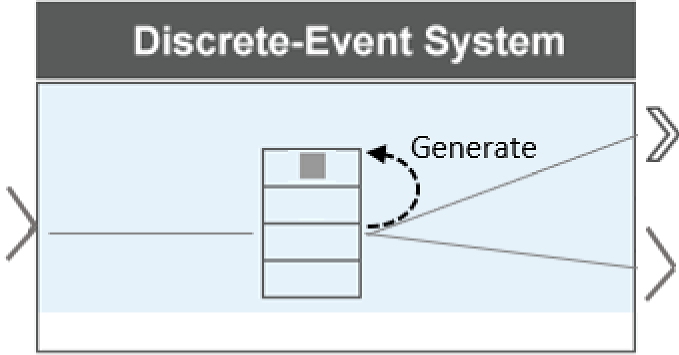
# Custom Entity Generator Block with Signal Input and Signal Output

This example shows how to create a custom source block that generates entities and to manage discrete states when implementing the discrete-event System object methods.

Suppose that you manage a facility that produces raw materials with a fixed deterministic rate. The materials contain a 12-digit bar code for stock management and priority values for order prioritization. To represent this behavior, this example shows how to generate a custom entity storage block is generated with one signal input port, one entity output port, one signal output port, and one storage element. The block generates entities with distinct priority values. The entities carry data and depart the block from its output port. The entity priority values are acquired from values of the incoming signal.

To open the model and to observe the behavior of the custom block, see `CustomEntityGeneratorBlockExample`.

## Create the Discrete-Event System Object




The block is defined as a custom entity generator block that generates entities with specified intergeneration periods. The generated entities carry data, and their priority values are determined by the values of the input signal.

### See the Code to Create the Custom Entity Generator Block

```
classdef CustomEntityStorageBlockGeneration < matlab.DiscreteEventSystem...

    % A custom entity generator block.

    % Nontunable properties
    properties (Nontunable)
        % Generation period
        period = 1;
    end

    properties(DiscreteState)
        % Entity priority
        priority;
        % Entity value
        value;
    end

    % Discrete-event algorithms
    methods
```

```
        function [events, out1] = setupEvents(obj)
            % Set up entity generation events at simulation start.
            events = obj.eventGenerate(1,'mygen',obj.period,obj.priority);
            % Set up the initial value of the output signal.
            out1 = 10;
        end

        function [entity,events,out1] = generate(obj,storage,entity,tag,in1)
            % Specify event actions when entity is generated in storage.
            entity.data = obj.value;
            % The priority value is assigned from the input signal.
            obj.priority = in1;
            % Output signal is the assigned priority value.
            out1 = obj.priority;
            events = [obj.eventForward('output',1,0) ...
                      obj.eventGenerate(1,'mygen',obj.period,obj.priority)];
        end
    end

    methods(Access = protected)

        function entityTypes = getEntityTypesImpl(obj)
            entityTypes = obj.entityType('Material');
        end

        function [inputTypes,outputTypes] = getEntityPortsImpl(obj)
            % Specify entity input and output ports. Return entity types at
            % a port as strings in a cell array. Use empty string to
            % indicate a data port.
            inputTypes = {''};
            outputTypes = {'Material',''};
        end

        function resetImpl(obj)
            % Initialize / reset discrete-state properties.
            obj.priority = 10;
            obj.value = 1:12;
        end

        function [storageSpecs, I, O] = getEntityStorageImpl(obj)
            storageSpecs = obj.queueFIFO('Material', 1);
            I = 0;
            O = [1 0];
        end

        function num = getNumInputsImpl(obj)
            % Define total number of inputs for system with optional
            % inputs.
            num = 1;
        end

        function num = getNumOutputsImpl(~)
            % Define total number of outputs.
            num = 2;
        end
        function [out1 out2] = getOutputSizeImpl(obj)
            % Return size for each output port.
            out1 = [1 12];
            out2 = 1;
        end

        function [out1 out2] = getOutputDataTypeImpl(obj)
            % Return data type for each output port.
            out1 = "double";
            out2 = "double";
        end

        function [out1 out2] = isOutputComplexImpl(obj)
            % Return true for each output port with complex data.
            out1 = false;
            out2 = false;
        end

        function [sz,dt,cp] = getDiscreteStateSpecificationImpl(obj,name)
            % Return size, data type, and complexity of discrete-state
            % specified in name.
            switch name
                case 'priority'
                    sz = [1 1];
                case 'value'
                    sz = [1 12];
```

```
            end
            dt = "double";
            cp = false;
        end
    end
end
```

## Custom Block Behavior

1. Define the time between material generations.

   ```
   % Nontunable properties
   properties (Nontunable)
       % Generation period
       period = 1;
   end
   ```
2. Initialize the discrete state variables.

   ```
   function resetImpl(obj)
       % Initialize / reset discrete-state properties.
       obj.priority = 10;
       obj.value = 1:12;
   end
   ```

   The variable `priority` represents material priority and the `value` represents bar code data carried by the materials.
3. Initialize the output for a source block.

   ```
   function num = getNumOutputsImpl(~)
       % Define total number of outputs.
       num = 2;
   end
   function [out1 out2] = getOutputSizeImpl(obj)
   % Return size for each output port.
       out1 = [1 12];
       out2 = 1;
   end

   function [out1 out2] = getOutputDataTypeImpl(obj)
       % Return data type for each output port.
       out1 = "double";
       out2 = "double";
   end

   function [out1 out2] = isOutputComplexImpl(obj)
       % Return true for each output port with complex data.
       out1 = false;
       out2 = false;
   end
   ```

   - First function declares the output size.
   - Second function declares that output port data types are `double`.
   - Third function declares `false` for output ports because they do not support complex data.
4. Declare the size, data, and complexity of the discrete states.

   ```
   function [sz,dt,cp] = getDiscreteStateSpecificationImpl(obj,name)
       % Return size, data type, and complexity of discrete-state.
           switch name
               case 'priority'
                   sz = [1 1];
               case 'value'
                   sz = [1 12];
           end
           dt = "double";
           cp = false;
   end
   ```

   - The discrete state `priority` is scalar. The data type is `double` and takes real values.
   - The discrete state `value` is a `1`-by-`12` vector. The data type is `double` and takes real values.

**5** Generate the materials with intergeneration period, priority, and data defined by:

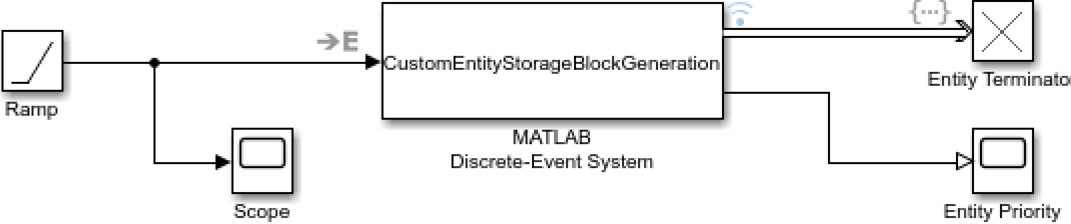
- The parameter `obj.period`, declared as a public parameter that can be changed from the block dialog box.
- The parameter `obj.priority` values, defined by the signal from the input port.
- The parameter `obj.value`, a `1`-by-`12` vector which represents the data carried by entities.

```
function events = setupEvents(obj)
    % Set up entity generation event for storage 1 at simulation start.
    events = obj.eventGenerate(1,'mygen',obj.period,obj.priority);
    % Set up the initial value of the output signal.
    out1 = 10;
end

function [entity,events,out1] = generate(obj,storage,entity,tag,in1)
    % Specify event actions when entity is generated in storage.
    entity.data = obj.value;
    % The value from the signal is assigned to the entity priority.
    obj.priority = in1;
    % Output signal is the assigned priority value.
    out1 = obj.priority;
    events = [obj.eventForward('output',1,0) ...
                obj.eventGenerate(1,'mygen',obj.period,obj.priority)];
end
```

## Implement Custom Block

**1** Save the `.m` file as `CustomEntityStorageBlockGeneration`. Link the System object to a SimEvents model by using a MATLAB Discrete-Event System block. For more information about linking, see “Create Custom Blocks Using MATLAB Discrete-Event System Block” on page 9-2.

**2** Create a SimEvents model that includes the MATLAB Discrete-Event System block, a Ramp block, an Entity Terminator block, and two Scope blocks. Connect the blocks as shown in the model.




**3** In the Ramp block, set **Slope** to `5` and **Initial output** to `10`.

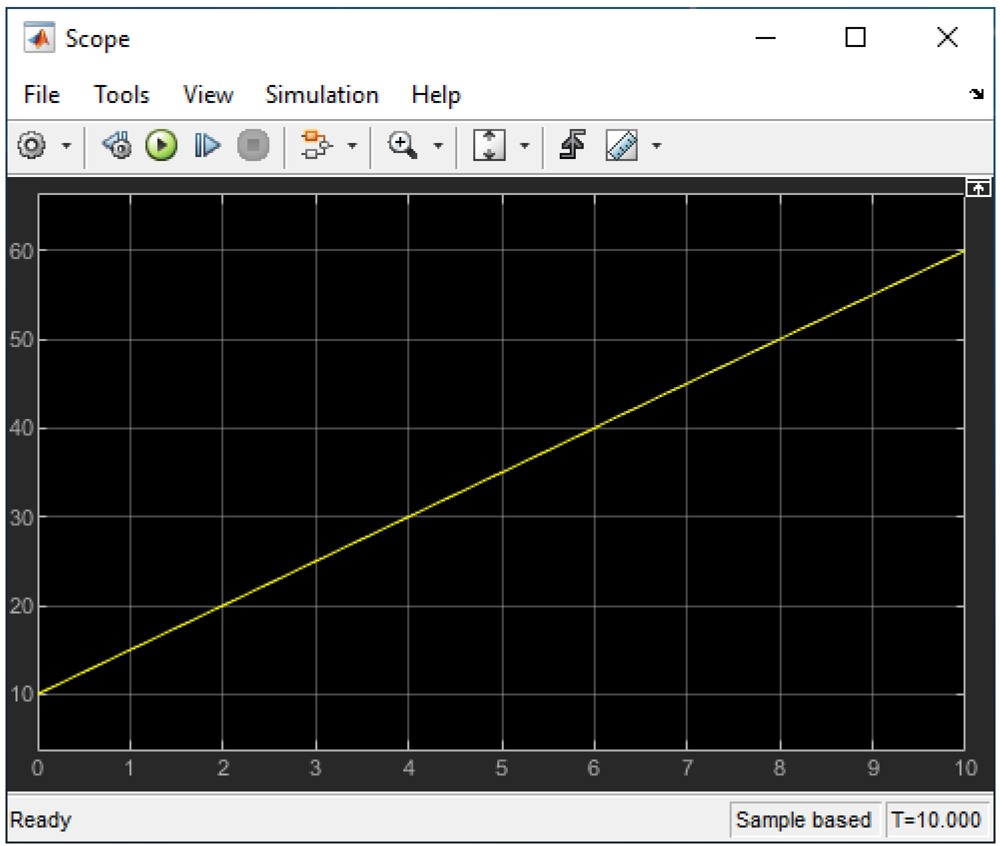
**4** In the Entity Terminator block, you can display the priority values of the entities arriving at the block, in the **Entry action** field enter this code.

```
coder.extrinsic('fprintf');
fprintf('Priority: %d\n', double(entitySys.priority))
```

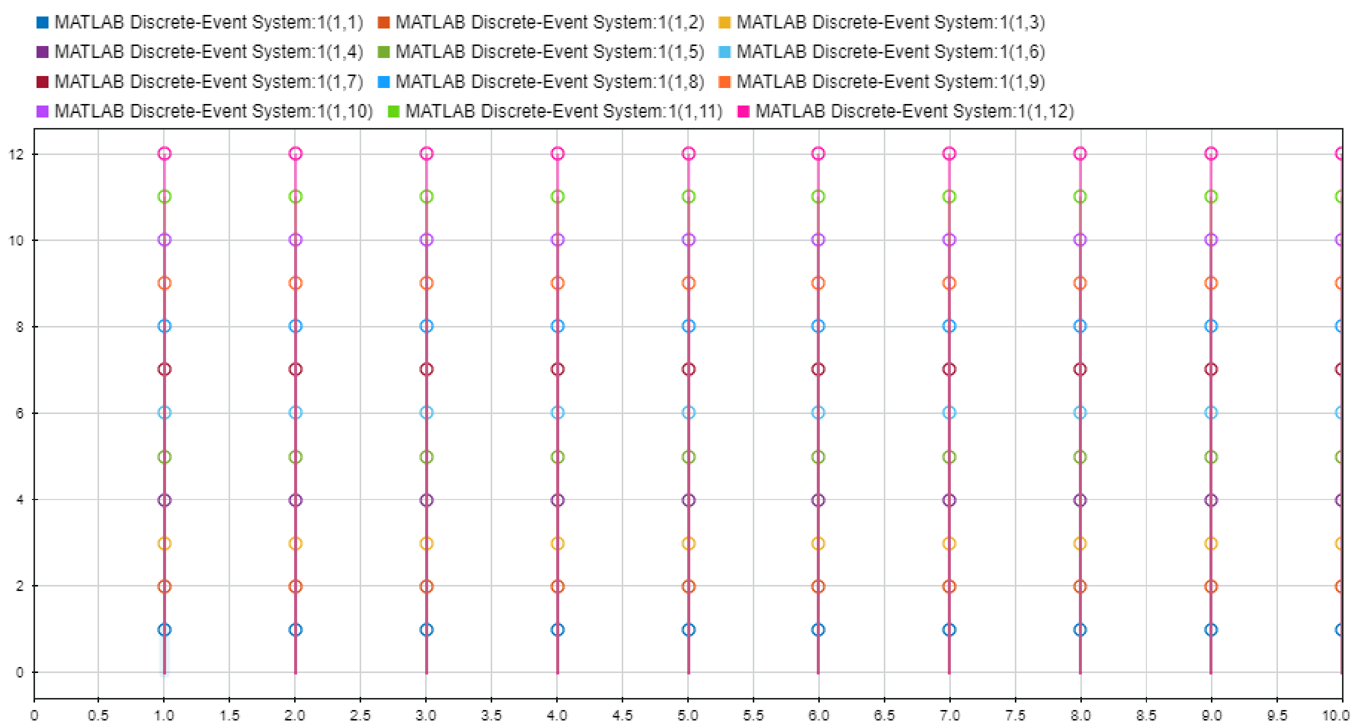
**5** Right-click the entity path from the custom Entity Generator to the Entity Terminator and select the **Log Selected Signals**.

**6** Simulate the model.

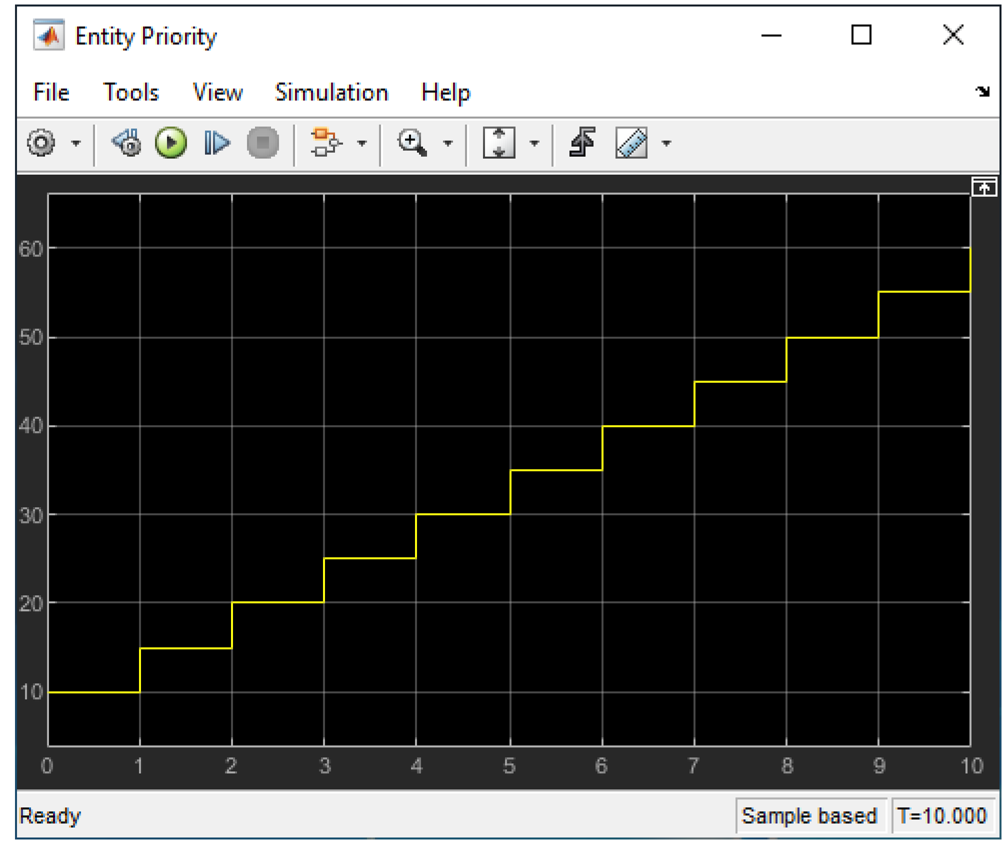
   **a** Observe the output of the Ramp block. For instance, the output value becomes `15`, `20`, `25`, and `30` for the simulation time `1`, `2`, `3`, and `4`, respectively.




**b** The Simulation Data Inspector shows that entities are forwarded to the Entity Terminator block with data of size 1-by-12.




**c** You can also observe the priority values from the scope labeled Entity Priority for generation times `1`, `2`,`3`, `4`, `5`, `6`, `7`, `8`, `9`, and `10`.




## See Also

`entry` | `generate` | `getEntityPortsImpl` | `getEntityStorageImpl` | `matlab.DiscreteEventSystem` | `matlab.System`

## More About

- "Integrate System Objects Using MATLAB System Block"
- "Create a Discrete-Event System Object" on page 9-44
- "Generate Code for MATLAB Discrete-Event System Blocks" on page 9-48
- "Call Simulink Function from a MATLAB Discrete-Event System Block" on page 9-55

# Build a Custom Block with Multiple Storages

This example shows how to create a custom block with multiple storages and manage storage behavior using discrete-event System object methods.

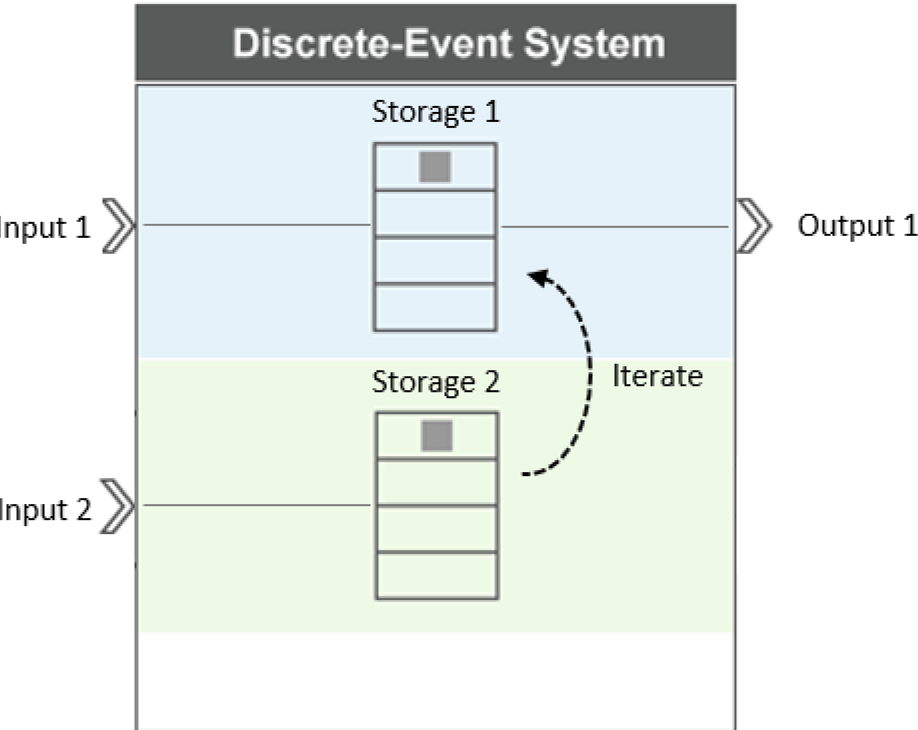
Suppose that you manage a facility that produces items for customer orders. To prepare for repetitive orders, the facility produces a supply of items before the orders arrive. When a new order arrives, the stocks are checked for availability.

- If the item is found in the storage, it departs the facility to fulfill the order.
- If the item is not found in the storage, a new item is produced and the generated item departs the facility to fulfill the order.

To generate this custom behavior, you manipulate multiple storages through a discrete-event System object, created using the `matlab.DiscreteEventSystem` methods. To observe the behavior of the custom block, see `CustomEntityStorageBlockWithTwoStoragesExample`.

## Create the Discrete-Event System Object

Generate a custom entity storage block with two inputs, one output, and two storage elements.




The desired behavior of the custom block is to select and output entities based on a reference entity.

1. Input port 1 accepts entities with type `Item` to storage 1.
2. Input port 2 accepts reference entities with type `Order` to storage 2.
3. When a reference `Order` arrives at storage 2, its attribute data is recorded as the reference value, and the entity is destroyed.

4. The `Order` arrival invokes an iteration event at storage `1` to search for an `Item` carrying data that is equal to the reference value.
5. If a match is found, the matching item is forwarded to output port `1` and the iteration ends.
6. If the match is not found, a new `Item` is generated at storage `1` with a matching attribute value and forwarded to output port `1`.

### See the Code to Generate the Custom Storage Block with Multiple Storages

```
classdef CustomBlockTwoEntityStorages < matlab.DiscreteEventSystem

    % Select from stored entities based on a lookup key.

    properties (Nontunable)
        % Capacity
        capacity = 100;
    end

    properties (DiscreteState)
        InputKey;
    end

   methods (Access=protected)

        function num = getNumInputsImpl(~)
            num = 2;
        end

        function num = getNumOutputsImpl(~)
            num = 1;
        end

        function [entityTypes] = getEntityTypesImpl(obj)
            entityTypes = [obj.entityType('Item'), ...
                           obj.entityType('Order')];
        end

        function [inputTypes, outputTypes] = getEntityPortsImpl(~)
            inputTypes = {'Item' 'Order'};
            outputTypes = {'Item'};
        end

        function [storageSpecs, I, O] = getEntityStorageImpl(obj)
            storageSpecs = [obj.queueFIFO('Item', obj.capacity)...
                obj.queueFIFO('Order', obj.capacity)];
            I = [1 2];
            O = 1;
        end

        function [sz, dt, cp] = getDiscreteStateSpecificationImpl(obj, name)
            sz = 1;
            dt = 'double';
            cp = false;
        end

        function resetImpl(obj)
            obj.InputKey = 0;
        end
    end

    methods

        function [Order,events] = OrderEntry(obj, storage, Order, source)
            % A key entity has arrived; record the Inputkey value.
            obj.InputKey = Order.data.Key;
            % Schedule an iteration of the entities in storage 1.
            % Destroy input key entity.
            events = [obj.eventIterate(1, '') ...
                      obj.eventDestroy()];
            coder.extrinsic('fprintf');
            fprintf('Order Key Value: %f\n', Order.data.Key);
        end

        function [Item,events,continueIter] = ItemIterate(obj,...
                                                  storage, Item, tag, cur)
            % Find entities with matching key.
            events = obj.initEventArray;
            continueIter = true;
```

```
            if (Item.data.Attribute1 == obj.InputKey)
                events = obj.eventForward('output', 1, 0.0);
                % If a match is found, the iteration ends and the state is reset.
                continueIter = false;

            elseif cur.size == cur.position
                % If a match is not found, a new matching entity is generated.
                events = obj.eventGenerate(1,'mygen',0.0,100);
            end
        end

        function [Item,events] = ItemGenerate(obj,storage,Item,tag)
            % Specify event actions when entity generated in the storage.
            Item.data.Attribute1 = obj.InputKey;
            events = obj.eventForward('output',1,0.0);
        end
    end
end
```

## Custom Block Behavior

1 Discrete state variable `InputKey` represents the recorded reference value from `Order`, which is used to select corresponding `Item`.

```
properties (DiscreteState)
    InputKey;
end
```

2 The block has two storages with FIFO behavior. Storage `1` supports entities with type `Item`, and storage `2` supports entities with type `Order`. The block has two input ports and one output port. Input port `1` and output port `1` are connected to storage `1`. Input port `2` is connected to storage `2`. For more information about declaring ports and storages, see “Implement a Discrete-Event System Object with MATLAB Discrete-Event System Block” on page 9-6.

```
function num = getNumInputsImpl(~)
    num = 2;
end

function num = getNumOutputsImpl(~)
    num = 1;
end

function [entityTypes] = getEntityTypesImpl(obj)
    entityTypes = [obj.entityType('Item'), ...
                   obj.entityType('Order')];
end

function [inputTypes, outputTypes] = getEntityPortsImpl(~)
    inputTypes = {'Item' 'Order'};
    outputTypes = {'Order'};
end

function [storageSpecs, I, O] = getEntityStorageImpl(obj)
    storageSpecs = [obj.queueFIFO('Item', obj.capacity)...
        obj.queueFIFO('Order', obj.capacity)];
        I = [1 2];
        O = 1;
end
```

3 Specify the discrete state and reset the state `InputKey`. For more information about states in discrete-event systems, see “Custom Entity Generator Block with Signal Input and Signal Output” on page 9-24.

```
function [sz, dt, cp] = getDiscreteStateSpecificationImpl(obj, name)
    sz = 1;
    dt = 'double';
    cp = false;
end

function resetImpl(obj)
    obj.InputKey = 0;
end
```

**4** When `Order` arrives at storage 2, its data `Key` is recorded in the discrete state variable `Obj.InputKey`. This entry also invokes an iteration event at storage `1` and another event to destroy `Order`.

```
function [Order, events] = OrderEntry(obj, storage, Order, source)
    % A key entity has arrived; record the Inputkey value.
    obj.InputKey = Order.data.Key;
    % Schedule an iteration of the entities in storage 1.
    % Destroy input key entity.
    events = [obj.eventIterate(1, '') ...
              obj.eventDestroy()];
    coder.extrinsic('fprintf');
    fprintf('Order Key Value: %f\n', Order.data.Key);
end
```

**5** The purpose of the iteration is to find items with data that matches `InputKey`.

```
function [Item,events,continueIter] = ItemIterate(obj,...
                                          storage, Item, tag, cur)
    % Find entities with matching key.
    events = obj.initEventArray;
    continueIter = true;

    if (Item.data.Attribute1 == obj.InputKey)
        events = obj.eventForward('output', 1, 0.0);
        % If a match is found, the iteration ends and the state is reset.
        continueIter = false;

    elseif cur.size == cur.position
        % If a match is not found, this invokes an entity generation event.
        events = obj.eventGenerate(1,'mygen',0.0,100);
    end
end
```

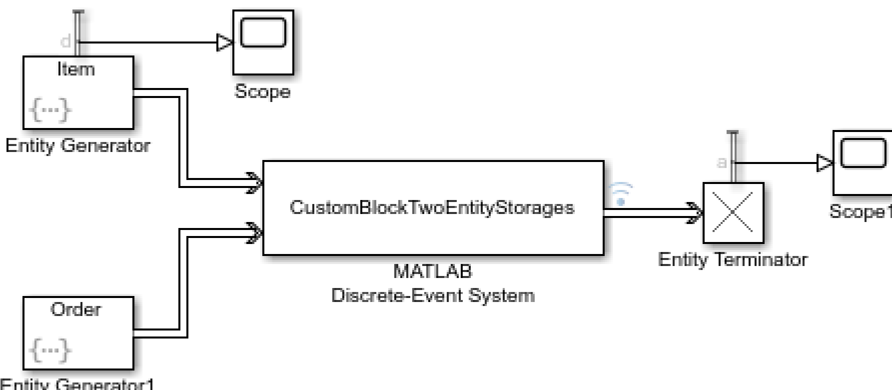
**6** Generate an entity with type `entity1` and a matching `Key` value. Then, forward the generated entity to output port `1`.

```
function [Item,events] = ItemGenerate(obj,storage,Item,tag)
    % Specify event actions when entity generated in the storage.
    Item.data.Attribute1 = obj.InputKey;
    events = obj.eventForward('output',1,0.0);
end
```

## Implement the Custom Block

**1** Save the `.m` file as `CustomBlockTwoEntityStorages`. Link the System object to a SimEvents model using a MATLAB Discrete-Event System block. For more information about linking, see “Create Custom Blocks Using MATLAB Discrete-Event System Block” on page 9-2.

**2** Create a SimEvents model including the MATLAB Discrete-Event System block, two Entity Generator blocks, and an Entity Terminator block. Connect the blocks as shown in the model.



3. In the Entity Generator block:

   a. In the **Entity generation** tab, set the **Generate entity at simulation start** to `off`.
   b. In the **Entity type** tab, set the **Entity type name** as `Item`.
   c. In the **Event Actions** tab, in the **Generate action field** enter:

      ```
      entity.Attribute1 = randi([1 3]);
      ```

      By default, the entities are generated with intergeneration time `1` and their `Attribute1` value is a random integer between `1` and `3`.
   d. In the **Statistics** tab, output the **Number of entities departed, d** statistic and connect it to a scope.
4. In the Entity Generator1 block:

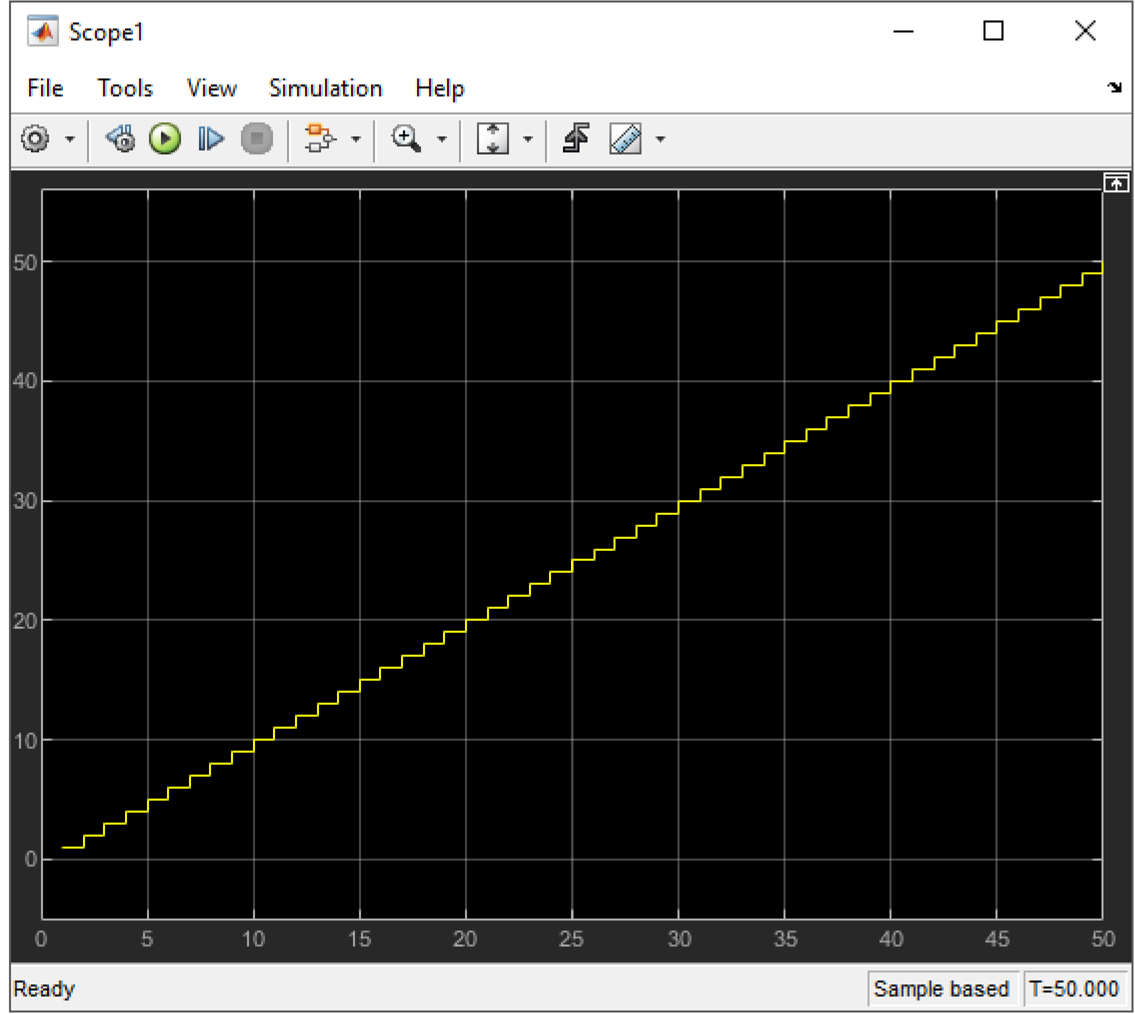
   a. In the **Entity generation** tab, set **Generate entity at simulation start** to `off`, and set **Period** to `5`.
   b. In the **Entity type** tab, set the **Entity type name** as `Order` and **Attribute Name** as `Key`.
   c. In the **Event Actions** tab, in the **Generate action field** enter:

      ```
      entity.Key = randi([1 4]);
      ```

   Entities with type `Order` are generated with intergeneration time `5`, and the `Key` attribute takes integer values between `1` and `4`.

   There is no possible match between `Key` and `Attribute1` when the `Key` value is `4` because `Attribute1` can take the value `1`, `2`, or `3`.
5. In the Entity Terminator block, output the **Number of entities arrived, a** statistic and connect it to a scope.
6. Right-click the entity path from the MATLAB Discrete-Event System block to the Entity Terminator block and select **Log Selected Signals**.
7. Increase simulation time to `50` and simulate the model. Observe that:

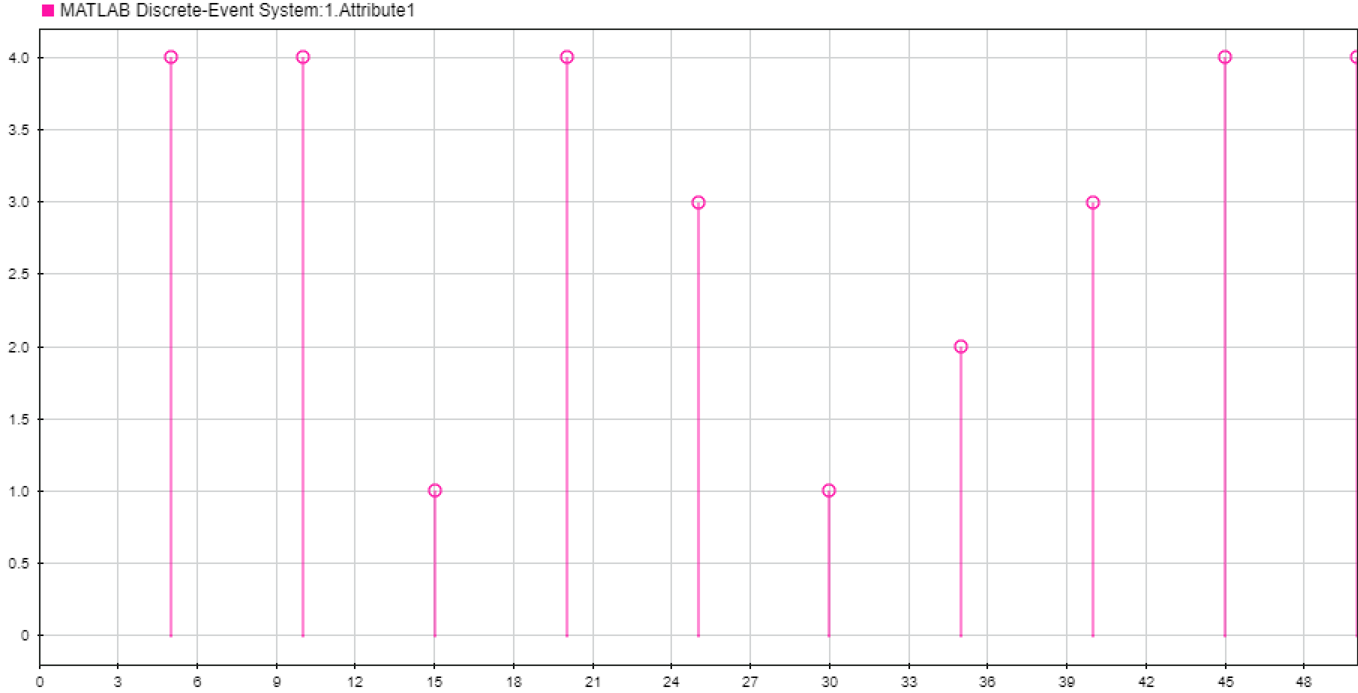
   a. `50` entities with type `Entity1` enter storage 1 in the block.




**b** In the Diagnostic Viewer, observe the incoming `Key` reference values carried by `10` entities that enter storage 2 and are destroyed afterward.

```
Order Key Value: 4.000000
Order Key Value: 4.000000
Order Key Value: 1.000000
Order Key Value: 4.000000
Order Key Value: 3.000000
Order Key Value: 1.000000
Order Key Value: 2.000000
Order Key Value: 3.000000
Order Key Value: 4.000000
Order Key Value: 4.000000
```

**c** The Simulation Data Inspector shows the departing items and their `Attribute1` values. The values match the `Key` values displayed in the Diagnostic Viewer.




Also observe `5` entities departing with `Attribute1` value `4`. These entities are generated in storage 2 because `Attribute1` cannot have the value `4` for the entities generated by the Entity Generator block.

## See Also

`entry` | `generate` | `getEntityPortsImpl` | `getEntityStorageImpl` | `iterate` | `matlab.DiscreteEventSystem` | `matlab.System`

## More About

- "Integrate System Objects Using MATLAB System Block"
- "Create a Discrete-Event System Object" on page 9-44
- "Generate Code for MATLAB Discrete-Event System Blocks" on page 9-48
- "Call Simulink Function from a MATLAB Discrete-Event System Block" on page 9-55

# Create a Custom Resource Acquirer Block

This example shows how to use resource management methods to create a custom entity storage block in which entities acquire resources from specified Resource Pool blocks.

Suppose that you manage a facility that produces parts from two different materials, material `1` and material `2`, to fulfill orders. After a part is produced, it is evaluated for quality assurance.

Two testing methods for quality control are:

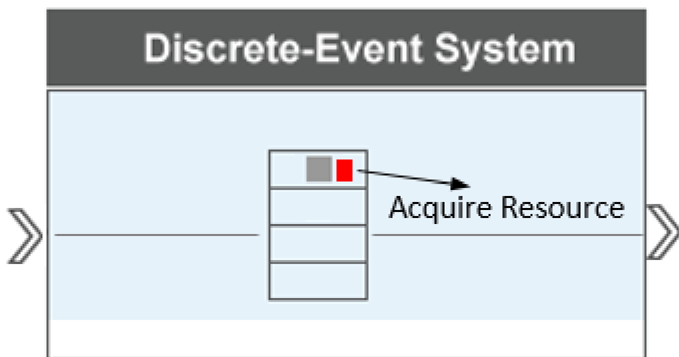
- Test 1 is used for parts that are produced from material `1`.
- Test 2 is used for parts that are produced from material `2`

After the production phase, parts are tagged based on their material to apply the correct test.

To generate the custom behavior, you create a discrete-event System object using the `matlab.DiscreteEventSystem` class methods for resource management.

## Create the Discrete-Event System Object

Generate a custom entity storage block with one input, one output, and one storage element.




The block accepts an entity of type `Part` to its storage with capacity `1`. The entity has an attribute `Test` to indicate the material from which the part is produced. Based on the value of the attribute, the entity acquires a resource from the specified Resource Pool block and departs the block to be tested.

### See the Code to Generate the Custom Block to Acquire Resources

```
classdef CustomBlockAcquireResources < matlab.DiscreteEventSystem
    % Custom resource acquire block example.

    methods(Access = protected)

        function num = getNumInputsImpl(obj)
            num = 1;
        end


        function num = getNumOutputsImpl(obj)
            num = 1;
        end
```

```
        function entityTypes = getEntityTypesImpl(obj)
            entityTypes(1) = obj.entityType('Part');
        end

        function [input, output] = getEntityPortsImpl(obj)
            input  = {'Part'};
            output = {'Part'};
        end

        function [storageSpec, I, O] = getEntityStorageImpl(obj)
            storageSpec(1) = obj.queueFIFO('Part', 1);
            I = 1;
            O = 1;
        end

        function resNames = getResourceNamesImpl(obj)
            % Define the names of the resources to be acquired.
            resNames = obj.resourceType('Part', {'Test1', 'Test2'}) ;
        end

    end

    methods

        function [entity,events] = entry(obj, storage, entity, source)
            % On entity entry, acquire a resource from the specified pool.
            if entity.data.Test == 1
            % If the entity is produced from Material1, request Test1.
            resReq = obj.resourceSpecification('Test1', 1);
            else
            % If the entity is produced from Material2, request Test2.
            resReq = obj.resourceSpecification('Test2', 1);
            end
            % Acquire the resource from the corresponding pool.
            events = obj.eventAcquireResource(resReq, 'TestTag');
        end

        function [entity,events] = resourceAcquired(obj, storage,...
                                       entity, resources, tag)
        % After the resource acquisition, forward the entity to the output.
            events = obj.eventForward('output', storage, 0.0);
        end

    end

end
```

## Custom Block Behavior

1 Define `Test1` and `Test2` type resources to be acquired by the entity type `Part`.

```
function resNames = getResourceNamesImpl(obj)
    % Define the names of the resources to be acquired.
    resNames = obj.resourceType('Part', {'Test1', 'Test2'}) ;
end
```

2 The entity enters the storage. If its `entity.data.Test` value is `1`, the entity is produced from `Material1`. The entity acquires `1` resource from the Resource Pool block with resources of type `Test1`. Similarly, If its `entity.data.Test` value is `2`, the entity acquires one resource from the Resource Pool block with resources of type `Test2`.

```
methods

    function [entity,events] = entry(obj, storage, entity, source)
        % On entity entry, acquire a resource from the specified pool.
        if entity.data.Test == 1
        % If the entity is produced from Material1, it acquires resource of type Test1.
        resReq = obj.resourceSpecification('Test1', 1);
        else
        % If the entity is produced from Material2, it acquires resource of type Test2.
        resReq = obj.resourceSpecification('Test2', 1);
        end
        % Acquire the resource from the corresponding pool.
        events = obj.eventAcquireResource(resReq, 'TestTag');
```

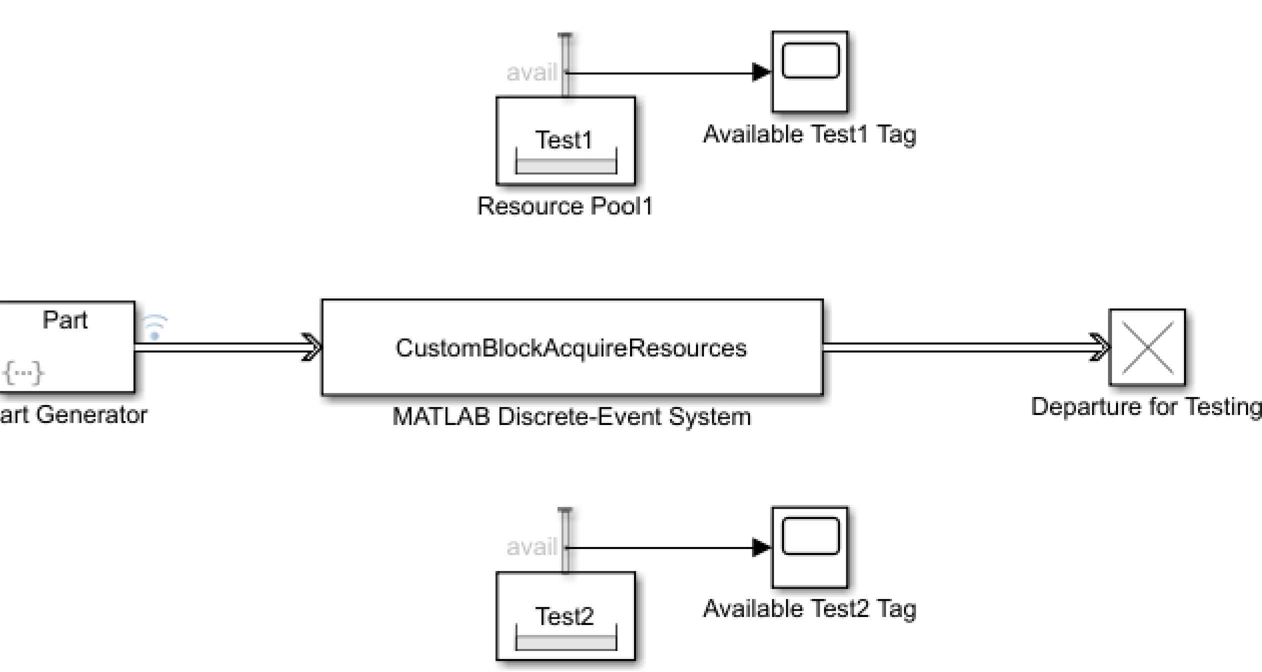
```
        end

        function [entity,events] = resourceAcquired(obj, storage,...
                                       entity, resources, tag)
            % After the resource acquisition, forward the entity to the output.
            events = obj.eventForward('output', storage, 0.0);
        end

    end
```

After the resource is successfully acquired, the `resourceAcquired` invokes the forwarding of the entity.

## Implement the Custom Block

1. Save the `.m` file as `CustomBlockAcquireResources`. Link the System object to a SimEvents model by using a MATLAB Discrete-Event System block. For more information about linking, see “Create Custom Blocks Using MATLAB Discrete-Event System Block” on page 9-2.
2. Create a SimEvents model using a MATLAB Discrete-Event System block, an Entity Generator block and an Entity Terminator block, and two Resource Pool blocks. Connect the blocks as shown in the diagram.

   Label Entity Generator block as Part Generator and Entity Terminator block as Departure for Testing.




3. In the Part Generator:

   a. In the **Entity generation** tab, set the **Generate entity at simulation start** to `off`.
   b. In the **Entity type** tab, set the **Entity type name** as `Part` and **Attribute Name** to `Test`.
   c. In the **Event Actions** tab, in the **Generate action field** enter:

   ```
   entity.Test= randi([1 2]);
   ```

Parts are generated with intergeneration time `1` and their `Test` attribute value is `1` or `2` to indicate the material type.

4. In the Resource Pool block:

   a. Set the **Resource name** to `Test1` and the **Reusable upon release** parameter to `off`.

   b. In the **Statistics** tab, output the **Amount available, avail** statistic and connect it to a scope.

5. In the Resource Pool1 block:

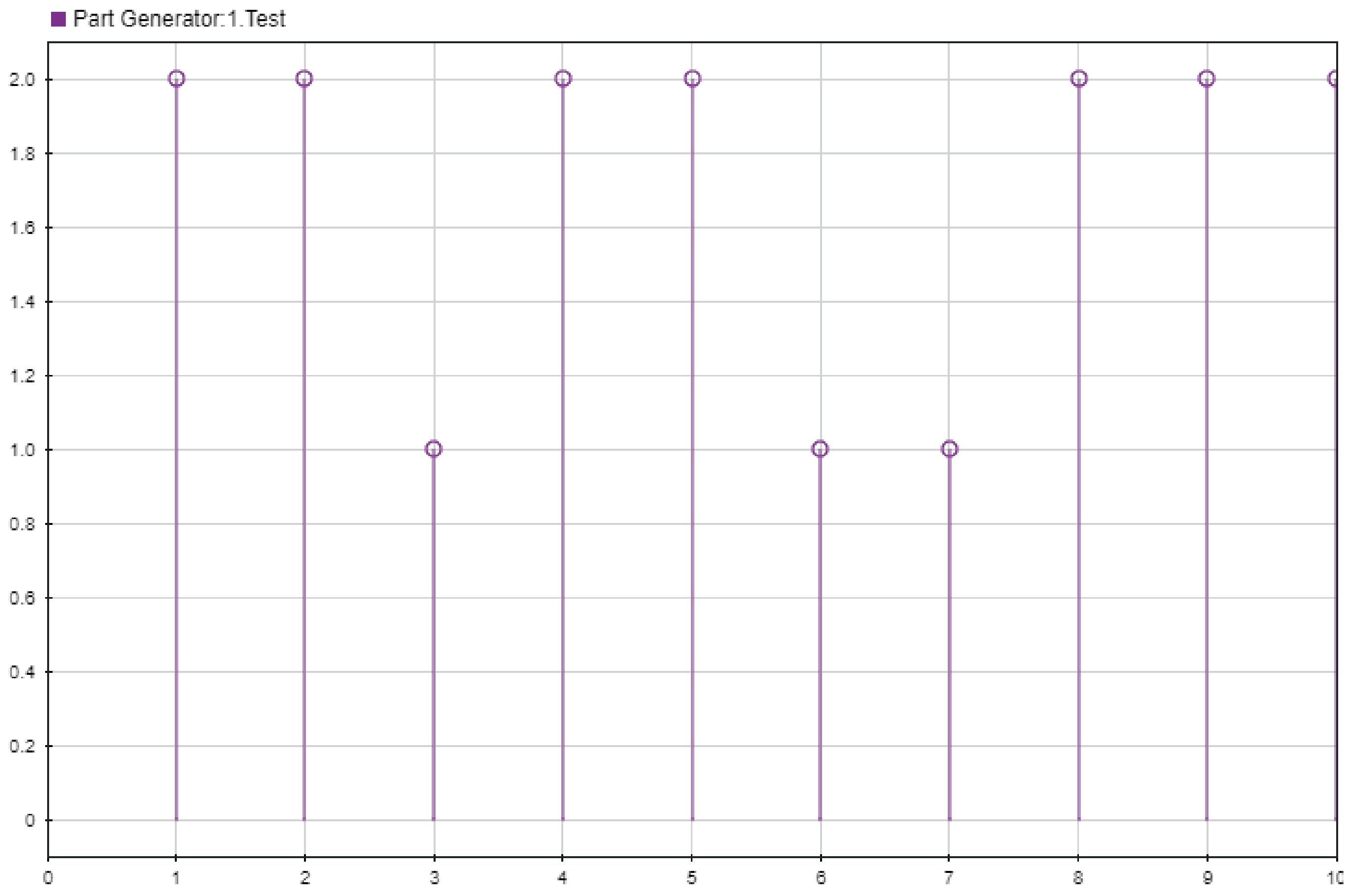
   a. Set the **Resource name** to `Test2` and the **Reusable upon release** parameter to `off`.

   b. In the **Statistics** tab, output the **Amount available, avail** statistic and connect it to a scope.

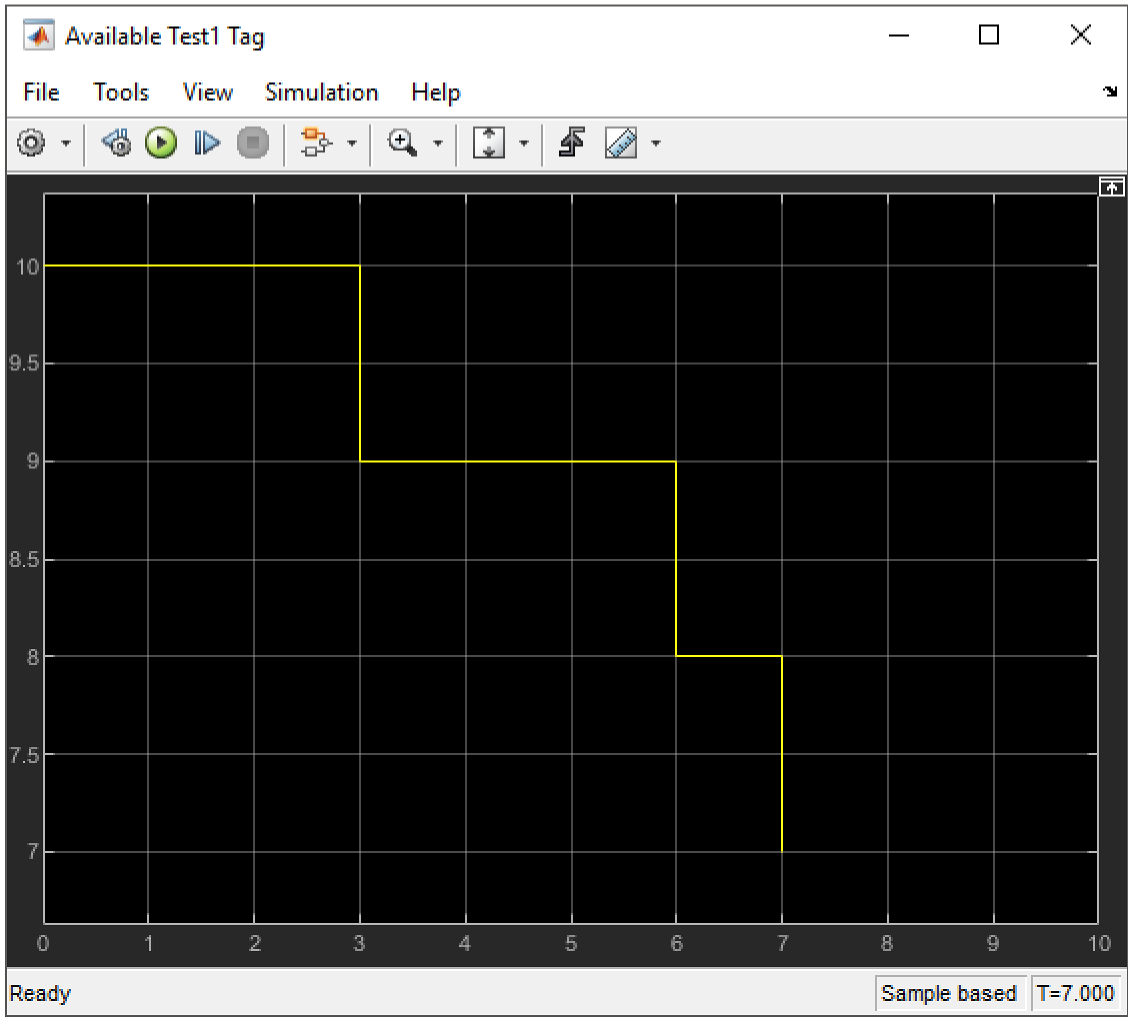
6. Right-click the entity path from Part Generator to the MATLAB Discrete-Event System block and select **Log Selected Signals**.

7. Simulate the model.

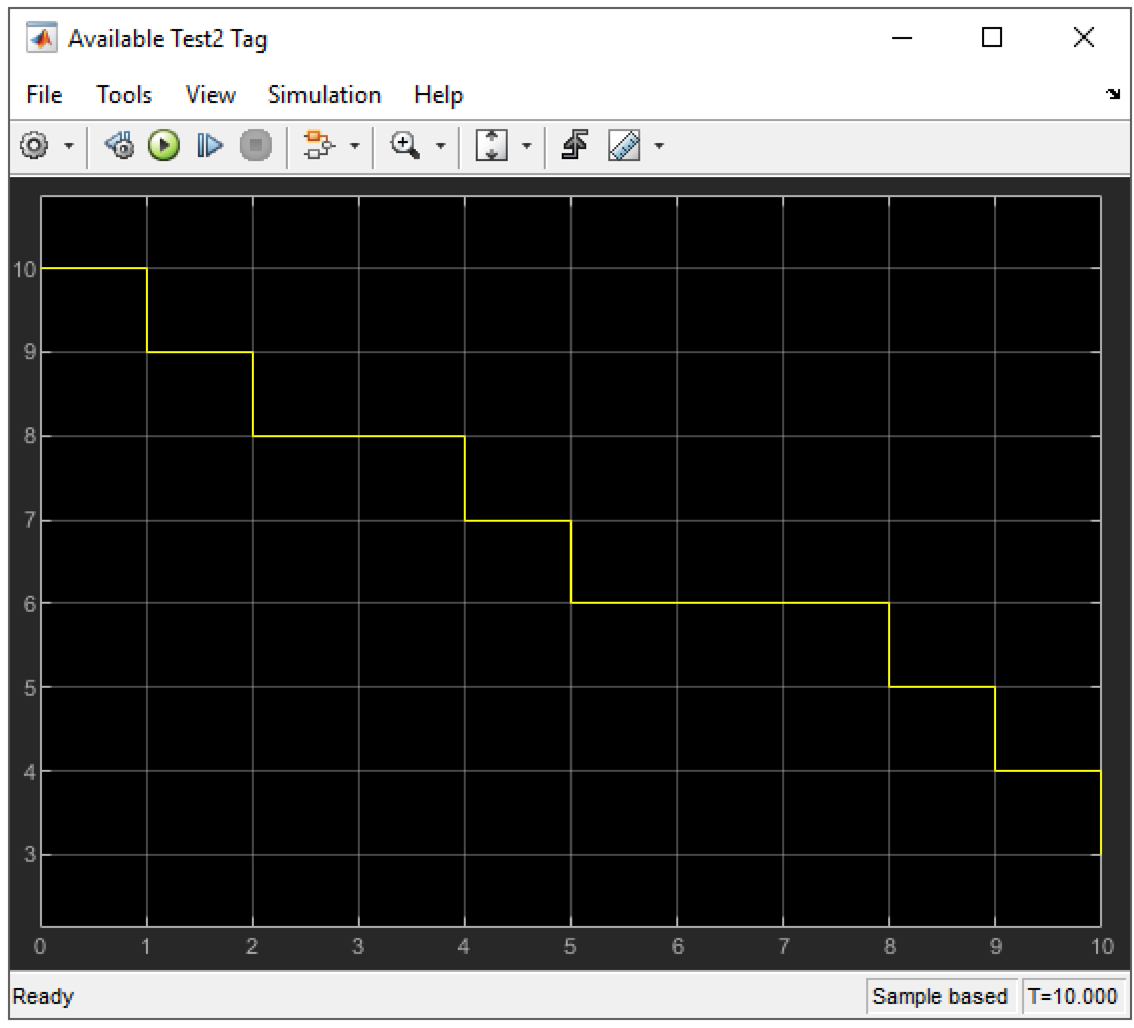
   - Observe the `Test` attribute values of the incoming entities to the custom block. Three entities require test 1 and seven entities requires test 2.




   - Observe that three resources of type `Test1` are acquired by entities.




- Observe that seven resources of type `Test2` are acquired by entities.




## See Also

`cancelAcquireResource` | `entry` | `eventAcquireResource` | `getResourceNamesImpl` | `matlab.DiscreteEventSystem` | `matlab.System` | `resourceAcquired` | `resourceSpecification`

## More About

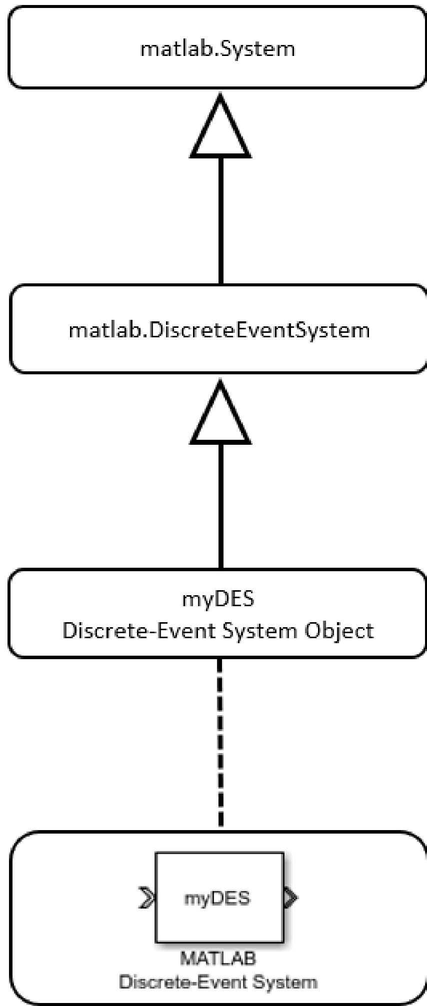
- "Integrate System Objects Using MATLAB System Block"
- "Create a Discrete-Event System Object" on page 9-44
- "Generate Code for MATLAB Discrete-Event System Blocks" on page 9-48
- "Call Simulink Function from a MATLAB Discrete-Event System Block" on page 9-55

# Create a Discrete-Event System Object

**In this section...**

"Methods" on page 9-44
"Inherited Methods from matlab.System Class" on page 9-46
"Reference and Extract Entities" on page 9-47

The MATLAB Discrete-Event System block allows you to author a custom discrete-event System object and use it in SimEvents models. To author event-driven entity-flow systems, the block uses discrete-event System object with the `matlab.DiscreteEventSystem` class, which inherits and extends the `matlab.System` class.




## Methods

The `matlab.DiscreteEventSystem` class provides methods that let you work with these elements of a discrete-event system:

- Define properties of the object entity types, ports, and storage
  - `getEntityPortsImpl` — Define input ports and output ports of discrete-event system
  - `getEntityStorageImpl` — Define entity storage elements of discrete-event system
  - `getEntityTypesImpl` — Define entity types of discrete-event system
- Event initialization
  - `setupEvents` — Initialize entity generation events
- Runtime behavior of the object
  - `blocked` — Event action when entity forward fails
  - `destroy` — Event action upon entity destruction
  - `entry` — Event action when entity enters storage element
  - `exit` — Event action before entity exit from storage
  - `generate` — Event action upon entity creation
  - `iterate` — Event action when entity iterates
  - `modified` — Event action upon entity modification by the Entity Find block
  - `resourceAcquired` — Specify event actions upon successful resource acquisition.
  - `resourceReleased` — Specify event actions upon successful resource release.
  - `testEntry` — Event action to accept or refuse entity
  - `timer` — Event action when timer completes

While implementing these methods, define entity type, entity storage, create, schedule, and cancel events. Use these functions:

- Define entity type
  - `entityType` — Define entity type
- Define entity storage
  - `queueFIFO` — Define first-in first-out (FIFO) queue storage
  - `queueLIFO` — Define last-in last-out (LIFO) queue storage
  - `queuePriority` — Define priority queue storage
  - `queueSysPriority` — Define system priority queue storage
- Create events
  - `eventGenerate` — Create entity generate event
  - `eventIterate` — Create entity iterate event
  - `eventTimer` — Create entity timer event
  - `eventForward` — Create entity forward event
  - `eventDestroy` — Create entity destroy event
  - `eventTestEntry` — Create an event to indicate that the acceptance policy for the storage has changed and the storage retests arriving entities
  - `eventAcquireResource` — Create a resource-acquiring event
  - `eventReleaseResource` — Create an event to release previously acquired resources(This method allows for partial resource release)

  - `eventReleaseAllResources` — Create an event to release all the resources acquired by an entity
- Cancel events
  - `cancelDestroy` — Cancel previously scheduled entity destroy event
  - `cancelForward` — Cancel entity forward event
  - `cancelGenerate` — Cancel previously scheduled entity generation event
  - `cancelIterate` — Cancel previously scheduled iterate event
  - `cancelTimer` — Cancel previously scheduled timer event
  - `cancelAcquireResource` — Cancel previously scheduled resource acquisition event
- Resource Management
  - `getResourceNamesImpl` — Define resource pools from which the discrete-event system acquires the resources
  - `resourceType` — Specify an entity type and the name of the resources to be acquired by the specified entity
  - `eventAcquireResource` — Create a resource-acquiring event
  - `eventReleaseResource` — Create an event to release previously acquired resources (This method allows for partial resource release)
  - `eventReleaseAllResources` — Create an event to release all the resources acquired by an entity
  - `cancelAcquireResource` — Cancel previously scheduled resource acquisition event
  - `resourceSpecification` — Specify the type and amount of resources for `eventAcquireResource` or `eventReleaseResource` requests
  - `initResourceArray` — Initialize a `resourceSpecification` array, required for code generation
  - `resourceAcquired` — Specify event actions upon successful resource acquisition
  - `resourceReleased` — Specify event actions upon successful resource release

## Inherited Methods from matlab.System Class

Inheriting `matlab.DiscreteEventSystem` class also inherits a subset of the `matlab.System` class methods.

| | |
|---|---|
| `getHeaderImpl` | Header for System object display |
| `getPropertyGroupsImpl` | Property groups for System object display |
| `isInactivePropertyImpl` | Inactive property status |
| `validatePropertiesImpl` | Validate property values |
| `processTunedPropertiesImpl` | Action when tunable properties change |
| `getNumInputsImpl` | Number of inputs to step method |
| `getInputNamesImpl` | Names of System block input ports |
| `getNumOutputsImpl` | Number of outputs from step method |
| `getOutputNamesImpl` | Names of System block output ports |

| | |
|---|---|
| `getDiscreteStateImpl` | Discrete state property values |
| `setupImpl` | Initialize System object |
| `resetImpl` | Reset System object states |
| `releaseImpl` | Release resources |
| `loadObjectImpl` | Load System object from MAT file |
| `saveObjectImpl` | Save System object in MAT file |
| `infoImpl` | Information about System object |
| `getOutputSizeImpl` | Sizes of output ports |
| `getOutputDataTypeImpl` | Data types of output ports |
| `isOutputComplexImpl` | Complexity of output ports |
| `getDiscreteStateSpecificationImpl` | Discrete state size, data type, and complexity |
| `getIconImpl` | Name to display as block icon |
| `getSampleTime` | Query sample time |

For more information about these methods, see "Customize System Objects for Simulink".

## Reference and Extract Entities

1 When referencing entity attributes or system properties in a discrete-event System object, use these formats:

| Attribute or Property | Format | Access |
|---|---|---|
| attribute | `entity.data.`*`attribute_name`* | Read/write |
| priority property | `entity.sys.priority` | Read/write |
| ID property | `entity.sys.id` | Read-only |

2 If an entity that is a part of a MATLAB Discrete-Event System block is requested for extraction, the `exit` method of the block is triggered. When the `exit` method is called, its **destination** argument is set to `extract`. See `modified` for entity modification.

## See Also

`matlab.DiscreteEventSystem` | `matlab.System`

## More About

- "Integrate System Objects Using MATLAB System Block"
- "Create Custom Blocks Using MATLAB Discrete-Event System Block" on page 9-2
- "Customize Discrete-Event System Behavior Using Events and Event Actions" on page 9-51

# Generate Code for MATLAB Discrete-Event System Blocks

To improve simulation performance, you can configure the MATLAB Discrete-Event System to simulate using generated code. With the **Simulate using** parameter set to `Code generation` option, the block simulates and generates code using only MATLAB functions supported for code generation.

MATLAB Discrete-Event System blocks support code reuse for models that have multiple MATLAB Discrete-Event System blocks using the same System object source file. Code reuse enables the code to be generated only once for the blocks sharing the System object.

## Migrate Existing MATLAB Discrete-Event System System object

Starting in R2017b, the MATLAB Discrete-Event System block can simulate using generated code. Existing applications continue to work with the **Simulate using** parameter set to `Interpreted execution`.

If you want to generate code for the block using MATLAB discrete-event system acceleration, update the System object code using these guidelines. For an example of updated MATLAB Discrete-Event System System object, see the "Develop Custom Scheduler of a Multicore Control System" on page 6-86 example.

### Replace Renamed matlab.DiscreteEventSystem Methods

To take advantage of simulation with code generation for the `matlab.DiscreteEventSystem` class:

1. In the `matlab.DiscreteEventSystem` application file, change these method names to the new names:

| Old Method Name | New Method Name |
| --- | --- |
| `blockedImpl` | `blocked` |
| `destroyImpl` | `destroy` |
| `entryImpl` | `entry` |
| `exitImpl` | `exit` |
| `generateImpl` | `generate` |
| `iterateImpl` | `iterate` |
| `setupEventsImpl` | `setupEvents` |
| `timerImpl` | `timer` |

2. In the code, move the renamed method definitions from a protected area to a public area for each `matlab.DiscreteEventSystem` method.

### Initialize System Properties

Initialize System object properties in the properties section. Do not initialize them in the constructor or other methods. In other words, you cannot use variable-size for System object properties.

### Initialize Empty Arrays of Events

Use the `initEventArray` to initialize arrays.

| Before | After |
| --- | --- |
| `function events = setupEventsImpl(obj)` | `function events = setupEvents(obj)`<br>`events = obj.initEventArray;` |

**Append Elements to Array of Structures**

Append elements to array of structures. For example:

| Before | After |
| --- | --- |
| `events(id) = obj.eventGenerate(1, num2str(id),...`<br>`0, obj.Priorities(id)); %#ok<*AGROW>` | `events = [events obj.eventGenerate(1, int2str(id),...`<br>`0, obj.Priorities(id))]; %#ok<AGROW>` |

**Replace Functions That Do Not Support Code Generation**

Replace functions that do not support code generation with functional equivalents that support code generation. For example:

| Before | After |
| --- | --- |
| `events(id) = obj.eventGenerate(1, num2str(id),...`<br>`0, obj.Priorities(id)); %#ok<*AGROW>` | `events = [events obj.eventGenerate(1, int2str(id),...`<br>`0, obj.Priorities(id))]; %#ok<AGROW>` |

**Declare Functions That Do Not Support Code Generation**

For functions that do not support code generation and that do not have functional equivalents, use the `coder.extrinsic` function to declare those functions as extrinsic. For example, `str2double` does not have a functional equivalent. Before calling the `coder.extrinsic`, make the returned variable the same data type as the function you are identifying. For example:

| Before | After |
| --- | --- |
| `id = str2double(tag);` | `coder.extrinsic('str2double');`<br>`id = 1;`<br>`id = str2double(tag);` |

- Do not pass System object to functions that are declared as extrinsic.
- Declare only static System object methods as extrinsic.

**Replace Cell Arrays**

Replace cell arrays with matrices or arrays of structures.

| Before | After |
| --- | --- |
| `entity.data.execTime = obj.ExecTimes{id}(1);` | `entity.data.execTime = obj.ExecTimes(id, 1);` |

**Change Flags to Logical Values**

Change flags from values such as `1` and `0` to logical values, such as `true` and `false`.

**Manage Global Data**

Manage global data while simulating with code generation using one of these:

- `evalin` and `assignin` functions in the MATLAB workspace

- “Static Data Object”

#### Move Logging and Graphical Functions

Many MATLAB logging and graphical functions do not support code generation. You can move logging and graphical functions into:

- A new `matlab.DiscreteEventSystem` object and configure the associated MATLAB Discrete-Event System block to simulate using `Interpreted execution` mode.
- An existing `simevents.SimulationObserver` object

#### Replace Persistent Variables

Replace persistent variable by declaring a System object property. See “Create System Objects” for more information.

## Limitations of Code Generation with Discrete-Event System Block

Limitations include:

- No “Global Variables”
- “System Objects in MATLAB Code Generation”
- “MATLAB System Block Limitations”

### See Also
`blocked` | `cancelForward` | `cancelGenerate` | `cancelIterate` | `cancelTimer` | `entry` | `eventForward` | `generate` | `getEntityPortsImpl` | `getEntityTypesImpl` | `iterate` | `matlab.DiscreteEventSystem` | `matlab.System` | `queueFIFO` | `setupEvents` | `timer`

### More About
- “Integrate System Objects Using MATLAB System Block”
- “Create Custom Blocks Using MATLAB Discrete-Event System Block” on page 9-2
- “Create a Discrete-Event System Object” on page 9-44

# Customize Discrete-Event System Behavior Using Events and Event Actions

| In this section... |
| --- |
| “Event Types and Event Actions” on page 9-51<br>“Event Identifiers” on page 9-53 |

You can customize the behavior of a discrete-event system by defining events and event actions.

You can:

- Schedule events
- Define event actions in response to events
- Initialize events
- Cancel events

## Event Types and Event Actions

### Event Types

A discrete-event system can have these event types and their targets.

| Event type | Target | Purpose |
| --- | --- | --- |
| `eventAcquireResource` | Entity | Allow an entity to acquire one or more resources. |
| `eventDestroy` | Entity | Destroy an existing entity in storage. |
| `eventForward` | Entity | Move an entity from its current storage to another storage or output port. |
| `eventIterate` | Storage | Iterate and process each entity in storage. |
| `eventReleaseResource` | Entity | Allow an entity to release one or more resources. |
| `eventReleaseAllResources` | Entity | Allow an entity to release all previously acquired resources. |
| `eventTestEntry` | Storage | Create an event to indicate that the storage acceptance policy is changed and the storage retests the arriving entities. |
| `eventTimer` | Entity | Create a timer event. |
| `eventGenerate` | Storage | Create an entity inside storage. |

- Forward events

If a forward event fails because of blocking, the forward event remains active. When space becomes available, the discrete-event system reschedules the forward event for immediate execution.

- Tagging events

  You can schedule multiple events of the same type for the same actor. When using multiple events of the same type, use tags to distinguish between the events. For example, an entity can have multiple timers with distinct tags. When one timer expires, you can use the `tag` argument of the `timer` method to differentiate which timer it is. For more information, see "Custom Entity Storage Block with Multiple Timer Events" on page 9-19.

  If you schedule two events with the same tag on the same actor, the later event replaces the first event. If you schedule two events with different tags, the discrete-event system calls them separately.

### Event Actions

When an event occurs, a discrete-event system responds to it by invoking a corresponding action. Implement these actions as System object methods. This table lists each action method and the triggering event.

| Event Action | Triggering Event | Purpose |
| --- | --- | --- |
| `blocked` | `eventForward` | Called if, upon execution of a forward event, the entity cannot leave due to blocking from the target storage. |
| `destroy` | `eventDestroy` | Called before an entity is destroyed and removed from storage. |
| `entry` | `eventForward` | Called upon an entity entry. |
| `exit` | `eventForward` | Called upon entity exit. When an entity is forwarded from storage 1 to storage 2, the exit action of storage 1 and then the entry action of storage 2 are called. |
| `generate` | `eventGenerate` | Called after a new entity is created inside a storage element. |
| `iterate` | `eventIterate` | Upon the execution of an Iterate event, this method is invoked for each entity from the front to the back of the storage, with the option of early termination. If entities need to be resorted due to key value changes, resorting takes place after the entire iteration is complete. |

| Event Action | Triggering Event | Purpose |
| --- | --- | --- |
| resourceAcquired | eventAcquireResource | Called after a successful resource acquisition. A resource acquisition is successful only if all of the specified resources are acquired. |
| resourceReleased | eventReleaseResource | Called after the resource release. |
| testEntry | eventTestEntry | Called after the test entry event. |
| timer | eventTimer | Called upon executing a timer event of an entity. |

#### Initialize Events

Use these methods to initialize empty arrays and events of a discrete-event system.

| Event Type | Purpose |
| --- | --- |
| initEventArray | Initialize event array. |
| initResourceArray | Initialize a resource specification array. |
| setupEvents | Initialize entity generation events. |

#### Cancel Previously Scheduled Events

Use these methods to cancel previously scheduled events of a discrete-event system.

| Event type | Purpose |
| --- | --- |
| cancelAcquireResource | Cancel previously scheduled resource acquisition event |
| cancelDestroy | Cancel previously scheduled entity destroy event. |
| cancelForward | Cancel entity forward event. |
| cancelGenerate | Cancel previously scheduled entity generation event. |
| cancelIterate | Cancel previously scheduled iterate event. |
| cancelTimer | Cancel previously scheduled timer event. |

## Event Identifiers

There are two distinct identifiers for the events provided by the `matlab.DiscreteEventSystem` class.

- Tag — Use the `tag` as an input argument for a method.

  ```
  event1 = obj.eventTimer('mytimer1', 2);
  event2 = obj.eventTimer('mytimer2', 5);
  ```

  Here, `mytimer1` and `mytimer2` are used as tags to refer to these two timer events.
- Destination — Use the destination to identify forward events.

```
event1 = obj.eventForward('storage', 2, 0.8);
event2 = obj.eventForward('output', 1, 2);
```

Here, `storage` and `output` are used to distinguish two forward events.

The events are not distinguishable when their identifiers are the same. This table shows how to identify an event when multiple events of the same type act on the same target.

| Event Type | Identification |
| --- | --- |
| `eventAcquireResource` | Tag |
| `eventGenerate` | Tag |
| `eventIterate` | Tag |
| `eventReleaseResource` | Tag |
| `eventReleaseAllResources` | Tag |
| `eventTimer` | Tag |
| `eventForward` | Destination |

**Note** If you define an event that is yet to be executed and a second event with the same type and identifier, the first event is replaced by the second one.

## See Also

`blocked` | `destroy` | `entry` | `eventForward` | `eventGenerate` | `generate` | `matlab.DiscreteEventSystem` | `matlab.System` | `setupEvents`

## More About

- “Create a Custom Entity Storage Block with Iteration Event” on page 9-14
- “Integrate System Objects Using MATLAB System Block”
- “Create Custom Blocks Using MATLAB Discrete-Event System Block” on page 9-2
- “Create a Discrete-Event System Object” on page 9-44

# Call Simulink Function from a MATLAB Discrete-Event System Block

This example shows how to call a Simulink function when an entity enters the storage element of a custom discrete-event system block, and to modify entity attributes. For more information about calling Simulink functions from MATLAB System block, see “Call Simulink Functions from MATLAB System Block”.

To represent this behavior, a custom block is generated with one input, one output, and one storage element. For more information about creating a custom entity storage block, see “Delay Entities with a Custom Entity Storage Block” on page 9-9.

### See the Code that Calls Simulink Function to Modify Entity Attributes

```
classdef CustomEntityStorageBlockSLFunc < matlab.DiscreteEventSystem

    % A custom entity storage block with one input, one output, and one storage.

    % Nontunable properties
    properties (Nontunable)
    % Capacity
        Capacity = 1;
    % Delay
        Delay = 4;
    end

    methods (Access=protected)
        function num = getNumInputsImpl(~)
            num = 1;
        end

        function num = getNumOutputsImpl(~)
            num = 1;
        end

        function entityTypes = getEntityTypesImpl(obj)
            entityTypes = obj.entityType('Car');
        end

        function [inputTypes,outputTypes] = getEntityPortsImpl(obj)
            inputTypes = {'Car'};
            outputTypes = {'Car'};
        end

        function [storageSpecs, I, O] = getEntityStorageImpl(obj)
            storageSpecs = obj.queueFIFO('Car', obj.Capacity);
            I = 1;
            O = 1;
        end

        function name = getSimulinkFunctionNamesImpl(obj)
            name = {'assignData'};
        end

    end

   methods

        function [entity,event] = CarEntry(obj,storage,entity,source)
            % Specify event actions when entity enters the storage.
            entity.Attribute1 = assignData();
            coder.extrinsic('fprintf');
            fprintf('Entity Attribute Value: %f\n', entity.Attribute1);

            event = obj.eventForward('output', 1, obj.Delay);

        end

    end

end
```

## Modify Entity Attributes

1 Define the name of the Simulink function to be called in the discrete-event System object using the `getSimulinkFunctionNamesImpl` method.

```
function name = getSimulinkFunctionNamesImpl(obj)
    % Declare the name of the Simulink Function.
    name = {'assignData'};
end
```

The name of the Simulink function is declared as `assignData`.

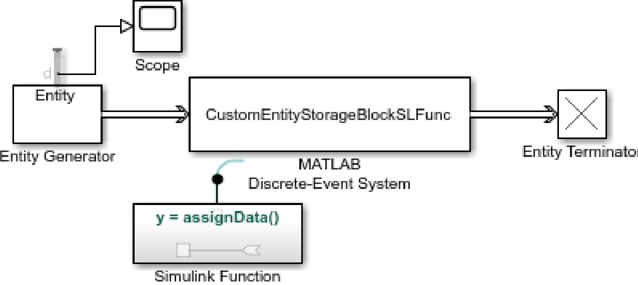
2 Call `assignData` in the entry event action.

```
function [entity,event] = CarEntry(obj,storage,entity,source)
    % Assign data when an entity enters the storage.
    entity.Attribute1 = assignData();
    coder.extrinsic('fprintf');
    fprintf('Entity Attribute Value: %f\n', entity.Attribute1);

    event = obj.eventForward('output', 1, obj.Delay);
end
```

## Build the Model

1 Create a model using an Entity Generator block, MATLAB Discrete-Event System block, and an Entity Terminator block.

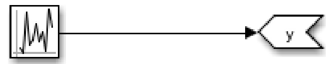
2 Open the MATLAB Discrete-Event System block, and set the **Discrete-event System object name** to `CustomEntityStorageBlockSLFunc`.




3 Output the **Number of entities departed, d** statistic from the Entity Generator block and connect it to a scope.

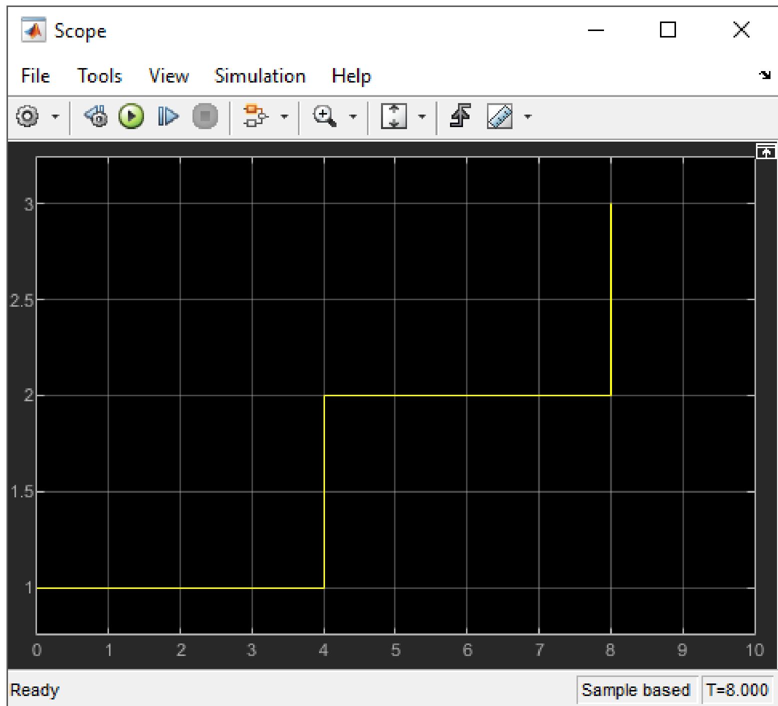
4 Add a Simulink Function block to your model.

a On the Simulink Function block, double-click the function signature and enter `y = assignData()` .



**b** In the Simulink Function block, add a Uniform Random Number block and change its **Sample time** parameter to `-1`.

5 Simulate the model. The scope displays 3 entities departed the Entity Generator block.




6 The Diagnostic Viewer displays the random attribute values assigned to 3 entities when they enter the storage.

```
Entity Attribute Value: -0.562082
Entity Attribute Value: -0.905911
Entity Attribute Value: 0.357729
```

## See Also

`entry` | `getEntityPortsImpl` | `getEntityStorageImpl` | `getEntityTypesImpl` | `matlab.DiscreteEventSystem` | `matlab.System`

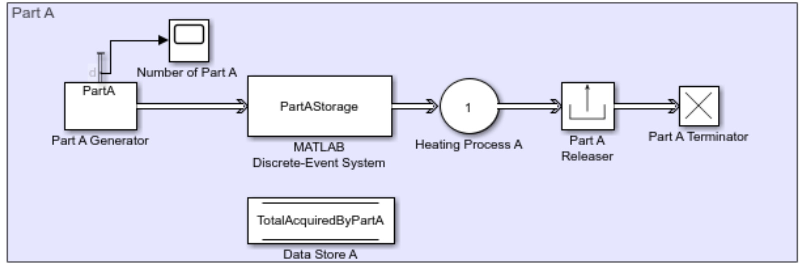
## More About

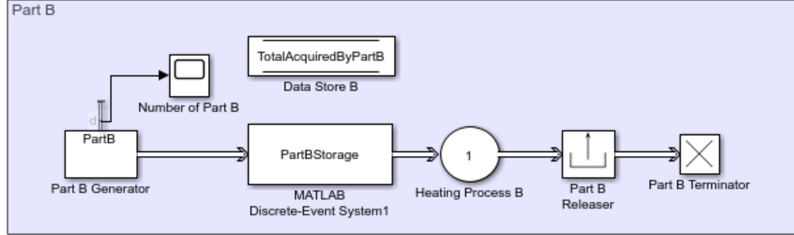

- "Delay Entities with a Custom Entity Storage Block" on page 9-9
- "Create a Custom Entity Storage Block with Iteration Event" on page 9-14
- "Create a Discrete-Event System Object" on page 9-44
- "Generate Code for MATLAB Discrete-Event System Blocks" on page 9-48

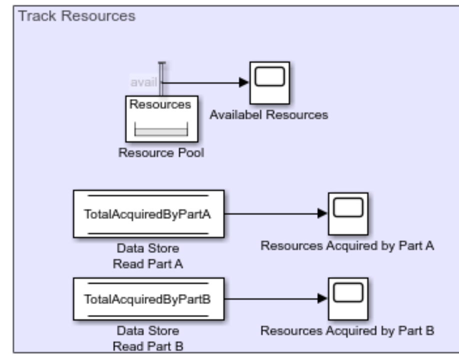
# Resource Scheduling Using MATLAB Discrete-Event System and Data Store Memory Blocks

This example shows how to model resource scheduling using data exchange between the MATLAB Discrete-Event System block and the Data Store Memory block.

The example models a facility that generates two types of parts, `Part A` and `Part B`, that undergo a heating process. Both parts acquire resources for the heating process from the same resource pool. The resource acquisition for `Part A` has a higher priority. When `Part A` acquires a certain number of resources, `Part B` can acquire only `1` resource. This constraint requires that the total number of resources be shared between the processes and the acquisition scheduled based on the shared data.




Copyright 2019 The MathWorks, Inc.

### Model Description

In the model, an Entity Generator Block generates entities of type `PartA`. The parts are then sent to a storage unit to acquire resources from the Resource Pool block. A MATLAB Discrete-Event System Block that uses the `PartAStorage` System Object™ represents the storage unit.

The System Object™ defines the amount of acquired resources and the resource acquisition event for `Part A`.

```
function [entity,event] = PartAEntry(obj,storage,entity,source)
    % Define the amount of acquired resources as a random value.
    Amount = randi([1 3]);
    resReq = obj.resourceSpecification('Resources', Amount);
    % Define the resource acquisition event.
    event = obj.eventAcquireResource(resReq, 'ResourceAcq');
end
```

When `Part A` acquires the resources successfully, the entity is forwarded to the output. `TotalAcquiredByPartA` is the data stored in the Data Store memory block representing the total number of acquired resources by `Part A`. The System Object™ first calls the value stored in Data

Store A. It updates and writes the new `TotalAcquiredByPartA` value by adding the number of acquired resources.

```
function [entity,events] = resourceAcquired(obj, storage,...
                                entity, resources, tag)
    global TotalAcquiredByPartA;
    % After succesful resource acquisition, forward the entity
    % to the output |1|.
    events = obj.eventForward('output', 1, obj.Delay);
    % Update the total number of resources acquired.
    TotalAcquiredByPartA = TotalAcquiredByPartA + resources.amount;
end
```

The part is sent to Heating Process A, which is represented by an Entity Server block. When the heating process is complete, the parts release the acquired resources and depart the facility.

In the model, another Entity Generator block generates entities of type `Part B`. The parts are then sent to a storage unit to acquire resources from the Resource Pool block. A MATLAB Discrete-Event System Block that uses the `PartBStorage` System Object™ represents the other storage unit.

The System Object™ defines the amount of acquired resources and the resource acquisition event for `Part B`.

```
function [entity,event] = PartBEntry(obj,storage,entity,source)
    global TotalAcquiredByPartA;
        % If the number of resources acquired by Part A is greater than
        % 30 then Part B acquires only |1| resource.
        if TotalAcquiredByPartA > 30
        Amount = 1;
        else
        % Otherwise, Part B can acquire any number of resources between
        % |1| and |5|.
        Amount = randi([1 5]);
        end
    resReq = obj.resourceSpecification('Resources', Amount);
    % Define the resurce acquisition event.
    event = obj.eventAcquireResource(resReq, 'ResourceAcq');
end
```

The amount of resources `Part B` acquires depends on the resources acquired by `Part A`. This acquisition is achieved by `PartBStorage` System Object™ that reads the value of `TotalAcquiredByPartA` stored in Data Store A for each entity entry.

After successfully acquiring the resources, the entity is forwarded to the output. The System Object (TM) updates `TotalAcquiredByPartB` and writes its new value to Data Store B.

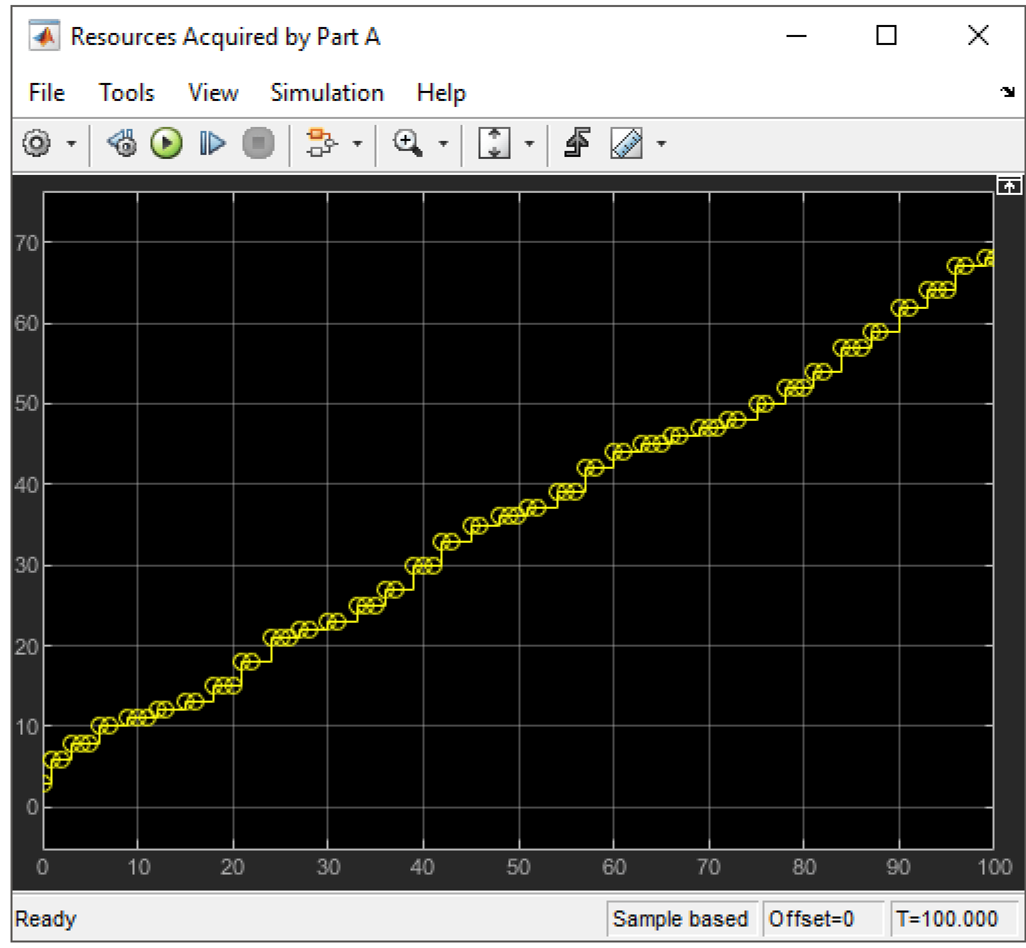
```
function [entity,events] = resourceAcquired(obj, storage,...
                                 entity, resources, tag)
    global TotalAcquiredByPartB; % After succesful resource
    acquisition, forward the entity to the output. events =
    obj.eventForward('output', 1, obj.Delay); % Update the total number
    of resources acquired. TotalAcquiredByPartB = TotalAcquiredByPartB
    + resources.amount;
end
```

Then the parts are sent to Heating Process B. They release the resources after the process is complete and depart the facility.

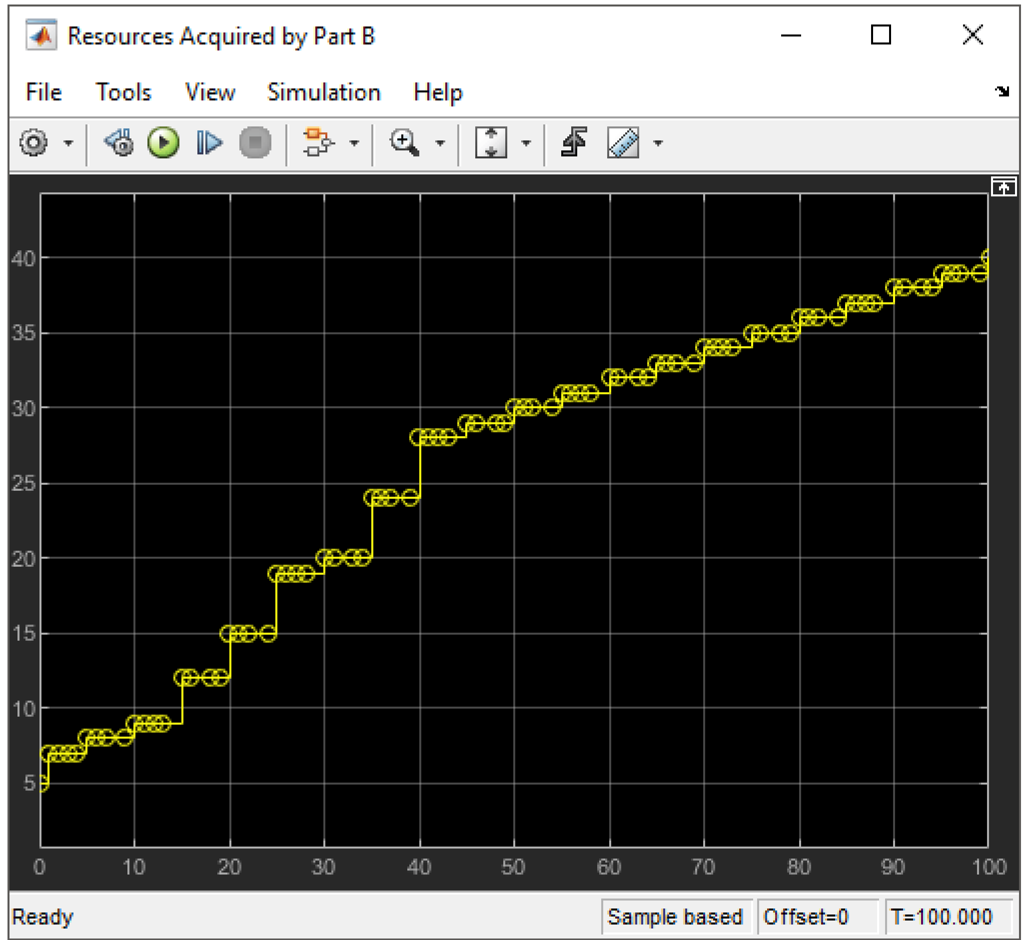
Track Resources component in the model, tracks available resources and acquired number of resources by each part. Available resources are measured by the **Amount available, avail** statistic from the Resource Pool block. Resources acquired by `Part A` and `Part B` is observed by the output of the Data Store Read blocks that read values from Data Store A and Data Store B.

### Simulation Results

Simulate the model. Observe the Scope block connected to the Data Store Read Part A. The scope shows that `Part A` acquires `30` resources around the simulation time `40`.




Also observe the Scope block connected to Data Store Read Part B. The scope shows that `Part B` acquires `1` resource after the simulation time `40` due to the prioritization of resources.




## See Also

### More About

- “Call Simulink Function from a MATLAB Discrete-Event System Block” on page 9-55
- “Delay Entities with a Custom Entity Storage Block” on page 9-9
- “Create a Custom Entity Storage Block with Iteration Event” on page 9-14
- “Create a Discrete-Event System Object” on page 9-44

10

# Custom Visualization


- “Use SimulationObserver Class to Monitor a SimEvents Model” on page 10-2
- “Observe Entities Using simevents.SimulationObserver Class” on page 10-5

# Use SimulationObserver Class to Monitor a SimEvents Model

**In this section...**

"SimulationObserver Class" on page 10-2
"Custom Visualization Workflow" on page 10-2
"Create an Application" on page 10-2
"Use the Observer to Monitor the Model" on page 10-4
"Stop Simulation and Disconnect the Model" on page 10-4

## SimulationObserver Class

To create an observer, create a class that derives from the `simevents.SimulationObserver` object. You can use observers to:

- Help understand queue impact, visualize entities moving through the model during simulation,
- Develop presentation tools showing model simulation via an application-oriented interface, such as restaurant queue activity.
- Debug and examine entity activity.
- Examine queue contents.

The `simevents.SimulationObserver` object provides methods that let you:

- Create observer or animation objects.
- Identify model blocks for notification of run-time events.
- Interact with the event calendar.
- Perform activities when a model pauses, continues after pausing, and terminates.

SimEvents models call these functions during model simulation.

## Custom Visualization Workflow

1. Create an application file.
   a. Define a class that inherits from the `simevents.SimulationObserver` class.
   b. Create an observer object that derives from this class.
   c. From the `simevents.SimulationObserver` methods, implement the functions you want for your application. This application comprises your observer.
2. Open the model.
3. Create an instance of your class.
4. Run the model.

For more information about custom visualization, see "Create Custom Visualization".

## Create an Application

You can use these methods in your derived class implementation of `simevents.SimulationObserver`.

| Action | Method |
|---|---|
| Specify behavior when simulation starts. | simStarted |
| Specify behavior when simulation pauses. | simPaused |
| Specify behavior when simulation resumes. | simResumed |
| Define observer behavior when simulation is terminating. | simTerminating |
| Specify list of blocks to be notified of entity entry and exit events. | getBlocksToNotify |
| Specify whether you want notification for all events in the event calendar. | notifyEventCalendarEvents |
| Specify behavior after an entity enters a block that has entity storage. | postEntry |
| Specify behavior before an entity exits a block with entity storage. | preExit |
| Specify behavior before execution of an event. | preExecute |
| Add block to list of blocks to be notified. | addBlockNotification |
| Remove block from list of blocks being notified. | removeBlockNotification |
| Get handles to event calendars. | getEventCalendars |
| Get list of blocks that store entities. | getAllBlockWithStorages |
| Return block handle for a given block path. | getHandleToBlock |
| Return storage handles of specified block. | getHandlesToBlockStorages |

1. In the MATLAB Command Window, select **New** > **Class**.
2. In the first line of the file, inherit from the `simevents.SimulationObserver` class. For example:

   ```
   classdef seRestaurantAnimator < simevents.SimulationObserver
   ```

   `seRestaurantAnimator` is the name of the new observer object.
3. In the `properties` section, enter the properties for your application.
4. In the `methods` section, implement the functions for your application.
5. To construct the observer object, enter a line like the following in the `methods` section of the file:

   ```
   function this = seRestaurantAnimator
               % Constructor
               modelname = 'seCustomVisualization';
               this@simevents.SimulationObserver(modelname);
               this.mModel = modelname;
           end
   ```

For more information, see “Using Custom Visualization for Entities” on page 6-62.

## Use the Observer to Monitor the Model

1. Open the model to observe.
2. At the MATLAB command prompt, to enable the animator for the model:

   ```
   >> obj=seRestaurantAnimator;
   ```
3. Simulate the model.

   When the model starts, the animator is displayed in a figure window. As the model runs, it makes calls into your application to see if you have implemented one of the predefined set of functions. If your model does not contain a SimEvents block, you receive an error.

---

**Note** As a result of the instrumentation to visualize the simulation, the simulation is slower than without the instrumentation.

---

## Stop Simulation and Disconnect the Model

1. Stop the simulation.
2. At the MATLAB command prompt, clear the animator from the model. For example:

   ```
   clear obj;
   ```

## See Also

`simevents.SimulationObserver`

## Related Examples

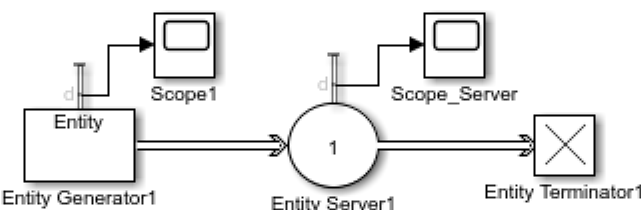
- “Observe Entities Using simevents.SimulationObserver Class” on page 10-5
- “Visualization and Animation for Debugging” on page 5-10
- “Using Custom Visualization for Entities” on page 6-62

# Observe Entities Using simevents.SimulationObserver Class

This example shows how to use `simevents.SimulationObserver` object to count entity departures and acquire departure timestamps.

Use the `simevents.SimulationObserver` object to observe or visualize entities, and implement animators to debug model simulations. For more information, see "Use SimulationObserver Class to Monitor a SimEvents Model" on page 10-2.

In this model, the `simevents.SimulationObserver` object is used to acquire the number of entities departing a block or a set of blocks in the model and timestamp their departures. The model has two Entity Generator and Entity Terminator blocks and an Entity Server Block. The Scope blocks display the **Number of entities departed, d** statistics for the Entity Generator and Entity Server blocks.



Copyright 2019 The MathWorks, Inc.

### Create the Observer

Open a new script and initiate the `simevents.SimulationObserver` object by this code.

```
classdef myObserverPreexit < simevents.SimulationObserver
    % Add the observer properties.
    properties
    Model
    % Initialize the property count.
    count
    end

properties (Constant, Access=private)
    increment = 1;
end

methods

     % Observe any model by incorporating its name to MyObserverPreexit.
     function this = myObserverPreexit(Model)
```

```
            % Input model name to the simulation observer.
            this@simevents.SimulationObserver(Model);
            this.Model = Model;
        end

        % Initialize the count in the simulation start.
        function simStarted(this)
            this.count = 0;
        end

        % Specify list of blocks to be notified of entity entry and exit
        % events.
        function Block = getBlocksToNotify(this)
            Block = this.getAllBlockWithStorages();
        end

        function preExit(this,evSrc,Data)
            % Get the names of all storage blocks that the entities depart.
            % This returns the block with its path.
            Block = Data.Block.BlockPath;
            % Remove the path to display only the
            % block name.
            Block = regexprep(Block,'ObserverPreexitModel/' ,'');
            % Initialize the blocks to observe.
            BlockName = 'Entity Server';
            % If the block that entity exits contains the block name
            % acquire data for exit time and block name.
            if contains(Block, BlockName)
                % Get time for entity preexit from event calendar.
                evCal = this.getEventCalendars;
                Time = evCal(1).TimeNow;
                % Increase the count for departing entities.
                this.count = this.count + this.increment;

                myInfo = [' At time ',num2str(Time), ...
                ' an entity departs ', Block, ', Total entity count is ', ...
                num2str(this.count)];
                disp(myInfo);
                end
            end
        end
    end
```

Save the file as `myObserverPreexit.m` file.

#### Monitor the Model

Enable the observer object to monitor `ObserverPreexitModel` model.

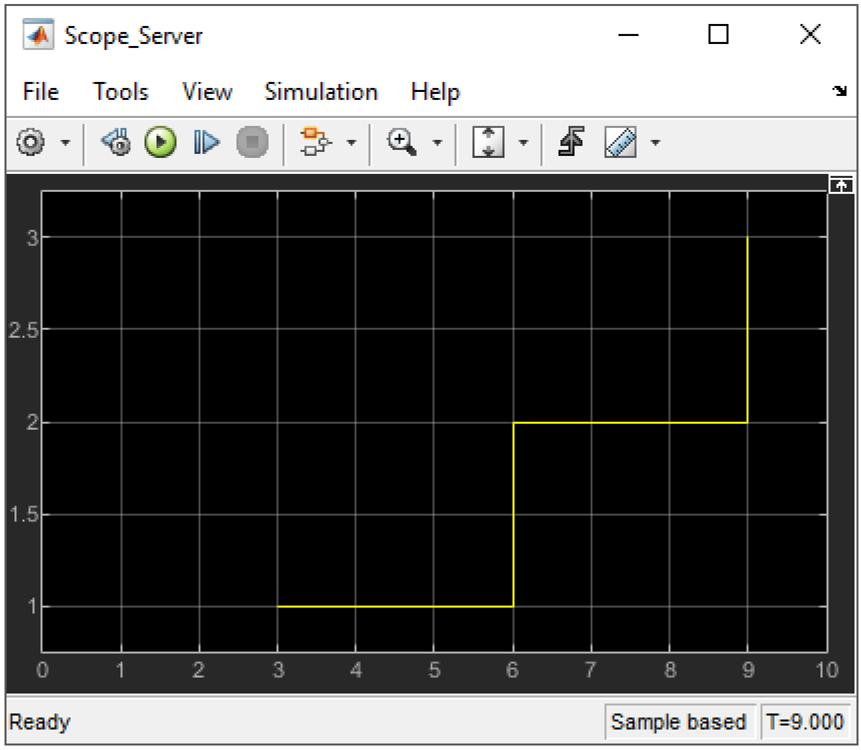
```
obj  = myObserverPreexit('ObserverPreexitModel');
```

The observer monitors the Entity Server block, which is determined by the `BlockName` parameter in the observer file `myObserverPreexit.m`.

- Simulate the model. Click **View Diagnostics** on the model window and observe that the number of entities departing the Entity Server block and the departure timestamps.

```
At time 3 an entity departs Entity Server1, Total entity count is 1
At time 6 an entity departs Entity Server1, Total entity count is 2
At time 9 an entity departs Entity Server1, Total entity count is 3
```

- For validation, observe the Scope block that displays the **Number of entities departed, d** statistic for the Entity Server block.




#### Monitor Multiple Blocks in the Model

Use the same observer to monitor the entity departures from all of the Entity Generator blocks in your model.

- Change the `BlockName` parameter in the `preExit` method to `'Entity Generator'`. Entity Generator blocks in the model are labeled Entity Generator1 and Entity Generator2.

```
function preExit(this,evSrc,Data)
    % Get the names of all storage blocks that the entities depart.
    % returns the block with its path.
    Block = Data.Block.BlockPath;
    % Remove the path to display only the block name
    Block = regexprep(Block,'ObserverPreexitModel/' ,'');
    % Initialize the common Entity Generator phrase
    BlockName = 'Entity Generator';
    % If the block that the entity exits contains the block name
    % acquire the exit time and the block name.
    if contains(Block, BlockName)
        % Get the time of entity preexit from the event calendar.
        evCal = this.getEventCalendars;
        Time = evCal(1).TimeNow;
```

```
            % Increase the count of departing entities.
            this.count = this.count + this.increment;

             myInfo = [' At time ',num2str(Time), ...
                 ' an entity departs ', Block, ', Total entity count is ', ...
             num2str(this.count)];
             disp(myInfo);
        end
end
```

- Enable the observer object to monitor `ObserverPreexitModel` model.

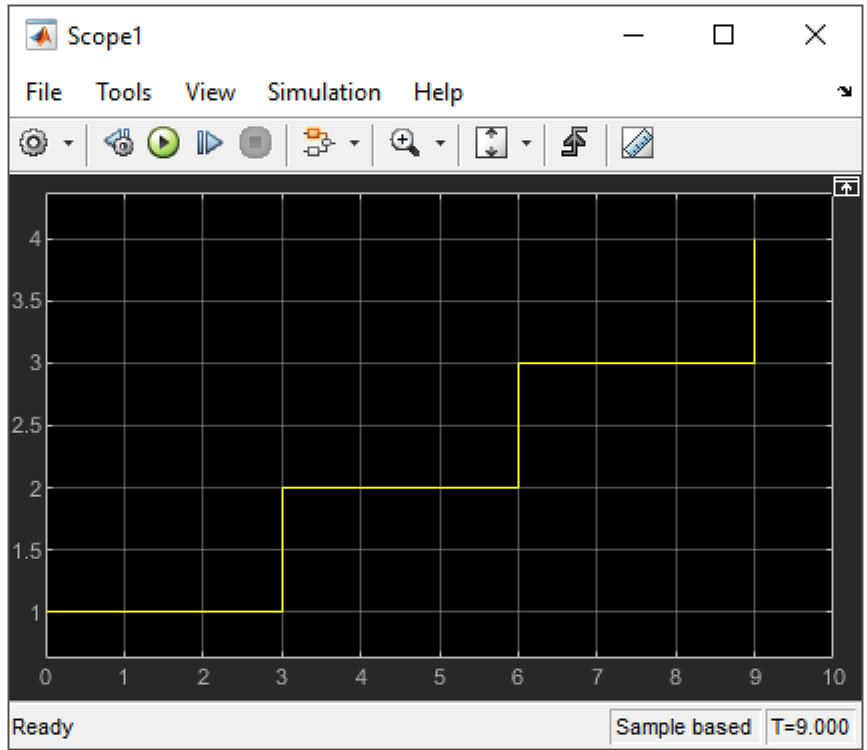
```
obj  = myObserverPreexit('ObserverPreexitModel');
```

- Simulate the model. Observe the Diagnostic Viewer that displays the information for 15 entities departing from both Entity Generator blocks.

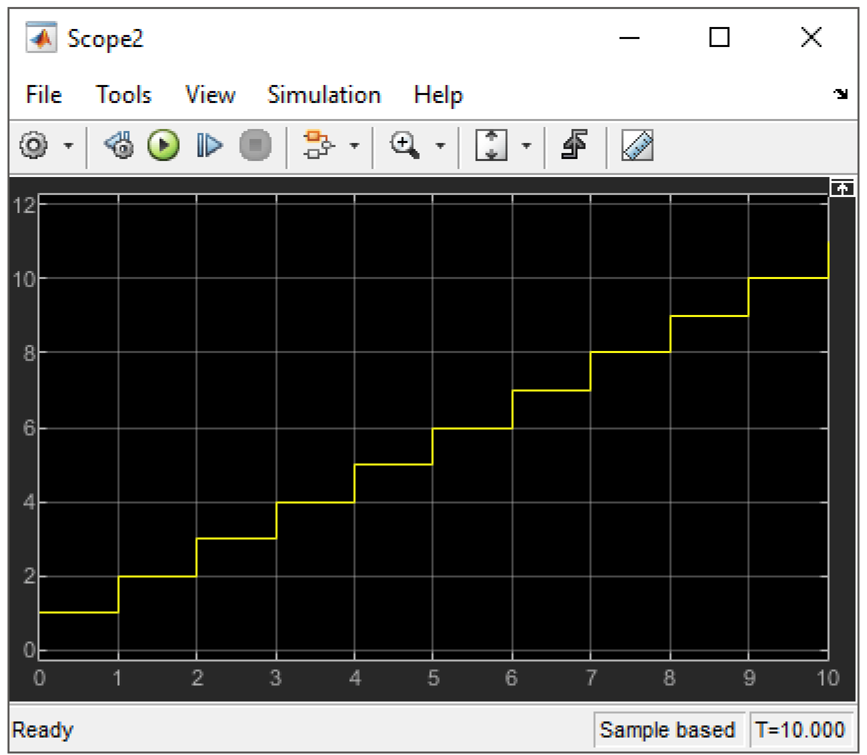
```
At time 0 an entity departs Entity Generator1, Total entity count is 1
At time 0 an entity departs Entity Generator2, Total entity count is 2
At time 1 an entity departs Entity Generator2, Total entity count is 3
At time 2 an entity departs Entity Generator2, Total entity count is 4
At time 3 an entity departs Entity Generator1, Total entity count is 5
At time 3 an entity departs Entity Generator2, Total entity count is 6
At time 4 an entity departs Entity Generator2, Total entity count is 7
At time 5 an entity departs Entity Generator2, Total entity count is 8
At time 6 an entity departs Entity Generator1, Total entity count is 9
At time 6 an entity departs Entity Generator2, Total entity count is 10
At time 7 an entity departs Entity Generator2, Total entity count is 11
At time 8 an entity departs Entity Generator2, Total entity count is 12
At time 9 an entity departs Entity Generator1, Total entity count is 13
At time 9 an entity departs Entity Generator2, Total entity count is 14
At time 10 an entity departs Entity Generator2, Total entity count is 15
```

- For validation, observe Scope1 and Scope2 blocks display the **Number of entities departed, d** statistic for the Entity Generator1 and the Entity Generator2.

Observe that 4 entities depart Entity Generator1.




Also, 11 entities depart Entity Generator2. In total, 15 entities departed from the Entity Generator blocks in the model.




## See Also

`getBlocksToNotify` | `getEventCalendars` | `preExit` | `simStarted` | `simevents.SimulationObserver`

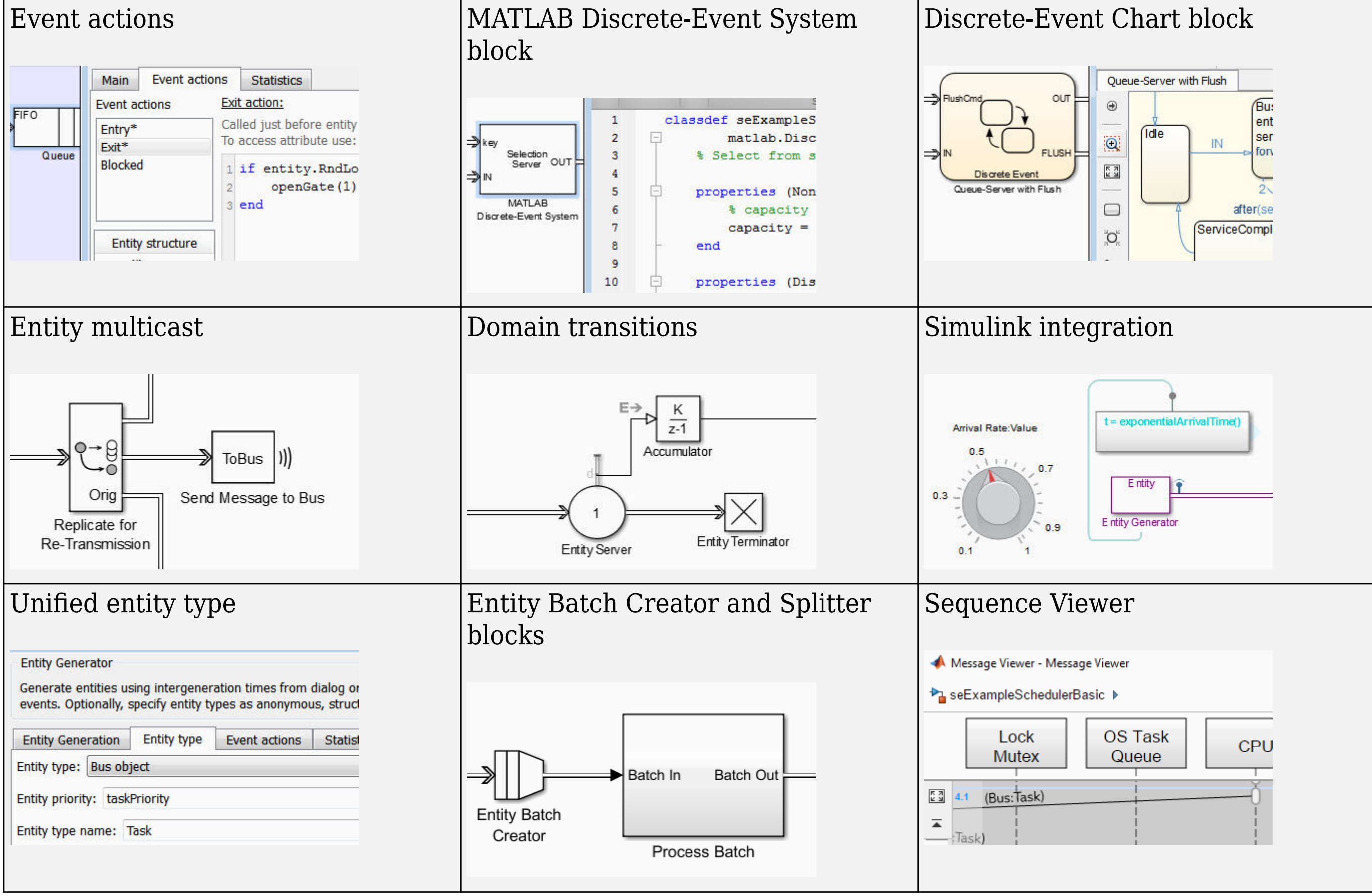
## More About

- "Use SimulationObserver Class to Monitor a SimEvents Model" on page 10-2
- "Visualization and Animation for Debugging" on page 5-10
- "Using Custom Visualization for Entities" on page 6-62

# Migrating SimEvents Models

# Migration Considerations

To take advantage of SimEvents features, migrate legacy SimEvents models (pre-R2016a). Benefits include:




Use SimEvents software to:

- Modify entity attributes, service, and routes on events such as entity generation, entry, and exit.
- Create custom SimEvents blocks using MATLAB.
- Create Stateflow state transition diagrams that process entities, react to entity events, and follow precise timing for temporal operations.
- Wirelessly broadcast copies of entities to multiple receive queues.
- Automatically switch between time-based and event-based signals.
- Use Simulink features, such as Fast Restart to speed up simulation runs and Simulation Stepper to debug.
- Define entity types that are consistent across Simulink, Stateflow, and SimEvents products.
- Create and split batch of entities.
- Display interchange of messages and entities.

## When You Should Not Migrate

If your legacy model contains timeout blocks, do not migrate the model. You can still access legacy blocks to continue developing older models by using the blocks in the Legacy Block Library.

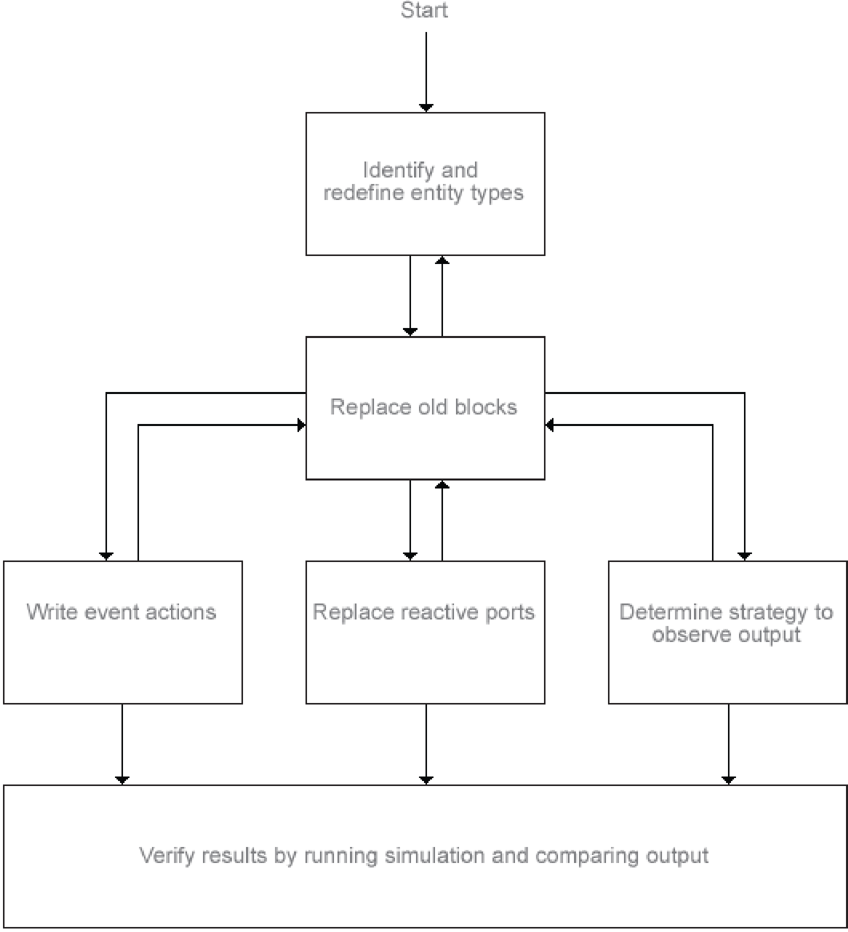
## See Also

## More About


- "Migration Workflow" on page 11-4
- "Identify and Redefine Entity Types" on page 11-6
- "Replace Old Blocks" on page 11-8
- "Connect Signal Ports" on page 11-11
- "Write Event Actions for Legacy Models" on page 11-15
- "Observe Output" on page 11-22
- "Reactive Ports" on page 11-23

# Migration Workflow

This migration workflow helps you migrate legacy SimEvents models to R2016a or later. In this workflow, you create a new SimEvents model to replace your legacy SimEvents model. This is an iterative workflow that requires you to repeat some steps.




1. Before you start, copy your legacy model to a backup folder. Run the old model and collect the results using the Simulation Data Inspector ("Inspect Simulation Data").

   **Note** Pre-R2016a SimEvents blocks cannot coexist in a model with post-R2016a SimEvents blocks.

2. Identify and redefine entity types ("Identify and Redefine Entity Types" on page 11-6)
3. When possible, replace old blocks with new blocks ("Replace Old Blocks" on page 11-8) and reconfigure the new blocks.
4. Write event actions for these instances:

   a. Replace Set Attribute blocks with event actions in other blocks ("Replace Set Attribute Blocks with Event Actions" on page 11-15)

   b. Replace Get Attribute blocks with event actions in other blocks (“Connect Signal Ports” on page 11-11)
   c. Replace Attribute Function blocks with event actions in other blocks (“Replace Attribute Function Blocks with Event Actions” on page 11-18)
   d. Replace random number generators with event actions in other blocks (“Replace Random Number Distributions in Event Actions” on page 11-16)
5. Replace reactive ports (see “If Connected to Reactive Ports” on page 11-13).
6. Determine a strategy to observe output by replacing Discrete Event Signal to Workspace blocks with To Workspace blocks or logging (“Observe Output” on page 11-22).
7. Verify the results by running the simulation and using Simulation Data Inspector to compare these results with those you collect in step 1.

## See Also

## More About


- “Migration Considerations” on page 11-2
- “Identify and Redefine Entity Types” on page 11-6
- “Replace Old Blocks” on page 11-8
- “Connect Signal Ports” on page 11-11
- “Write Event Actions for Legacy Models” on page 11-15
- “Observe Output” on page 11-22
- “Reactive Ports” on page 11-23

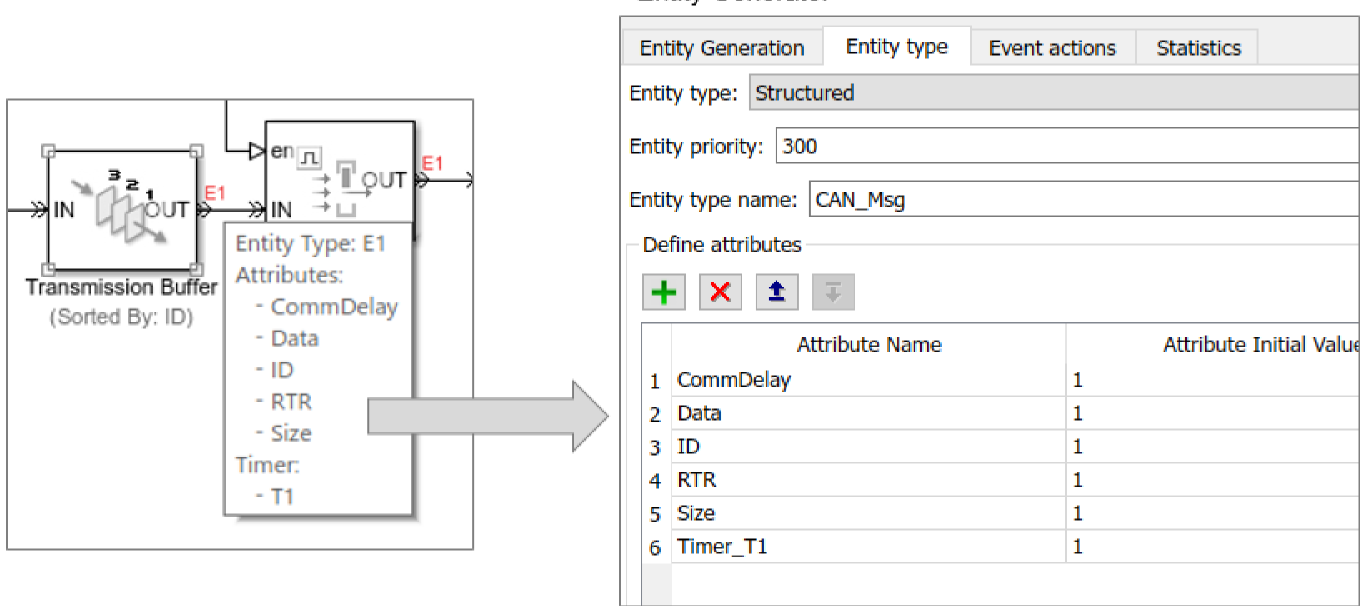
# Identify and Redefine Entity Types

Identify entity types in the legacy model and redefine them in the new model.

1. In the old model, identify all Entity Generator blocks that feed each Entity Sink block.
2. In the model, from the Toolstrip, select **Debug** > **Information Overlays** > **Base Data Types**.
3. To see the attributes at each Entity Generator, Entity Sink, or other termination points of entity flow, hover over the entity label to display attribute associated with the entity. A popup window displays the attributes associated with the port.

   Repeat this step for each block and note the attributes.
4. In the new model, add Entity Generator blocks to replace those in the legacy model.
5. In the model, in the Entity Generator block **Entity type** tab, define the entity type for each block with the full list of attributes for that block (found in step 3).

This example shows the redefined attributes,




Once you define the entity types, return to “Migration Workflow” on page 11-4.

## See Also

## More About

- “Migration Considerations” on page 11-2
- “Migration Workflow” on page 11-4
- “Replace Old Blocks” on page 11-8
- “Connect Signal Ports” on page 11-11
- “Write Event Actions for Legacy Models” on page 11-15

- “Observe Output” on page 11-22
- “Reactive Ports” on page 11-23

# Replace Old Blocks

The primary goal in migration is to replace legacy SimEvents behavior with new SimEvents behavior.

This table lists:

- New SimEvents blocks to replace legacy SimEvents blocks
- Actions to take when there is no equivalent new SimEvents block to replace the legacy block. Some of these actions are also part of the migration workflow.

| Old Block | Action for New SimEvents Model |
|---|---|
| Attribute Function | Wait until “Replace Attribute Function Blocks with Event Actions” on page 11-18. |
| Attribute Scope | Wait until “If Using Get Attribute Blocks to Observe Output” on page 11-11. |
| Cancel Timeout | Consider not yet migrating your model. |
| Conn | Simulink Inport or Outport block. |
| Discrete Event Signal to Workspace | Wait until “Observe Output” on page 11-22. |
| Enabled Gate | Replace with Entity Gate. |
| Entity Combiner | Replace with Composite Entity Creator. |
| Entity Departure Counter | Wait until “Write Event Actions for Legacy Models” on page 11-15. |
| Entity Departure Function-Call Generator | Wait until “Write Event Actions for Legacy Models” on page 11-15. |
| Entity Sink | Replace with Entity Terminator. |
| Entity Splitter | Replace with Composite Entity Splitter. |
| Entity Departure Function-Call Generator | Wait until “Write Event Actions for Legacy Models” on page 11-15. |
| Event Filter | Delete (block no longer needed). |
| Event to Timed Function-Call | Delete (block no longer needed). |
| Event to Timed Signal | Delete (block no longer needed). |
| Event-Based Entity Generator | Replace with Entity Generator. |
| Event-Based Random Number | Wait until “Replace Random Number Distributions in Event Actions” on page 11-16. |
| Event-Based Sequence | Wait until “Write Event Actions for Legacy Models” on page 11-15. |
| FIFO Queue | Replace with Entity Queue. |
| Get Attribute | Wait until “Connect Signal Ports” on page 11-11. |
| Infinite Server | Replace with Entity Server. |
| Initial Value | Delete (block no longer needed). |
| Input Switch | Replace with Entity Input Switch. |

| Old Block | Action for New SimEvents Model |
| --- | --- |
| Instantaneous Entity Counting Scope | Wait until "If Using Get Attribute Blocks to Observe Output" on page 11-11. |
| Instantaneous Event Counting Scope | Delete (block no longer needed). |
| LIFO Queue | Replace with Entity Queue. |
| N-Server | Replace with Entity Server. |
| Output Switch | Replace with Entity Output Switch. |
| Path Combiner | Input Switch (with **All** selected). |
| Priority Queue | Replace with Entity Queue. |
| Read Timer | For an example, see "Measure Point-to-Point Delays" on page 1-46. |
| Release Gate | Replace with Entity Gate. |
| Replicate | Replace with Entity Replicator. |
| Resource Acquire | Replace with Resource Acquire. |
| Resource Pool | Replace with Resource Pool. |
| Resource Release | Replace with Resource Releaser. |
| Schedule Timeout | Consider not yet migrating your model. |
| Set Attribute | Wait until "Replace Set Attribute Blocks with Event Actions" on page 11-15. |
| Signal Latch | Delete (block no longer needed). |
| Signal Scope | Replace with Simulink Scope. |
| Signal-Based Function-Call Event Generator | Wait until "If Connected to Reactive Ports" on page 11-13. |
| Signal-Based Function-Call Generator | Wait until "If Connected to Reactive Ports" on page 11-13. |
| Single Server | Replace with Entity Server. |
| Start Timer | For an example, see "Measure Point-to-Point Delays" on page 1-46. |
| Time-Based Entity Generator | Replace with Entity Generator. |
| Time-Based function-Call Generator | Replace with Entity Generator. |
| Timed to Event Function-Call | Delete (block no longer needed). |
| Timed to Event Signal | Delete (block no longer needed). |
| X-Y Attribute Scope | See "If Connected to Computation Blocks" on page 11-12. |
| X-Y Signal Scope | Simulink XY Graph. |

When done, return to "Migration Workflow" on page 11-4.

## See Also

## More About

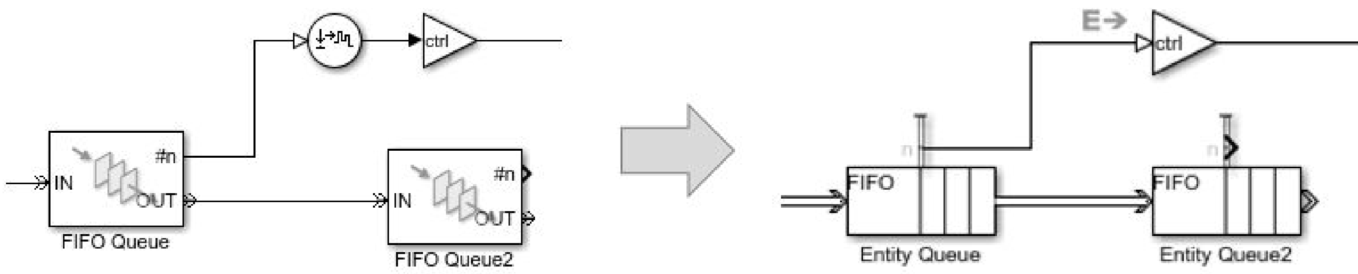

- “Migration Considerations” on page 11-2
- “Migration Workflow” on page 11-4
- “Identify and Redefine Entity Types” on page 11-6
- “Connect Signal Ports” on page 11-11
- “Write Event Actions for Legacy Models” on page 11-15
- “Observe Output” on page 11-22
- “Reactive Ports” on page 11-23

# Connect Signal Ports

Previous releases use Get Attribute blocks to output the values of entity attributes. SimEvents 5.0 is more closely tied to Simulink. This close association lets you use traditional Simulink tools to get attribute values. Replace Get Attribute blocks using these guidelines.

## If Connected to Gateway Blocks

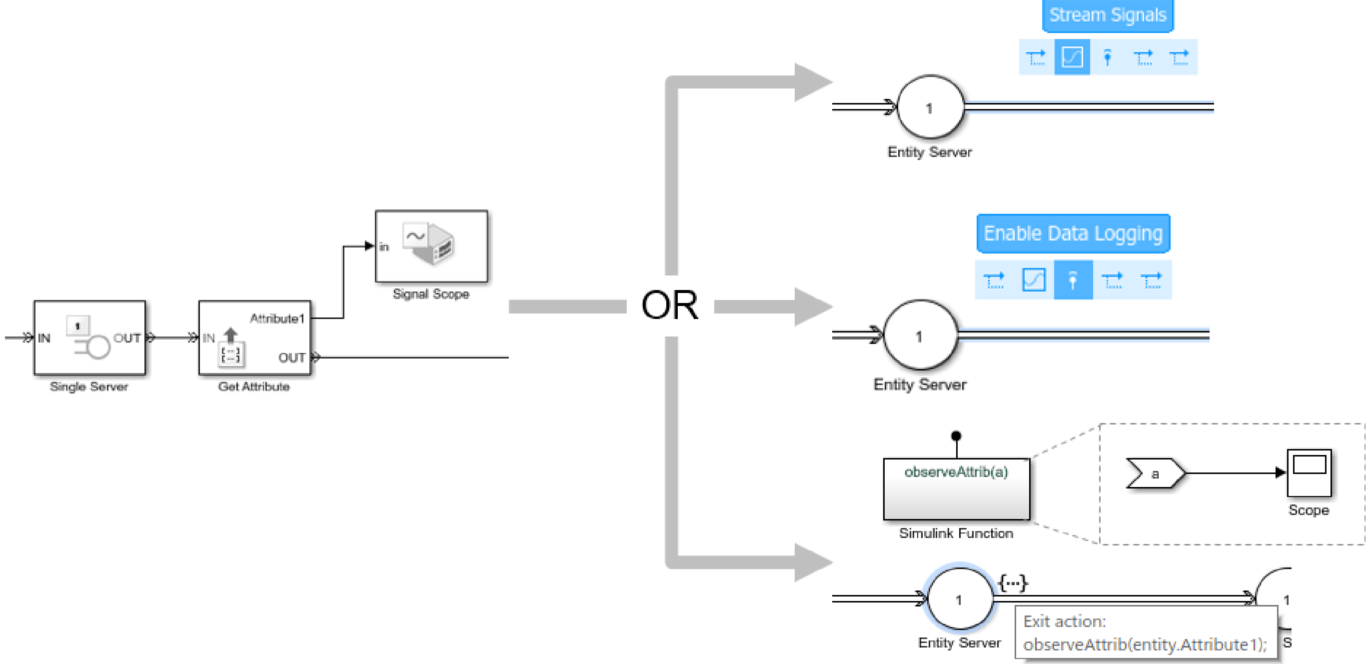
SimEvents models no longer require gateway blocks. Remove all gateway blocks, as shown in the figure:




Return to “Connect Signal Ports” on page 11-11.

## If Using Get Attribute Blocks to Observe Output

If you use Get Attribute blocks to observe output, see “Observe Output” on page 11-22. For example, you can use the Simulation Data Inspector to visualize entities from an Entity Generator block. This example shows how to visualize entities using the Simulation Data Inspector, logging, and a scope.



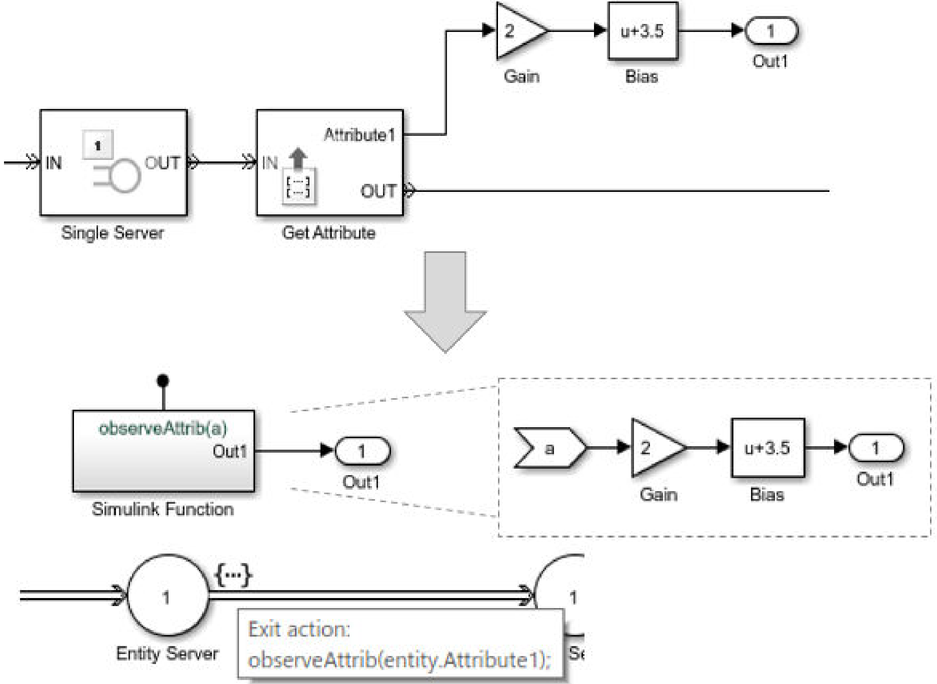
Return to “Connect Signal Ports” on page 11-11.

## If Connected to Computation Blocks

If the Get Attribute block is connected to computational blocks, reproduce the behavior of these blocks with Simulink Function blocks.

1. Place the computation blocks in a Simulink Function block.
2. Call the Simulink Function block from an event action.

This example places the Gain and Bias blocks in the Simulink Function block.




This table shows how each statistics port gets updated.

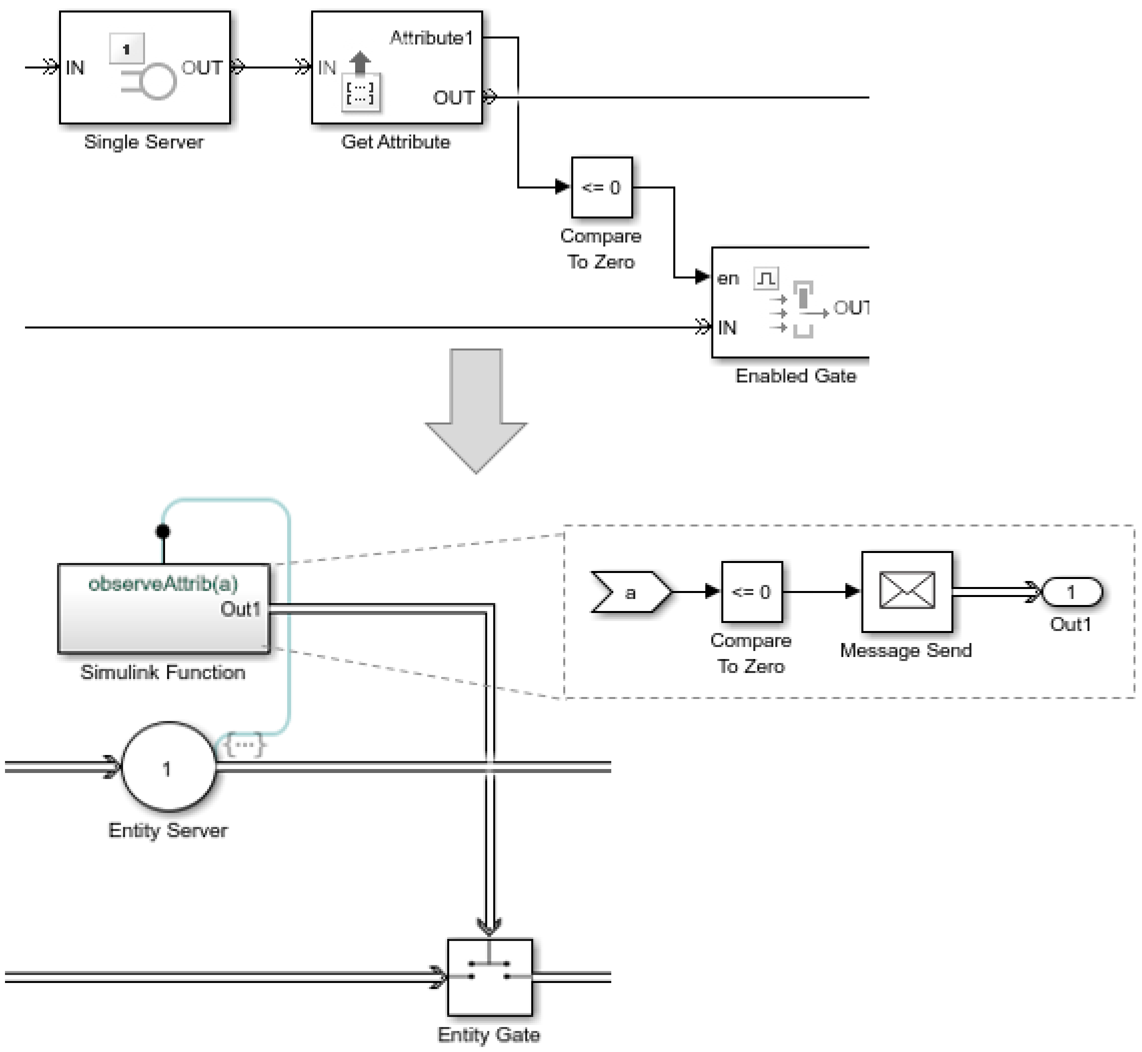
| Statistics Port | Updated on Event | | | |
|---|---|---|---|---|
| | Entry | Exit | Blocked | Preempted |
| Number of entities departed, d | | ✓ | | |
| Number of entities in block, n | ✓ | | | |
| Number of entities arrived, a | ✓ | | | |
| Pending entity present in block, pe | | ✓ | ✓ | ✓ |
| Number of pending entities, np | | ✓ | ✓ | ✓ |

| Statistics Port | Updated on Event | | | |
|---|---|---|---|---|
| | Entry | Exit | Blocked | Preempted |
| Number of entities preempted, p | | | | ✓ |
| Average intergeneration time, w | | | | |
| Average wait, w | | ✓ | | ✓ |
| Average queue length, l | ✓ | ✓ | | |
| Utilization, util | ✓ | ✓ | | ✓ |

Return to “Connect Signal Ports” on page 11-11.

## If Connected to Reactive Ports

In previous releases, reactive ports are signal input ports that listen for updates or changes in the input signal. When the input signal changes, an appropriate reaction occurs in the block possessing the port. Convert all reactive port event signals to messages, as in this example.




For more information, see “Reactive Ports” on page 11-23.

Return to “Connect Signal Ports” on page 11-11.

## See Also

## More About

- “Migration Considerations” on page 11-2
- “Migration Workflow” on page 11-4
- “Identify and Redefine Entity Types” on page 11-6
- “Replace Old Blocks” on page 11-8
- “Write Event Actions for Legacy Models” on page 11-15
- “Observe Output” on page 11-22
- “Reactive Ports” on page 11-23

# Write Event Actions for Legacy Models

When migrating legacy SimEvents models, you often must create event actions in these instances:

- Setting attribute values
- Getting attribute values
- Generating random number generation
- Using Event sequences
- Replacing Attribute Function blocks
- Using Simulink signals in an event-based computation

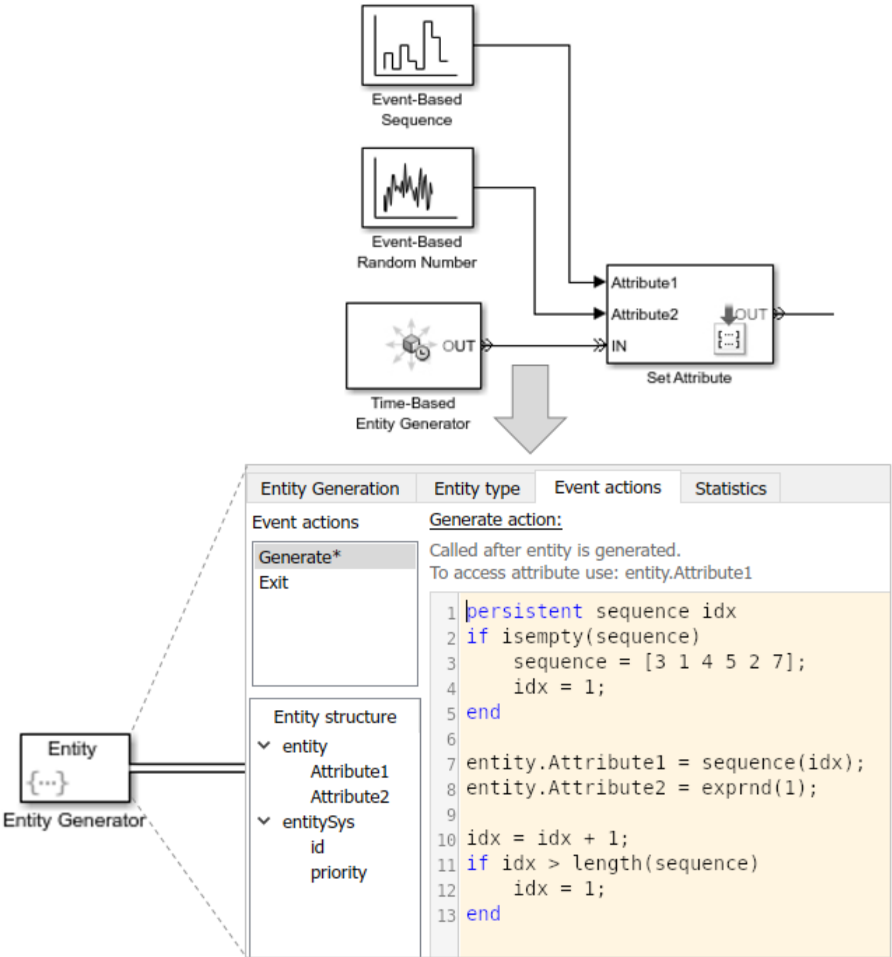
## Replace Set Attribute Blocks with Event Actions

Use these guidelines to replace Set Attribute blocks:

- If the Set Attribute blocks immediately follow entity generator blocks to initialize attributes, in the Entity Generator block, code the `Generate` action on the **Event actions** tab to set the attribute initial value. For example:

  ```
  entitySys.id=5;
  ```
- If the Set Attribute blocks change attributes, in the Entity Generator block, code the `Create` action on the **Event actions** tab.

This example illustrates the `Generation` action to initialize the attribute values:




Return to “Migration Workflow” on page 11-4.

## Get Attribute Values

If you write event actions to get attribute values, use a Simulink Function block:

1 Place the computation block in a Simulink Function block.
2 Pass the attribute value as an argument from the event action to the Simulink Function block.

## Replace Random Number Distributions in Event Actions

You can generate random numbers using:

- “Random Number Distribution” on page 11-16
- “Example of Arbitrary Discrete Distribution Replacement” on page 11-17

### Random Number Distribution

Replace Event-Based Random Number block random number distribution modes with equivalent MATLAB code in event actions. For more information about generating random distributions, see “Event Action Languages and Random Number Generation” on page 1-8.

If you need additional random number distributions, see “Statistics and Machine Learning Toolbox”.

Once you generate random numbers, return to “Migration Workflow” on page 11-4.

### Example of Arbitrary Discrete Distribution Replacement

Here is an example of how to reproduce the arbitrary discrete distribution for the Event-Based Random Number legacy block. Assume that the block has these parameter settings:

- **Distribution**: `Arbitrary discrete`
- **Value vector:** `[2 3 4 5 6]`
- **Probability vector:** `[0.3 0.3 0.1 0.2 0.1]`
- **Initial seed:** `12234`

As a general guideline:

1. Set the initial seed, for example:

    ```
    persistent init
    if isempty(init)
            rng(12234);
            init=true;
    end
    ```
2. Determine what the value vector is assigned to in the legacy model and directly assign it in the action code in the new model. In this example, the value vector is assigned to the `FinalStop`.
3. To assign values within the appropriate range, calculate the cumulative probability vector. For convenience, use the probability vector to calculate the cumulative probably vector. For example, if the probability vector is:

    ```
    [0.3 0.3 0.1 0.2 0.1]
    ```

    The cumulative probability vector is:

    ```
    [0.3 0.6 0.7 0.9 1]
    ```
4. Create a random variable to use in the code, for example:

    ```
    x=rand();
    ```

Here is example code for this example block to calculate the distribution. The value vector is assigned to `FinalStop`:

```
% Set initial seed.
persistent init
if isempty(init)
        rng(12234);
        init=true;
end
% Create random variable, x.
x=rand();
%
% Assign values within the appropriate range using the cumulative probability vector.
%
if x < 0.3
    entity.FinalStop=2;
elseif x >= 0.3 && x< 0.6
    entity.FinalStop=3;
elseif x >= 0.6 && x< 0.7
    entity.FinalStop=4;
elseif x >= 0.7 && x< 0.9
    entity.FinalStop=5;
else
```

```
    entity.FinalStop=6;
end
```

Once you generate random numbers, return to “Migration Workflow” on page 11-4.

## Replace Event-Based Sequence Block with Event Actions

Replace Event-Based Sequence blocks, which generate a sequence of numbers from specified column vectors, with event actions:

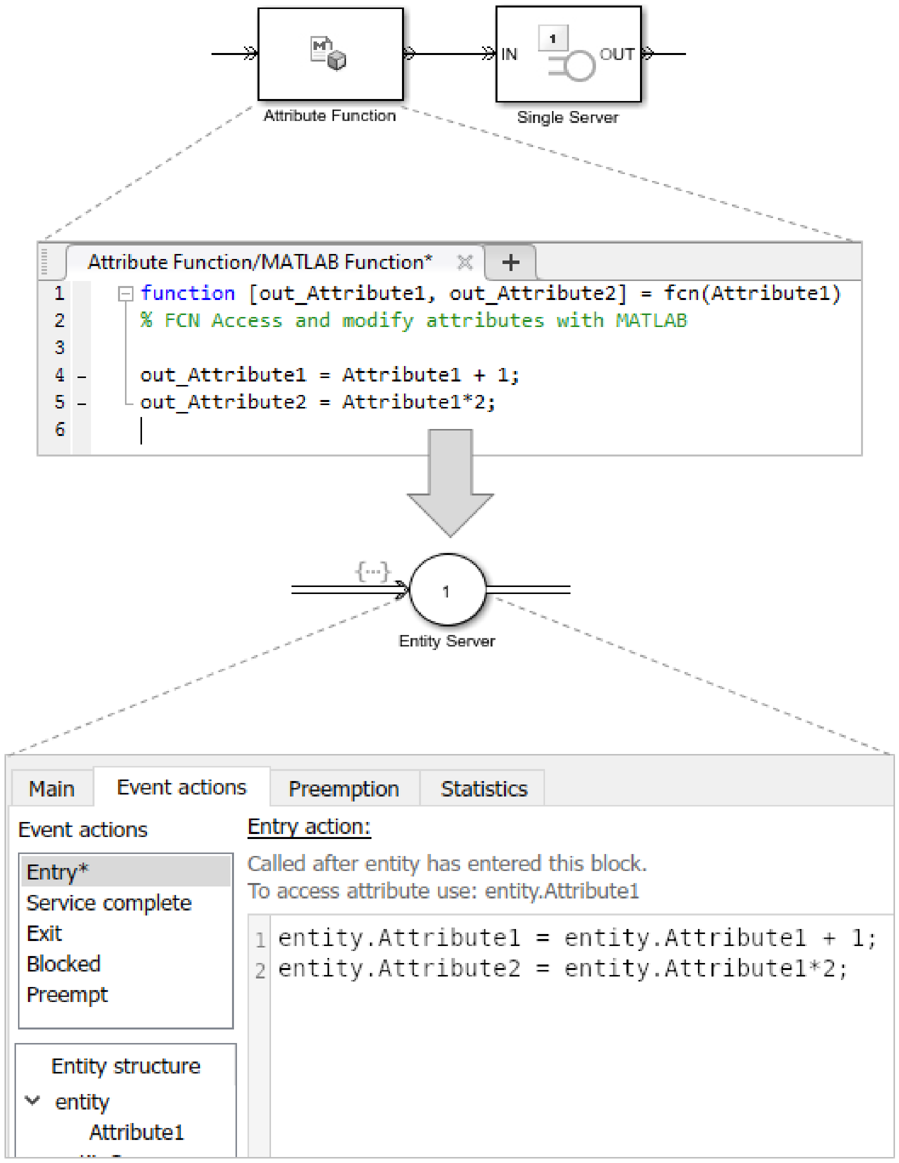
```
persistent sequence idx
if isempty(sequence)
    sequence = [3 1 4 5 2 7];
    idx = 1;
end

entity.Attribute1 = sequence(idx);

idx = idx + 1;
if idx > length(sequence)
    idx = 1;
end
```

## Replace Attribute Function Blocks with Event Actions

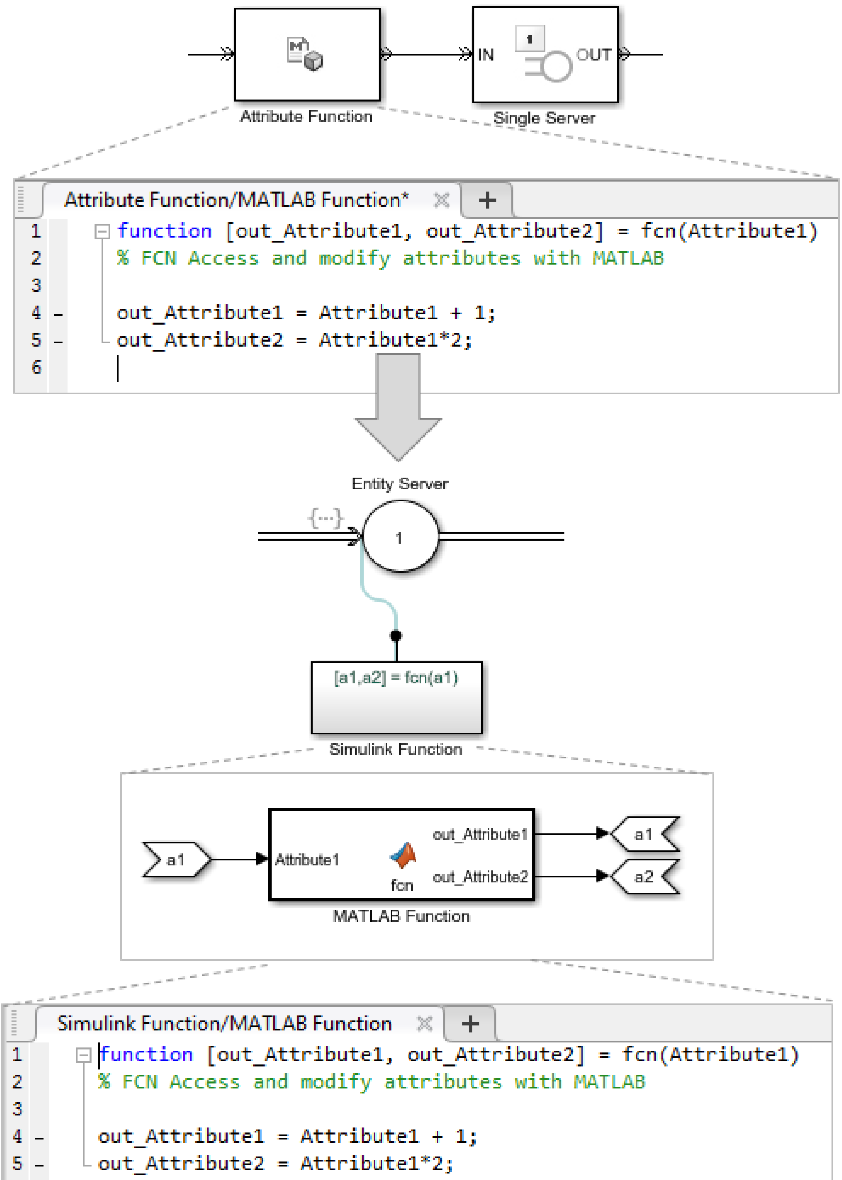
Replace Attribute Function blocks, which manipulate attributes using MATLAB code, with event actions:

1. Copy the Attribute Function code, without the function syntax, to the **Event actions** tab in the relevant event action.
2. To refer to the entity attribute, use the format `entity.`*`Attribute1`*.

For short or simple code, use constructs like these:




If you have longer or more complicated code, consider replacing the Attribute Function block with a Simulink Function and copying the code without modification into the Simulink Function block.



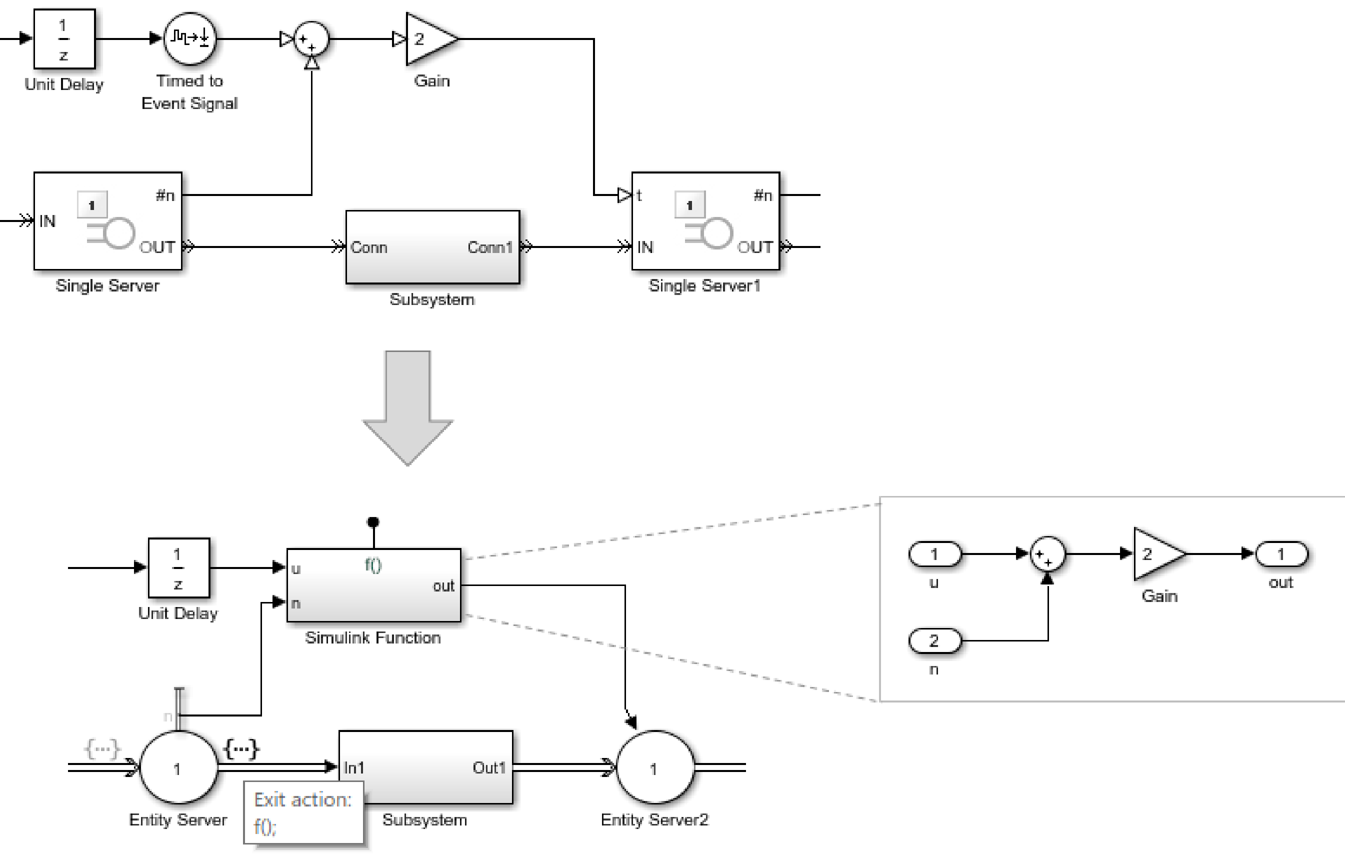

Return to “Migration Workflow” on page 11-4.

## If Using Simulink Signals in an Event-Based Computation

If you are using Simulink signals in an event-based computation, send the signals to a Simulink Function block.

1. Copy the event-based computation code to a Simulink Function block.
2. Send the Simulink signals as inputs to the Simulink Function block.

For example:




## See Also

### More About

- “Migration Considerations” on page 11-2
- “Migration Workflow” on page 11-4
- “Identify and Redefine Entity Types” on page 11-6
- “Replace Old Blocks” on page 11-8
- “Connect Signal Ports” on page 11-11
- “Observe Output” on page 11-22
- “Reactive Ports” on page 11-23

# Observe Output

Use these methods to observe output from your SimEvents model:

<table>
<tr><th>Items to Observe</th><th>Visualization Tool</th></tr>
<tr><td>Statistics</td><td rowspan="3"><ul><li>Simulation Data Inspector</li><li>Simulink To Workspace block</li><li>Simulink Scope block</li><li>Simulink Display block</li><li>Simulink To File block</li><li>Simulink Dashboard blocks</li></ul></td></tr>
<tr><td>Entities passing through model</td></tr>
<tr><td>Attributes</td></tr>
<tr><td>Count simultaneous entities and messages</td><td>Simulation Data Inspector</td></tr>
<tr><td>Count simultaneous events</td><td>Simulation Data Inspector — Each event is now a message reactive port</td></tr>
<tr><td>Entities moving through blocks in the model</td><td>Sequence Viewer</td></tr>
<tr><td>Entity animation</td><td>Animation — Highlight active entities in the simulation</td></tr>
<tr><td>Step through Simulation</td><td>Simulink Simulation Stepper</td></tr>
<tr><td>Custom animation</td><td>SimEvents custom visualization API.</td></tr>
</table>

Return to “Migration Workflow” on page 11-4.

## See Also

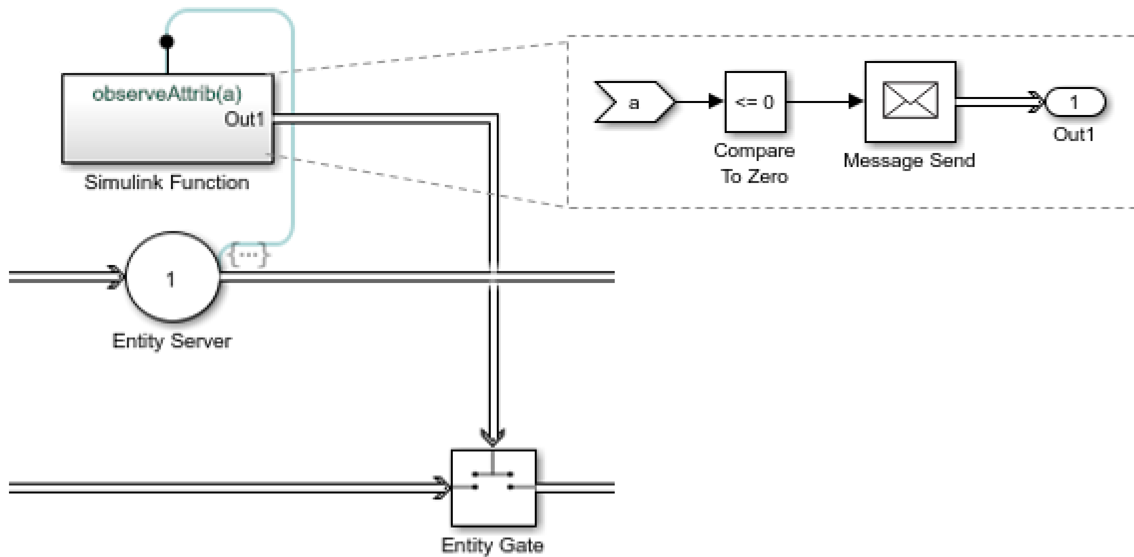
## More About

- “Migration Considerations” on page 11-2
- “Migration Workflow” on page 11-4
- “Identify and Redefine Entity Types” on page 11-6
- “Replace Old Blocks” on page 11-8
- “Connect Signal Ports” on page 11-11
- “Write Event Actions for Legacy Models” on page 11-15
- “Reactive Ports” on page 11-23

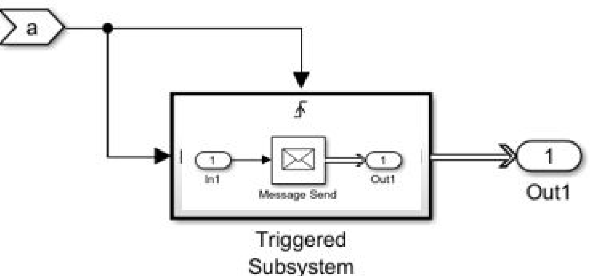
# Reactive Ports

In previous releases, reactive ports are signal input ports that listen for updates or changes in the input signal. When the input signal changes, an appropriate reaction occurs in the block possessing the port. Convert all reactive port event signals to messages.

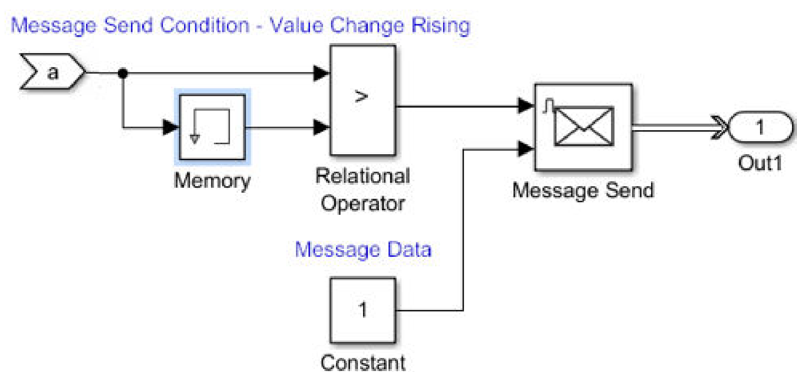
Here is an example of sending a message when data is less than or equal to 0.




Here is an example of sending messages on trigger edges (rising, falling, or either).




Here is an example of sending messages based on value changes (rising, falling, or either).



Here is a list of the reactive ports in SimEvents blocks and the action you can take for them.

**List of Reactive Ports**

| New Block with Reactive Port | Reactive Port Behavior | Action in New SimEvents Model |
|---|---|---|
| Entity Gate | To open a gate on an event | In enabled mode, send a message that carries a positive value to the port on the Entity Gate block.<br><br>In receive mode, send a message to advance one entity for each message that arrives on the control port. |
| Entity Input Switch<br><br>Entity Output Switch | Value change | To select a new port, send a message to the control port of the Entity Input Switch or Entity Output Switch. |
| Entity Generator | Message arrival | Send a message to create an event-based entity. |

Return to “Migration Workflow” on page 11-4

## See Also

## More About

- “Migration Considerations” on page 11-2
- “Migration Workflow” on page 11-4
- “Identify and Redefine Entity Types” on page 11-6
- “Replace Old Blocks” on page 11-8
- “Connect Signal Ports” on page 11-11
- “Write Event Actions for Legacy Models” on page 11-15
- “Observe Output” on page 11-22

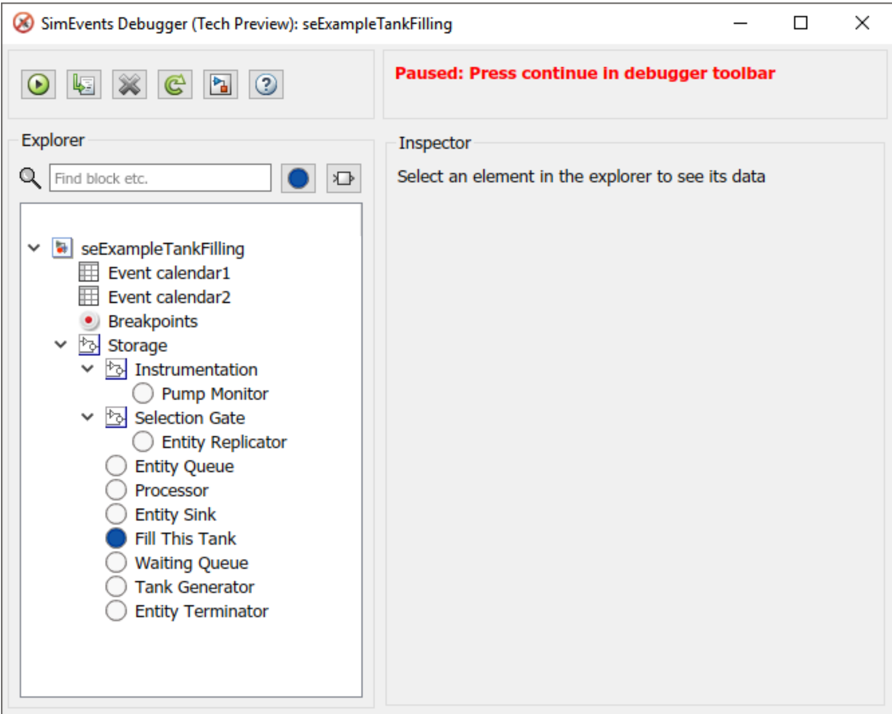
# 12

# Troubleshoot SimEvents Models

# Debug SimEvents Models

A breakpoint is a point of interest in the simulation at which the debugger can suspend the simulation. SimEvents Debugger allows you to inspect entities, set breakpoints based on entities leaving or entering storage elements, and step to events.

To enable debugging for a SimEvents model, add the SimEvents Debugger block to the model. When you click **Step Forward** in the Simulink Toolstrip, the SimEvents Debugger displays.




The **Explorer** pane contains these nodes:

- **Event calendar** — Maintains a list of current and pending events for the model. Select the **Break before event execution** check box to display event breakpoints on the **Breakpoints** node.
- **Breakpoints** — Lists the breakpoints previously set for the model. You can view breakpoints set for the block, on event calendar, and for watched entities.
- **Storage** — Displays the entity inspector listing all the storage blocks in the model and check boxes that let you select breakpoints. Blocks that contain entities are denoted with ●.

  To set breakpoints for post entry and pre-exit of entities, select the **PostEntry Break** and **PreExit Break** check boxes.
- *Entity Queue* — Displays the entity inspector listing the entities and attributes associated with that block.

SimEvents Debugger is used in the `Tank Filling Station` example to step through the model simulation, to set breakpoints, and to explore the event calendar.

The SimEvents software also provides an API that helps you to create your own visualization and debugging tools. For more information, see “Use SimulationObserver Class to Monitor a SimEvents Model” on page 10-2.

## Start the Debugger

1. Start `Tank Filling Station`.
2. Into the Simulink editor, add the SimEvents Debugger block at the top of the `Tank Filling Station` model.
3. To start the debugger, in the Simulink editor toolstrip, click the **Step Forward** button.

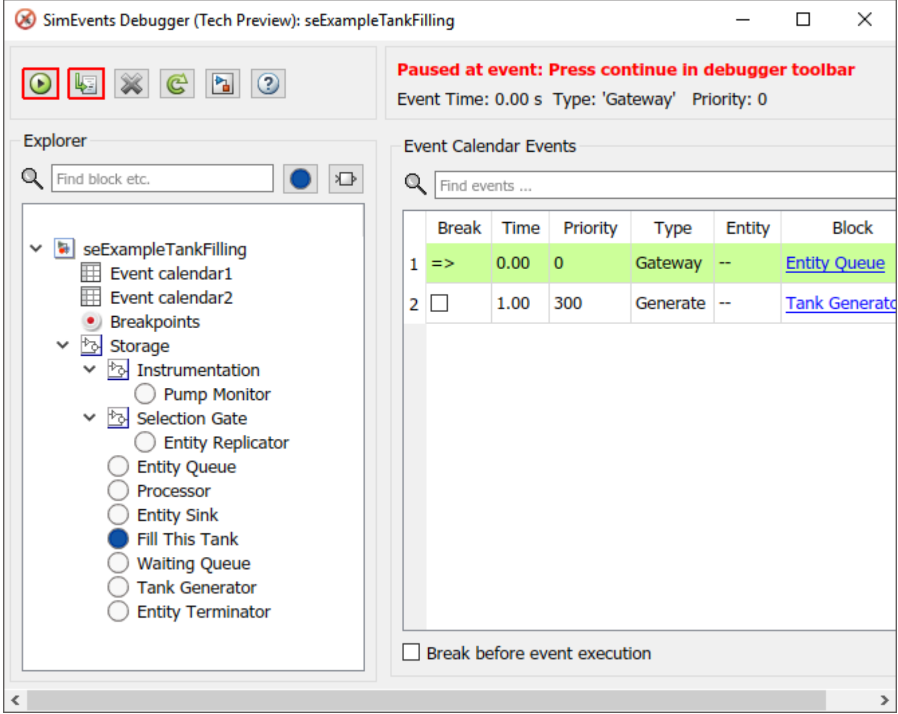
   The debugger displays in a paused state.
4. To step to the next event, click .

---

**Note** You can also click **Continue** () to have the debugger continue the simulation. However, doing so without setting breakpoints causes the simulation to complete and the debugger to close.

---

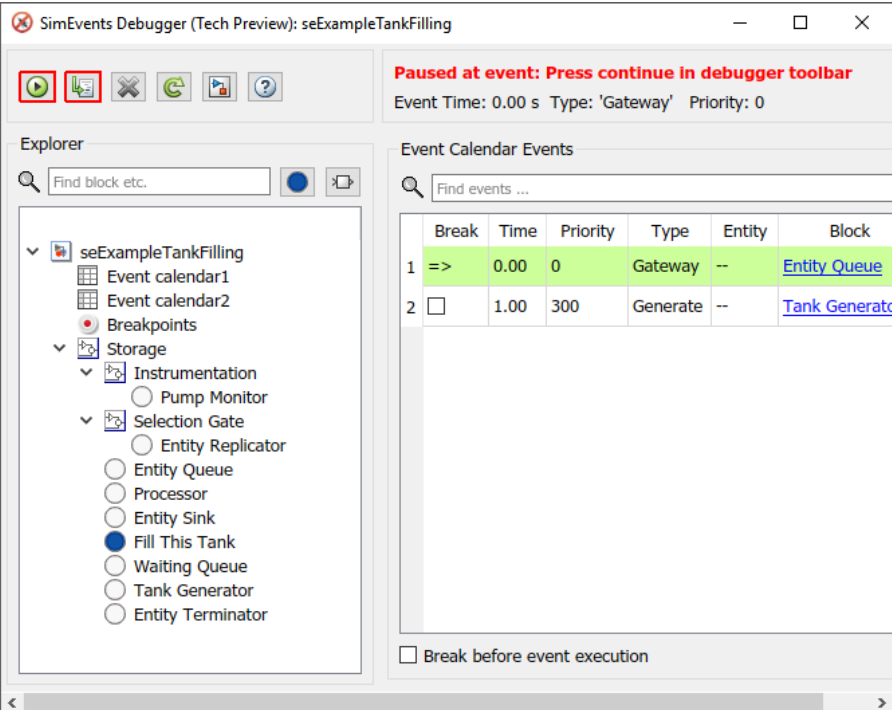
5. The debugger pauses at the next event and displays it in the event calendar. The current event is highlighted in green.



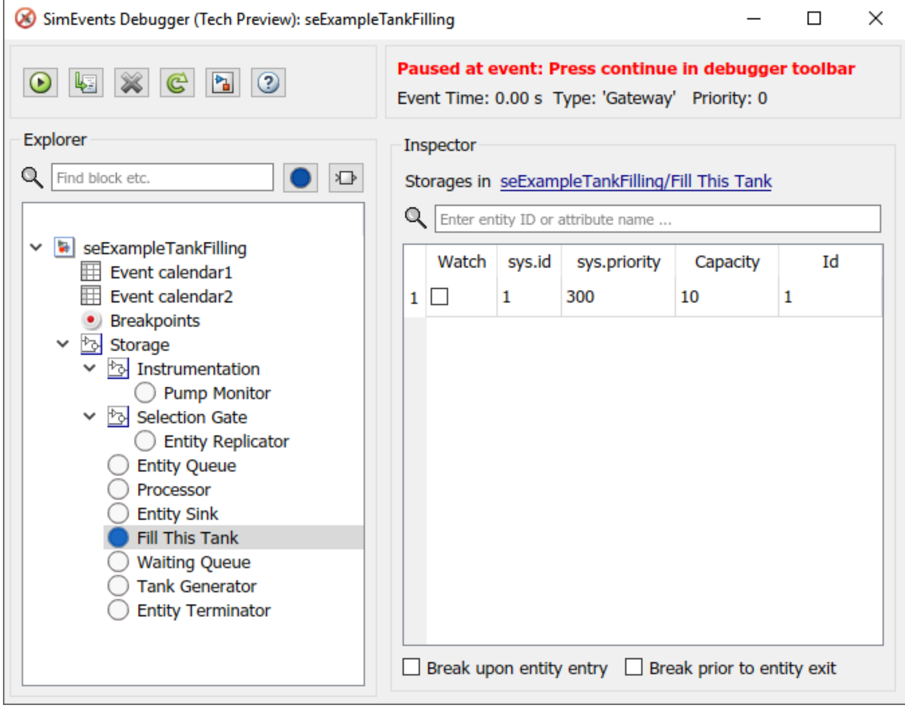

## Step Through Model

1. To look at the current and scheduled events, click the **Event calendar1** item. To set breakpoints, you can select the **Break before event execution** check box. The debugger hits the breakpoint

before the next scheduled event. This breakpoint is for any event type, including Forward, Generate, ServiceComplete, Gateway, Destroy, and Trigger. Do not select this check box now.




**2** To inspect the attributes of an entity, click the **Fill This Tank** storage element in the **Explorer** pane.



**3** The **Inspector** pane shows a table with the entity `sys.id`. To track the entity as the model simulates, click the associated check box.

**4** To set breakpoints for when this entity enters and leaves the block, at the bottom of the **Inspector** pane, select the two check boxes **Break upon entity entry** and **Break prior to entity exit**.

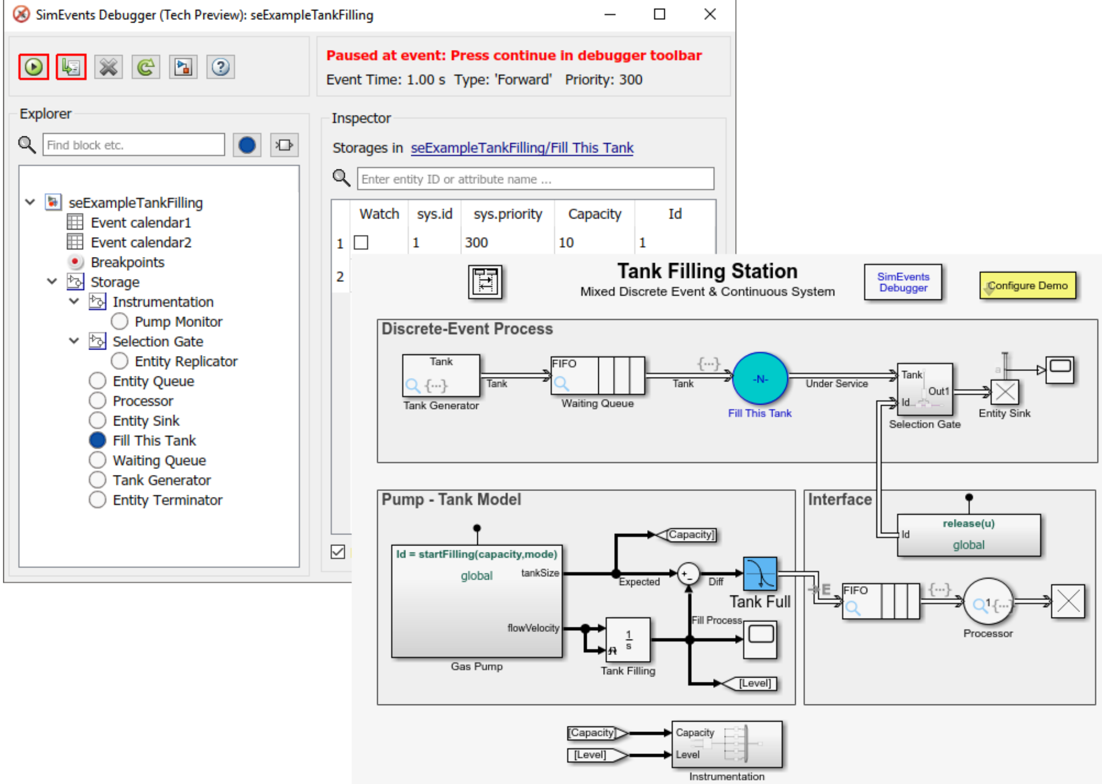
Alternatively, to set the breakpoints on storage blocks all at once, click the **Storage** item in the **Explorer** pane. Notice that the **Fill This Tank** block is highlighted because it contains entities.

Select the **PostEntry Break** check boxes for the blocks you want in this table.

**5**

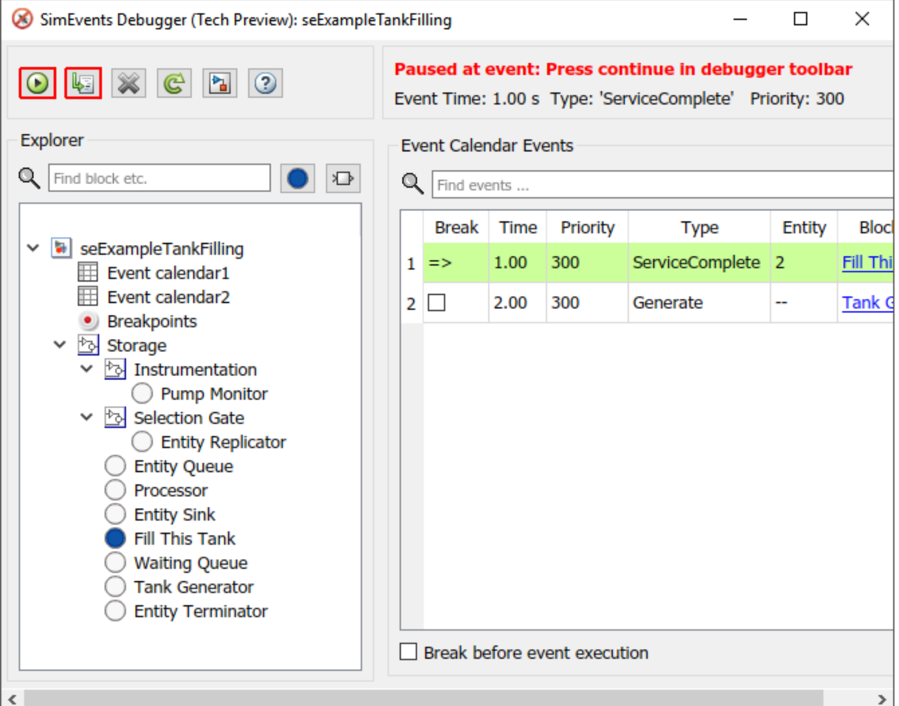
To progress to the next event, click .

**6** Click **Continue**. Simulation continues until the next `PostEntry` or `PreExit` event.



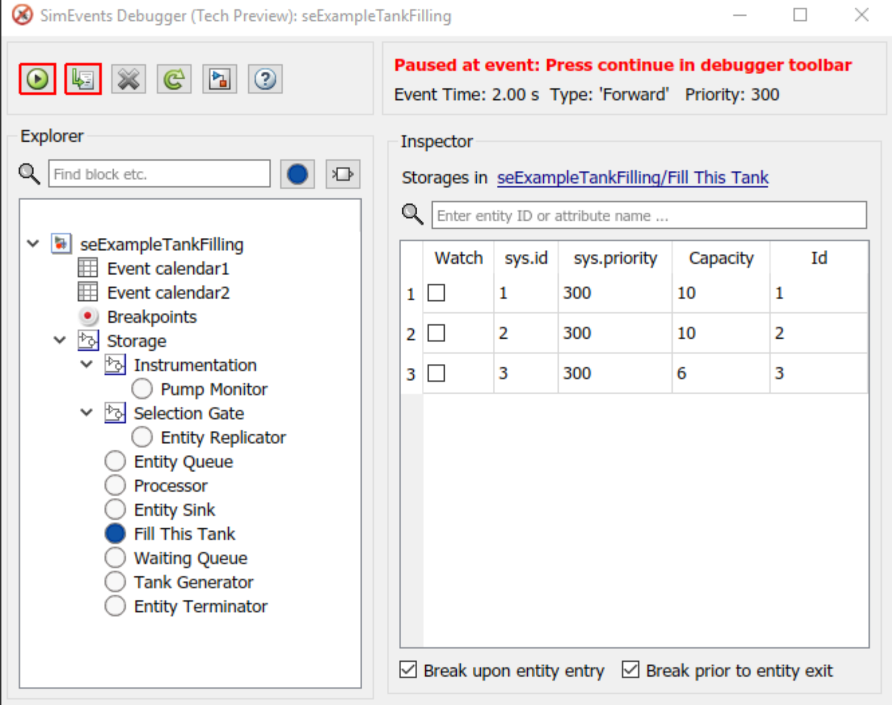

The block associated with the breakpoint is highlighted.

**7** Step to the next event.




The next breakpoint at which the debugger stops is highlighted in the event calendar.

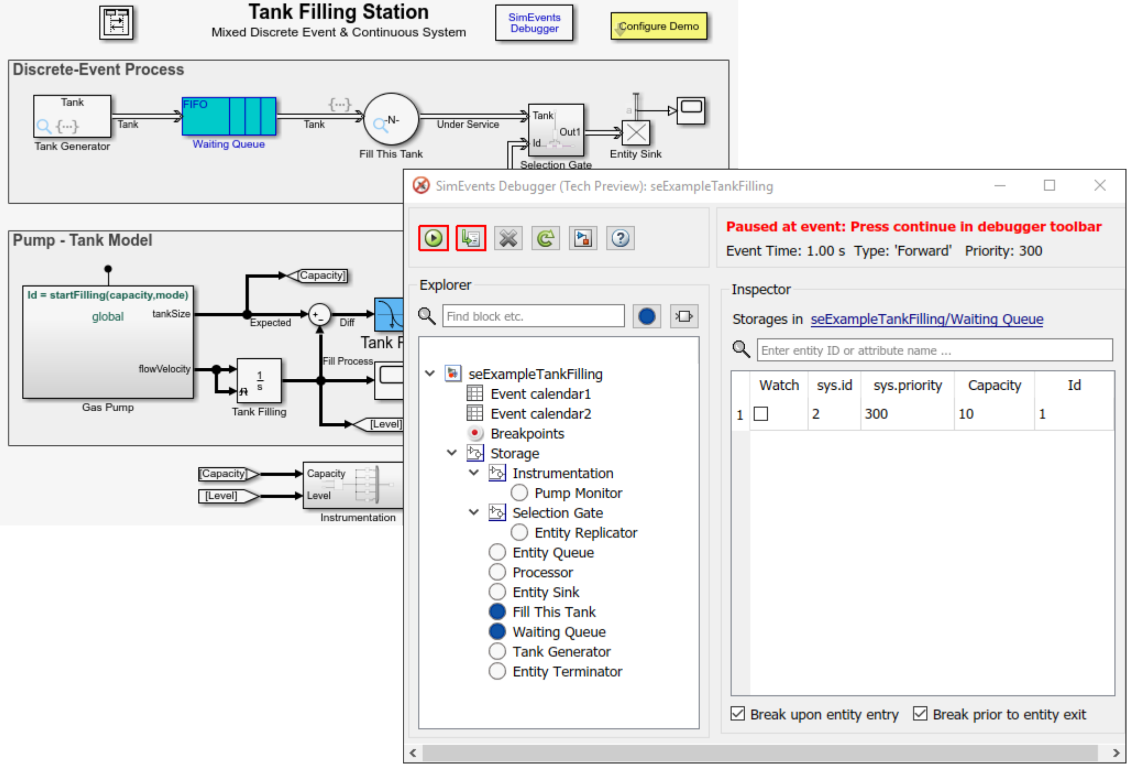
**8** Continue the simulation.



The simulation stops at the entity you opted to watch. As you continue the simulation or step through the model, the debugger stops at the various breakpoints and watchpoints that you set, letting you explore the model simulation.

**9** To inspect the entities in a currently selected block in the model, select the block in the model, then click the **Inspect GCB** button ( ).

The **Inspector** pane displays the current details of the entities in this block.




You can continue to set entity watchpoints and event breakpoints.

To list select blocks, events, or entities, type their names in the search boxes at the top of the **Explorer** or **Inspector** panes.

The SimEvents software also provides a programmatic interface that lets you create your own simulation observer or debugger. For more information, see “Create Custom Visualization”.

## See Also

SimEvents Debugger

## More About

- “Visualization and Animation for Debugging” on page 5-10
- “Observe Entities Using simevents.SimulationObserver Class” on page 10-5
- “Event Calendar” on page 6-3
- “Use SimulationObserver Class to Monitor a SimEvents Model” on page 10-2