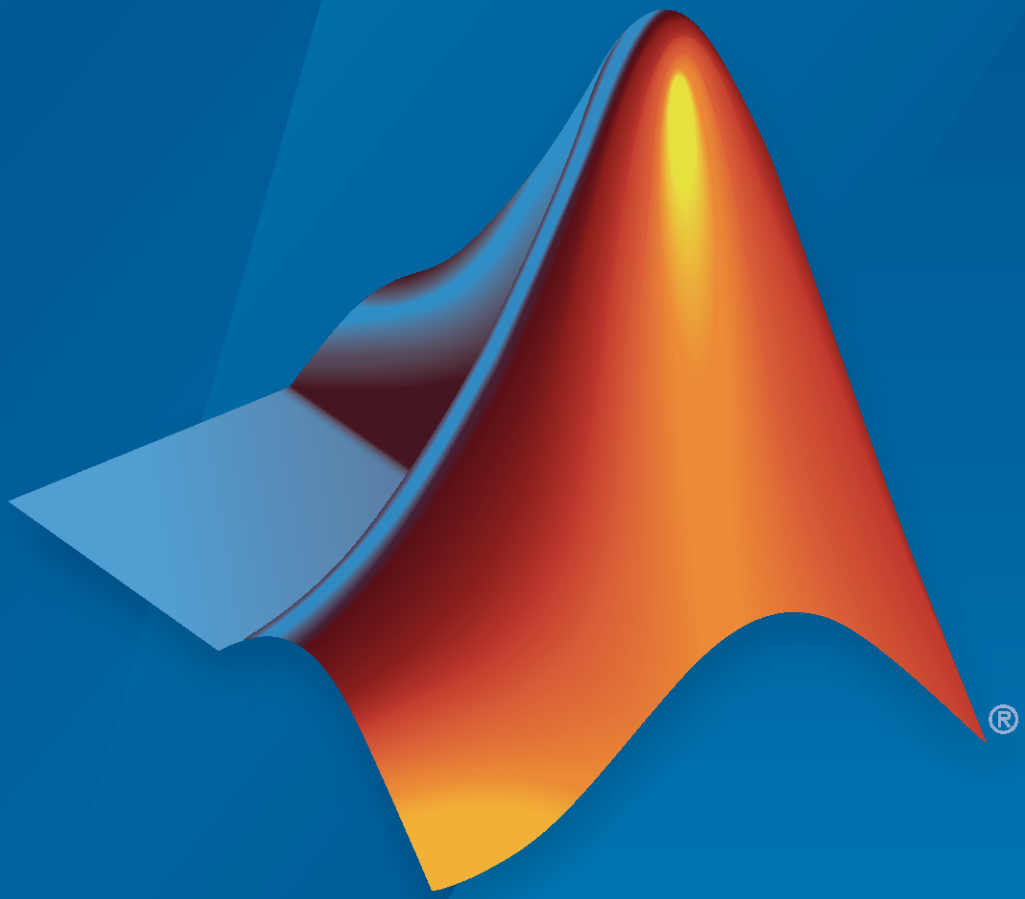
# SimEvents®

## Getting Started Guide



MATLAB® & SIMULINK®

R2020a

MathWorks®

# 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® Getting Started Guide*

**Revision History**

| | | |
|---|---|---|
| November 2005 | Online only | New for Version 1.0 (Release 14SP3+) |
| March 2006 | First printing | 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 | Second printing | 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) |

# Contents

## 1 Introduction



## 2 Build Simple Models with SimEvents Software

## 3 Key Concepts in SimEvents Software



## 4 Inspect Statistics



## 5 Simulate Multidomain Models

# Introduction

# SimEvents Product Description

**Model and simulate discrete-event systems**

SimEvents provides a discrete-event simulation engine and component library for analyzing event-driven system models and optimizing performance characteristics such as latency, throughput, and packet loss. Queues, servers, switches, and other predefined blocks enable you to model routing, processing delays, and prioritization for scheduling and communication.

With SimEvents, you can study the effects of task timing and resource usage on the performance of distributed control systems, software and hardware architectures, and communication networks. You can also conduct operational research for decisions related to forecasting, capacity planning, and supply-chain management.

## Key Features

- Discrete-event simulation engine for multidomain system models
- Entities with custom data attributes representing tasks, packets, and items
- Blocks for queuing, service, routing, resource management, multicasting, replication, and batching
- Statistics generation for delay, throughput, average queue length, and other metrics
- Library authoring with MATLAB® or Stateflow® for custom schedulers, hardware and software constructs, and communication channels
- Block diagram animation and inspection for visualizing model operation and debugging
- Custom animation creation for monitoring entities and events

# Discrete-Event Simulation in Simulink Models

| In this section... |
| --- |
|  |

SimEvents integrates discrete-event system modeling into the Simulink time-based framework. In time-based systems, a signal changes value in response to the simulation clock, and state updates occur synchronously with time. By contrast, in discrete-event or event-based systems state transitions depend on asynchronous discrete incidents called events.

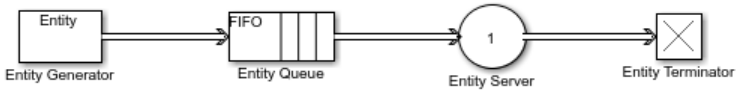
Suppose that you want to measure how long the average car waits in a queue for its turn to fill its tank at a busy gas station. Suppose that you also want to model the motion of the car by solving differential equations. You can use a combination of time-based simulation and discrete-event simulation, where:

- The time-based aspect controls the details of the car's trajectory
- The discrete-event aspect controls the queuing behavior

In a Simulink model, you typically construct a discrete-event system by adding various blocks, such as generators, queues, and servers, from the SimEvents block library. These blocks are suitable for producing and processing entities, which are abstractions of discrete items of interest. Examples of entities are vehicles arriving at a gas station, packets within a communication network, planes on a runway, or trains within a signaling system. Asynchronous events correspond to motion and changes in entity attributes through the system model, and they update the states of the underlying system. Examples of states are lengths of queues or service time for an entity in a server.

## A Simple Queuing System

This SimEvents model represents a simple queuing system that generates entities of interest and queues them in a specified order, services them to change their attributes, and terminates them to represent their departure from the line. To learn how to build this model, see “Create a Discrete-Event Model” on page 2-2.

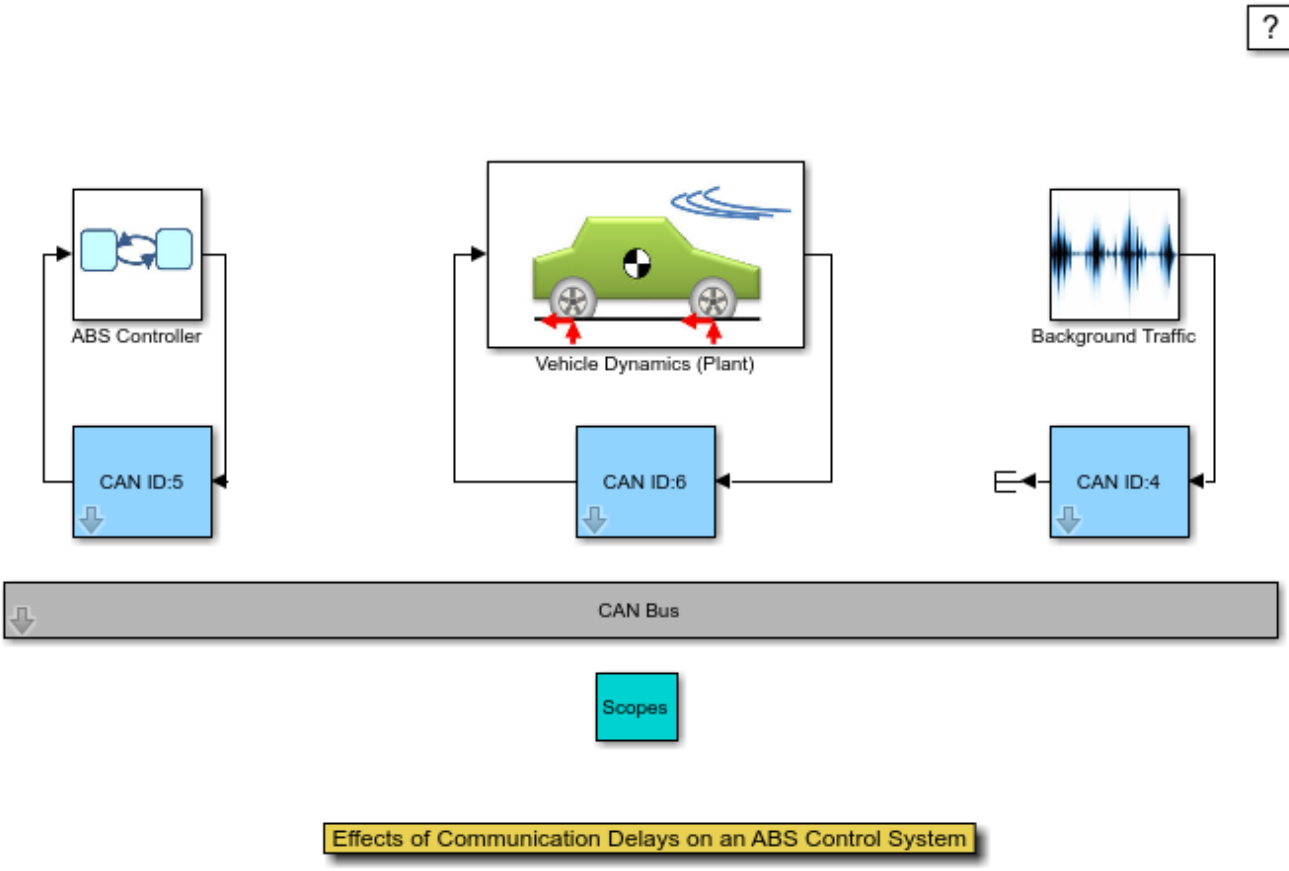



The Entity Generator block is used to generate entities with a fixed or randomized intergeneration time. The Entity Queue block queues the entities based on a specified order. The Entity Server block services entities for a length of time. The entities depart the line through the Entity Terminator block.

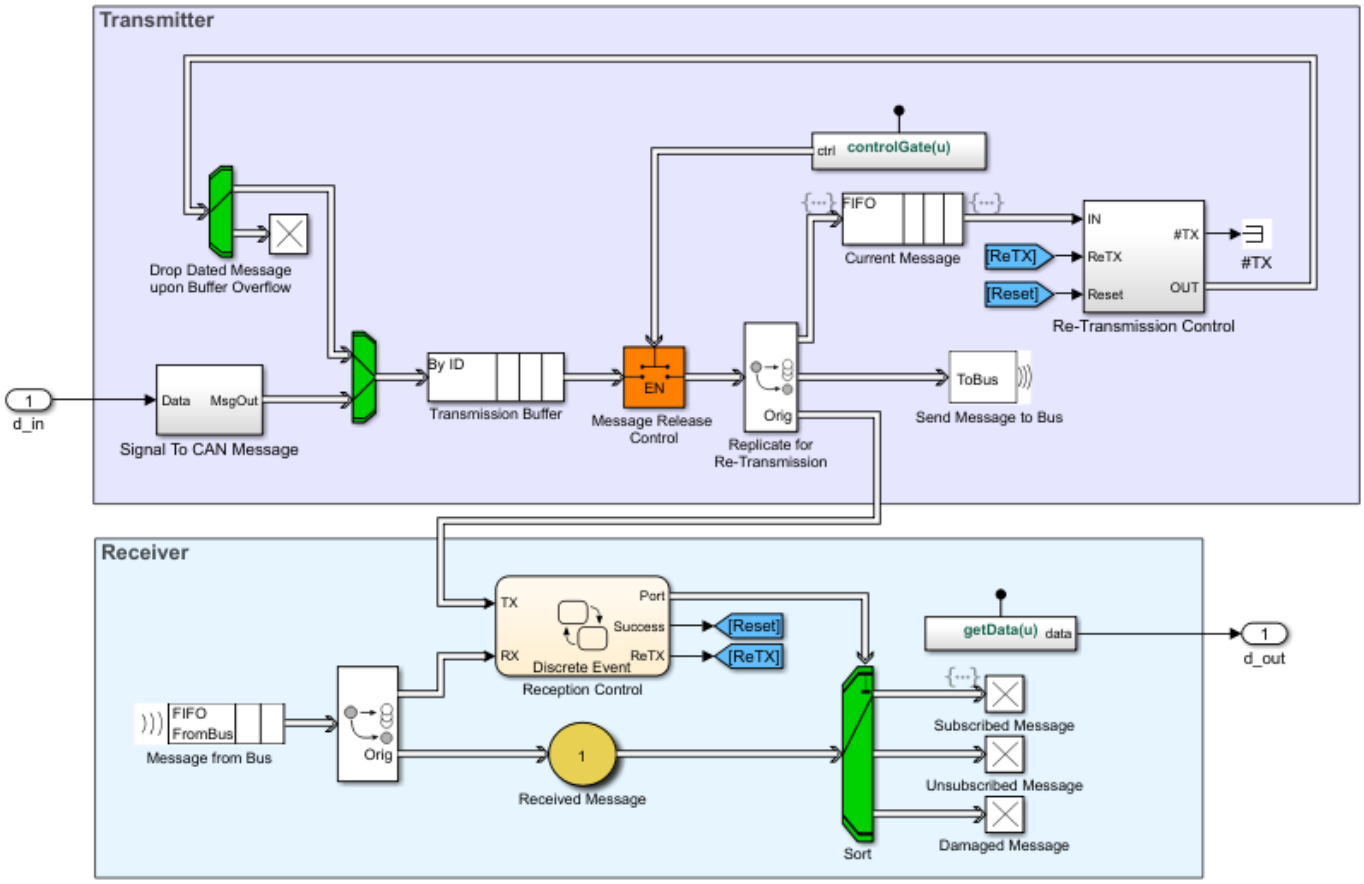
## Modeling Communication Delay on an Anti-Lock Braking System

The seExampleCanBus model provides a scenario for investigating the communication delay in a car anti-lock braking system (ABS). The system uses control area network (CAN) communications between components. The model illustrates a heavily loaded network of a distributed system.

The model investigates the delay of communication between the ABS controller and the vehicle in ideal conditions and in the presence of noise.

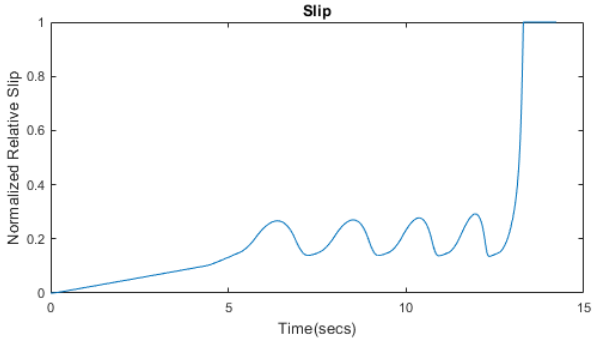


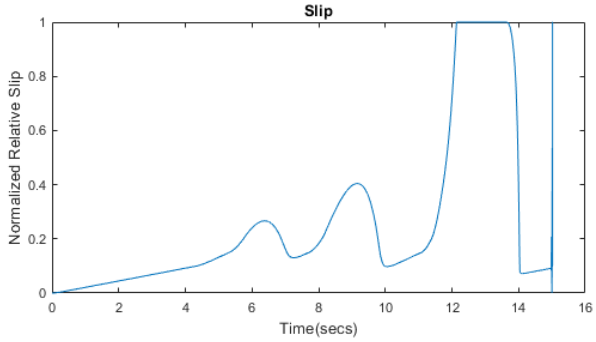

The CAN ID:5 subsystem consists of SimEvents library blocks that model a buffer in transmission, message queues, and replicated messages for communication.




The model is used to analyze the effect of communication delay on the slip value with the passage of time. The slip value is `0` when the wheel speed and the vehicle speed are equal. The slip value is `1` when the wheels are completely locked. The desirable slip value is `0.2`.

The plot on the left represents the slip in ideal conditions and on the right is the slip in the presence of noise. The decrease in slip performance is detected in the model and resolved with reprioritization of CAN messages.




For more information about the model, see “Effects of Communication Delays on an ABS Control System”.

## Modeling a Hybrid System with Event-Based and Time-Based Components

One or more discrete-event systems can coexist with time-based systems in a Simulink model. This coexistence facilitates the simulation of sophisticated hybrid systems. You can pass signals from time-based components/systems to and from discrete-event components/systems modeled with SimEvents blocks. The combination of time- and event-based modeling facilitates the simulation of large-scale systems that incorporate smaller subsystems from multiple environments. An example of a large-scale system has physical modeling for continuous-time systems, such as electrical systems, which communicate via a channel modeled as a discrete-event system. A Simulink model can also contain a purely discrete-event system with no time-based components when modeling event-based processes. These systems are common in models that represent logistic and manufacturing systems.

The seExampleTankFilling model incorporates both time-based and event-based modeling to represent vehicles queuing up to fill their tanks in a gas station.

The SimEvents part is an extension of the model presented in “A Simple Queuing System” on page 1-3 and it models the flow of the vehicle tanks. The tanks are generated, queued, and serviced to be filled. The Simulink part models the logic to fill the tanks. When a tank is filled to capacity, the completion of the tank filling process is detected and a message is sent to the SimEvents part to open the gate for releasing the tank. For more information, see “Modeling Hybrid Systems - Tank Filling”.

## See Also

## Related Examples

- “Create a Discrete-Event Model” on page 2-2
- “Create a Hybrid Model with Time-Based and Event-Based Components” on page 5-2
- “Effects of Communication Delays on an ABS Control System”

## More About

- “SimEvents Product Description” on page 1-2
- “Entities in a SimEvents Model” on page 3-2
- “Events and Event Actions”
- “Bibliography” on page 1-12

## External Websites

- Tech Talks: Understanding Discrete-Event Simulation

# Related Products

| In this section... |
| --- |
|  |

## Information About Related Products

See Related Products (`https://www.mathworks.com/products/simevents/related.html`).

## Limitations on Usage with Related Products

### Code Generation

SimEvents blocks do not support code generation using the Simulink Coder™ product in version 5.0 (R2016a). Before version 3.1.2 (R2010a), SimEvents blocks offered limited code generation support for rapid simulation. Since version 4.0 (R2011b), SimEvents blocks do not support code generation using the Simulink Coder product. Support for rapid simulation was removed because the improvements in normal model simulation performance for SimEvents models matched or surpassed the performance of rapid simulation in releases before version 4.0.

### Simulation Modes

SimEvents blocks do not support simulation using the Rapid Accelerator, Accelerator, Processor-in-the-Loop (PIL), or External mode.

### Model Reference

SimEvents blocks cannot be in a model that you reference through the Model block.

## See Also

## Related Examples

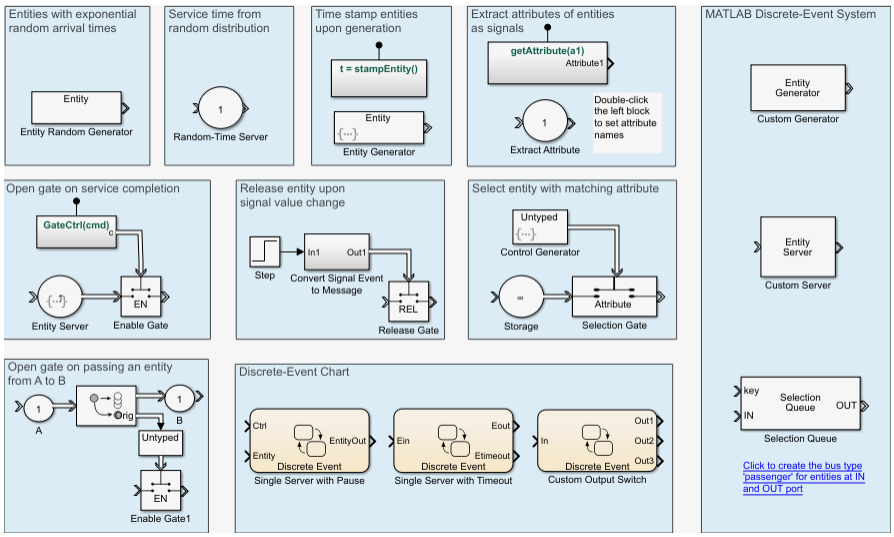
- “Create a Hybrid Model with Time-Based and Event-Based Components” on page 5-2

## More About

- “SimEvents Product Description” on page 1-2
- “Discrete-Event Simulation in Simulink Models” on page 1-3

# SimEvents Common Design Patterns

The SimEvents library provides design patterns that you can refer to while modeling. To access these patterns, open the SimEvents library and double-click the Design Patterns block.




Consider these design patterns while modeling:

| Design Pattern | Description | Input Specifications | Output Specifications | Application |
|---|---|---|---|---|
| Entities with exponential random arrival times | Generates entities with random interval time in exponential distribution fashion. | Not applicable | Structured entity with specified attributes | Model:<br>• Customers entering a store<br>• Incoming phone calls of a hotline |
| Service time from random distribution | Specifies waiting time in the Entity Server as a random number uniformly distributed from 0 through 1. | Any entity type | Inherited from the input | Model:<br>• Extension of an event that is random within a range (for example, length of a call<br>• Purposeful holding of an entity for a random time |

| Design Pattern | Description | Input Specifications | Output Specifications | Application |
|---|---|---|---|---|
| Extract attributes of entities as signals | Extracts one or more attributes of entities as signals. | A structured entity or bus object with specified attribute | `getAttribute` — Real double scalar signal<br><br>Extracted Attribute — Inherited from the input | Inspect or use a specific entity attribute |
| Timestamp entities upon generation | Generates entities with an attribute **TimeStamp** that records the simulation time upon generation. | Not applicable | Structured entity with attributes `Data` and `TimeStamp` | Use when generation time of entities is needed, for example, when calculating the priority in a combined scheduling algorithm. |
| Release entity upon signal value change | Releases an incoming entity when there is a jump in the step function. | Any entity type | Inherited from the input | Use to control the passing of entities based on the change of a function. |
| Open gate on service completion | Upon service completion, the gate opens and releases an entity. | Any entity type | Inherited from the input | Use task completion to trigger entity processing. |
| Sense an entity passing from A to B and open a gate | Passing an entity from *A* to *B* opens the gate and releases an entity. | Any entity type | Inherited from the input | Use to model the passing of an entity in one route to control the passing of another route. |
| Select an entity with a matching attribute | Select entities to advance whose specified attributes are matching the anonymous entity at the control port | A structured entity or bus object with a specified attribute | Inherited from the input | Select entities with a specified attribute to output |
| Discrete Event Chart: Single Server with Pause | A Ctrl message triggers pause of service for the incoming entity. A second Ctrl message continues the service. Entity data conveys the service time. | Ctrl — Anonymous entity specifying the pause and resume<br><br>Entity — Anonymous entity specifying service time | Inherited from the input | Use external events or signals to pause the service of entities. |

| Design Pattern | Description | Input Specifications | Output Specifications | Application |
|---|---|---|---|---|
| Discrete Event Chart: Single Server with Timeout | If the service time (which is random) exceeds the timeout limit specified by the entity data, the entity leaves the server. | Anonymous entity with specified timeout limit | Inherited from the input | Model:<br>• A protocol that explicitly calls for timeouts.<br>• Implementation of special routing or other handling of entities that exceed a time limit.<br>• Entities that represent perishable items. |
| Discrete Event Chart: Custom Output Switch | Randomly routes entities to one of the three output ports. | Anonymous entity | Inherited from the input | Implement a more complicated routing algorithm for an output switch. |
| MATLAB Discrete Event System: Custom Generator | The Custom Generator block, defined using the MATLAB Discrete Event System block, is a basic entity generator. The generator block requires specification of generation period. | Not applicable | Anonymous entity | Implement a more complicated entity generator. |
| MATLAB Discrete Event System: Custom Server | Custom Server block, defined using the MATLAB Discrete Event System block, is a basic entity server. The server block requires specification of server number and service time. | Any entity type | Inherited from the input | Implement a more complicated entity server. |

| Design Pattern | Description | Input Specifications | Output Specifications | Application |
|---|---|---|---|---|
| MATLAB Discrete Event System: Selection Queue | The Selection Queue block, defined using the MATLAB Discrete Event System block, stores entities of bus type `passenger` arriving at the IN port. Keys from the call port select `passenger` entities with the matching `trainNum` field and send them to the OUT port. | Key — Anonymous entity carrying the selection key<br><br>IN — A structured entity or bus object with specified attribute | Inherit from IN | Select a specific entity to output from a queue. |

## See Also

Discrete Event Chart | MATLAB Discrete Event System

## More About

- “Block Authoring”
- “Discrete-Event System Objects”
- “Implement Discrete-Event Systems with Charts”

# Bibliography

[1] Banks, Jerry, John Carlson, and Barry Nelson. *Discrete-Event System Simulation*, Second Ed. Upper Saddle River, N.J.: Prentice-Hall, 1996.

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

[3] Cassandras, Christos G., and Stéphane Lafortune. *Introduction to Discrete Event Systems*. Boston: Kluwer Academic Publishers, 1999.

[4] Fishman, George S. *Discrete-Event Simulation: Modeling, Programming, and Analysis*. New York: Springer-Verlag, 2001.

[5] Gordon, Geoffery. *System Simulation*, Second Ed. Englewood Cliffs, N.J.: Prentice-Hall, 1978.

[6] Kleinrock, Leonard. *Queueing Systems, Volume I: Theory*. New York: Wiley, 1975.

[7] Law, Averill M., and W. David Kelton. *Simulation Modeling and Analysis*, 3rd Ed. New York: McGraw-Hill, 1999.

[8] Moler, C. “Floating points: IEEE Standard unifies arithmetic model,” Cleve's Corner. The MathWorks, Inc., 1996. https://www.mathworks.com/company/newsletters/news_notes/pdf/Fall96Cleve.pdf.

[9] Watkins, Kevin. *Discrete Event Simulation in C*. London: McGraw-Hill, 1993.

[10] Zeigler, Bernard P., Herbert Praehofer, and Tag Gon Kim. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Second Ed. San Diego: Academic Press, 2000.

# Build Simple Models with SimEvents Software

# Create a Discrete-Event Model

| In this section... |
| --- |
|  |
|  |
|  |
|  |

This example describes how to build a new SimEvents model representing a discrete-event system. For more information about discrete-event systems, see “Discrete-Event Simulation in Simulink Models” on page 1-3. The example features a simple queuing system in which trucks arrive at a gas station to fill up their tanks. The tank of a truck is represented by an entity that arrives at a fixed deterministic rate, waits in a queue, and advances to a server that fills the tanks and also operates at a fixed deterministic rate. This type of system is known as a D/D/1 queuing system in queuing notation. The notation indicates the deterministic arrival rate, the deterministic service rate, and a single server.

The example shows how to perform basic model-building tasks in SimEvents— adding blocks to models and configuring blocks.

To open the model directly without performing the steps, see “A Simple Queuing System” on page 2-3.

## Add SimEvents Blocks to a Model

1. Open a new model window.

   On the **Home** tab, select **New > Simulink Model** and select **Blank Model**. Save the model in your working folder as `dd1`.

2. Open the SimEvents library.

   In the MATLAB Command Window, enter

   ```
   simevents
   ```

   The main SimEvents library window appears with the blocks it contains.

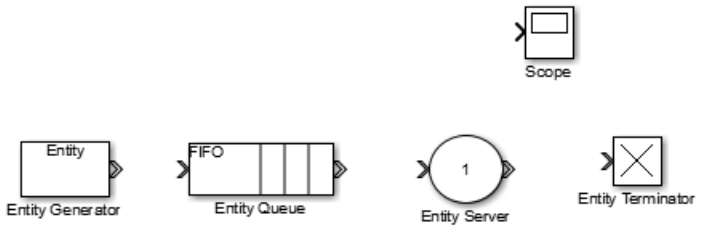
3. Add blocks to the model.

   From the SimEvents library, drag these blocks to the model.

   - Entity Generator — Generates entities to model the arrival of tanks.
   - Entity Queue — Stores entities in a queue to model the queuing of tanks waiting to be filled.
   - Entity Server — Serves entities to model the tank filling process.
   - Entity Terminator — Terminates entities to model the tanks' departure from the station.

   In the model window, double-click and type the name of the Scope block. Press Enter to add it.

   The added blocks represent the key processes in the simulation: generating entities, storing entities in a queue, serving entities, and creating a plot that shows relevant data.




## Configure Blocks

Each block in a model, in this case, `dd1`, has a dialog box that enables you to specify block parameters. Default parameter values might or might not fit your case, depending on your modeling needs.

Two important parameters in the D/D/1 queuing system are the arrival rate and service rate. The reciprocals of these rates are the duration between successive entities and the duration of service for each entity. To examine these durations:

1. Double-click the Entity Generator block. Observe that the **Period** parameter is set to `1`. This means that the block generates a new entity every second. A tank arrives at the station every second.
2. Double-click the Entity Server block. Observe that the **Service time** parameter is set to `1.0`. This means that the server spends one-second processing each entity that arrives at the block. Each tank is filled for one second duration.

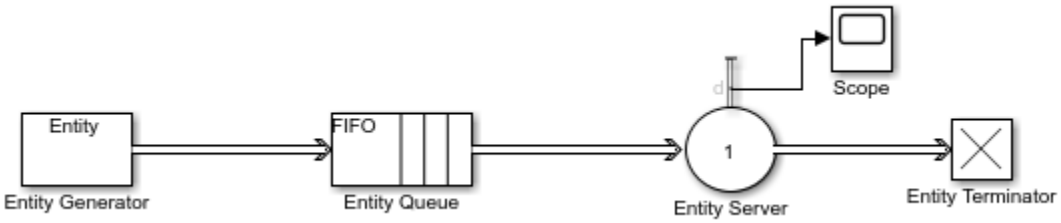
   The **Period** and **Service time** parameters have the same value, which means that the server completes servicing the entity at the same time that a new entity is being created.
3. Click **Cancel** in both dialog boxes to close them without changing any parameters.
4. Double-click the Entity Server block. Click the **Statistics** tab to view parameters related to the statistical reporting of the block. Select **Number of entities departed, d**. Click **OK**.

   The Entity Server block acquires a signal output port labeled **d**. During the simulation, the block produces an output signal at this **d** port. The value of the signal is the running count of entities that have completed their service and departed from the server.
5. Connect the Scope block to the **Number of entities departed, d** and display the statistics (running count of entities).
6. Double-click the Entity Queue block. Set the **Capacity** parameter to `Inf` to create a queue with infinite capacity and click **OK**.
7. Connect the blocks as shown and save the `dd1` model you have created. The entity path lengths do not affect the simulation.

   SimEvents connects the source block to the destination block. If necessary, the software also routes the connecting line around intervening blocks or lines.
8. Simulate the model.

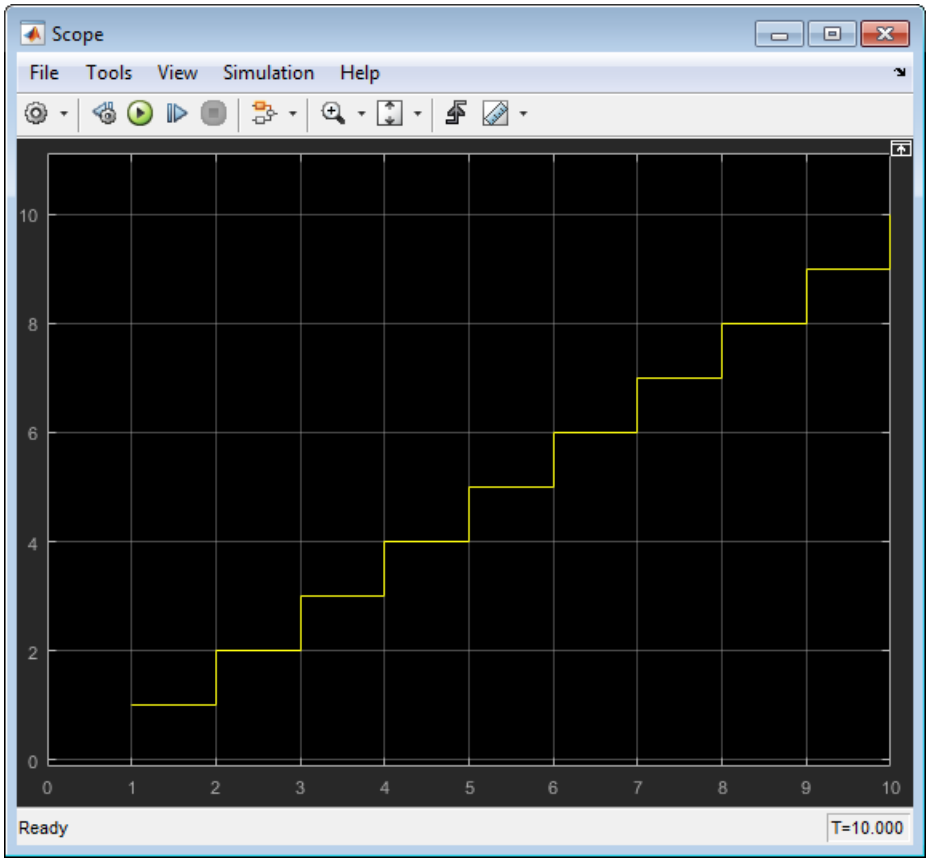
## A Simple Queuing System

Open the example to investigate a simple queuing system that generates, queues, services, and terminates entities.




## Results of the Simulation

When the simulation runs, the Simulink Scope block opens a window containing a plot. The horizontal axis represents the times at which entities depart from the server, while the vertical axis represents the total number of entities that have departed from the server.




After an entity departs from the Entity Server block, the block updates its output signal at the **d** port.

## See Also

Entity Generator | Entity Queue | Entity Server | Entity Terminator

## Related Examples

- “Explore Statistics and Visualize Simulation Results” on page 2-6

- “Manage Entities Using Event Actions” on page 2-13
- “Model Basic Queuing Systems”
- “Create a Hybrid Model with Time-Based and Event-Based Components” on page 5-2

## More About

- “Entities in a SimEvents Model” on page 3-2

# Explore Statistics and Visualize Simulation Results

The main purpose of creating a discrete-event simulation is to understand the underlying system or inform decisions about the underlying system.

Statistical data gathered during simulation can be important for interpreting the behavior of a model. For example:

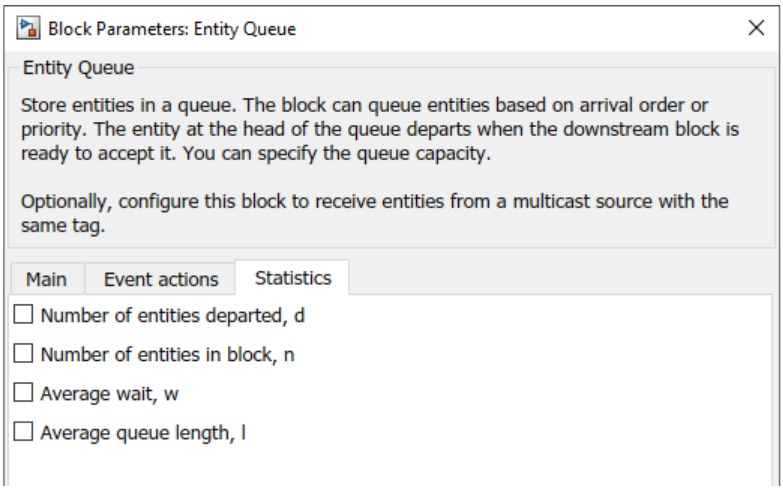
- If you simulate the operation and maintenance of equipment on an assembly line, you can use the computed production and defect rates to help decide whether to change your maintenance schedule.
- If you simulate a communication bus under varying bus loads, you might use computed average delays in high- or low-priority messages to help determine whether a proposed architecture is viable.

The number of entities departing a block, the average wait time of entities, utilization, and the average number of entities being served in an Entity Server block are a subset of statistics you would want to visualize.

Many SimEvents blocks have a **Statistics** tab, from which you can select the relevant data.

This procedure shows you how to access a statistical output signal for a given SimEvents block.

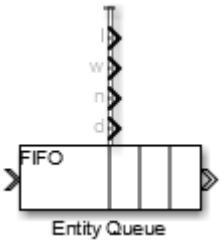
1. Determine which statistical output signal you want to access and find the associated parameter in the block dialog box. To see which statistics are available, open the block dialog box. The list of available statistics appears as a list of parameters on the **Statistics** tab of the dialog box.




2. Select the check box. After you apply the change, the block has a new signal output port corresponding to that statistic.

   For example, the Entity Queue block can display:

   - Number of entities departed, d
   - Number of entities in the block, n
   - Average wait time of the entities, w
   - Average queue length of entities, l




3 To display the statistics, connect those signal output ports to a Simulink Scope block.

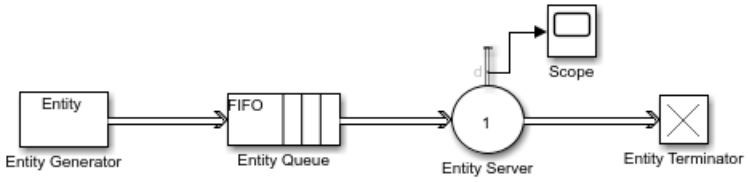
> **Note** Use scopes and other observer blocks to observe individual statistic ports. However, you cannot use the same scope to observe multiple statistics ports nor use a Scope Viewer for a statistics port. To observe multiple statistic ports, consider using a dashboard or the Simulation Data Inspector.

See “Visualization and Animation for Debugging” for a table showing all the visualization tools.

You can use the built-in statistical signals from SimEvents blocks to derive more specialized or complex statistics. One approach is to use a Simulink Function block, and another approach is to compute statistics using MATLAB code after the simulation is complete. For more information about using statistics for run-time control, see “Interpret SimEvents Models Using Statistical Analysis”. For an example to save statistics data to workspace, see “Optimize SimEvents Models by Running Multiple Simulations”.

## Explore a D/D/1 System Using Plots

This example shows how to modify a simple queuing system and plot statistical quantities to interpret its behavior. In the example, a `dd1` queuing model, which represents the tank filling process of the vehicles arriving at a gas station, is used to view the statistics for entity waiting time and server utilization. For more information about the `dd1` queuing model, see “Create a Discrete-Event Model” on page 2-2.




To open the model directly without performing the configuration steps, see “Visualize and Explore Simulation Results” on page 2-8.

### View Statistics for Waiting Times and Utilization

The queue length is an example of a statistic that quantifies a state at a particular instant. Other statistics, such as average waiting time and server utilization, summarize behavior between `simtime=0` and the current time. Take these steps to modify the model so that you can view the average waiting time of entities in the queue and server, and the proportion of time that the server spends storing an entity.

1. Double-click the Entity Queue block. Set **Capacity** to `Inf`. Click the **Statistics** tab, set **Average wait** to `On`, and click **OK**.

   An output port, **w**, representing the average duration that entities wait in a queue appears.Connect the statistic to a scope block and rename it to Average Wait Queue.

2. Double-click the Entity Server block. Click the **Statistics** tab, set both the **Average wait** and **Utilization** parameters to `On`, and click **OK**.

   Two output ports, **w** and **util** appear. **w** represents the average duration that entities wait in the server. **util** represents the proportion of time that the server spends storing an entity.

3. Add two Scope blocks. Rename all the Scope blocks with descriptive names, for example, Utilization, Number of entities departed, Average Wait Server.

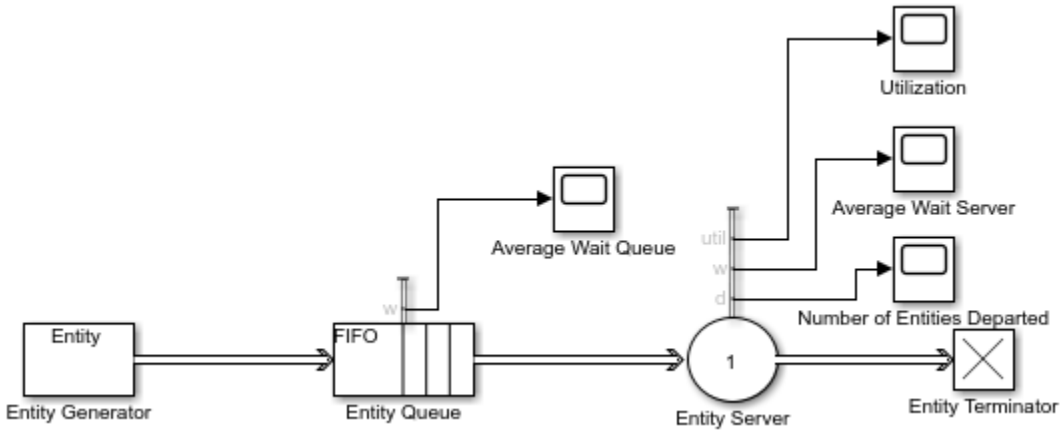
4. Connect the **util** signal output port and the two **w** signal output ports to the **in** signal input ports of the unconnected scope blocks. Save the model.

5. Simulate the model with different values of the **Period** parameter for the entity intergeneration times in the Entity Generator block. Observe the plots to see how they change if you set the intergeneration time to `0.3`, `1.1`, or `1.5`, for example.

**Note** Scope blocks do not support bus objects. SimEvents software supports Scope blocks with only single inputs.

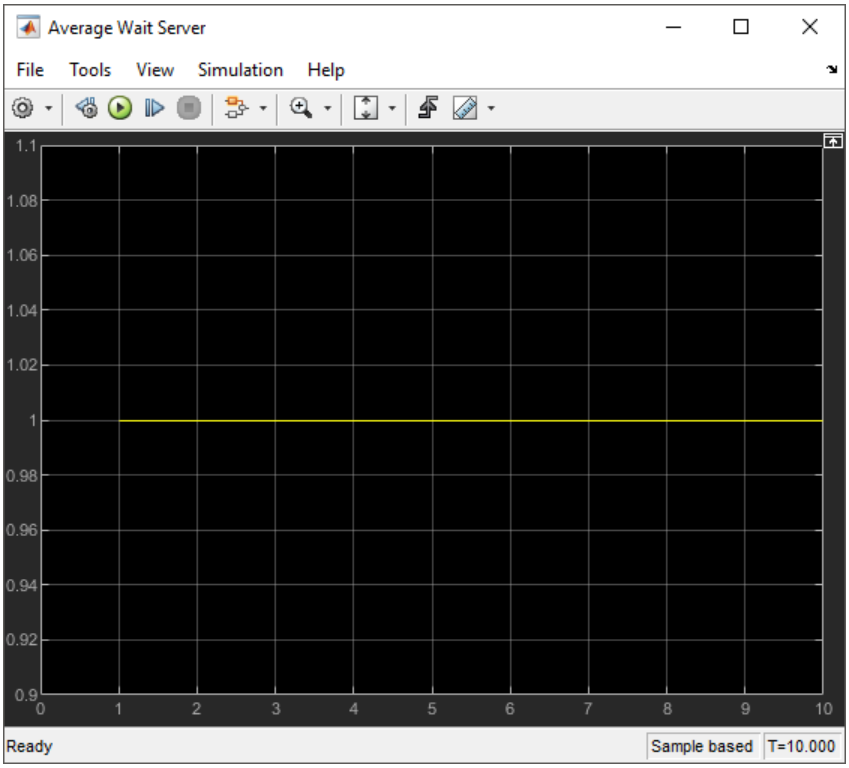
### Visualize and Explore Simulation Results

Open the example to explore simulation results.

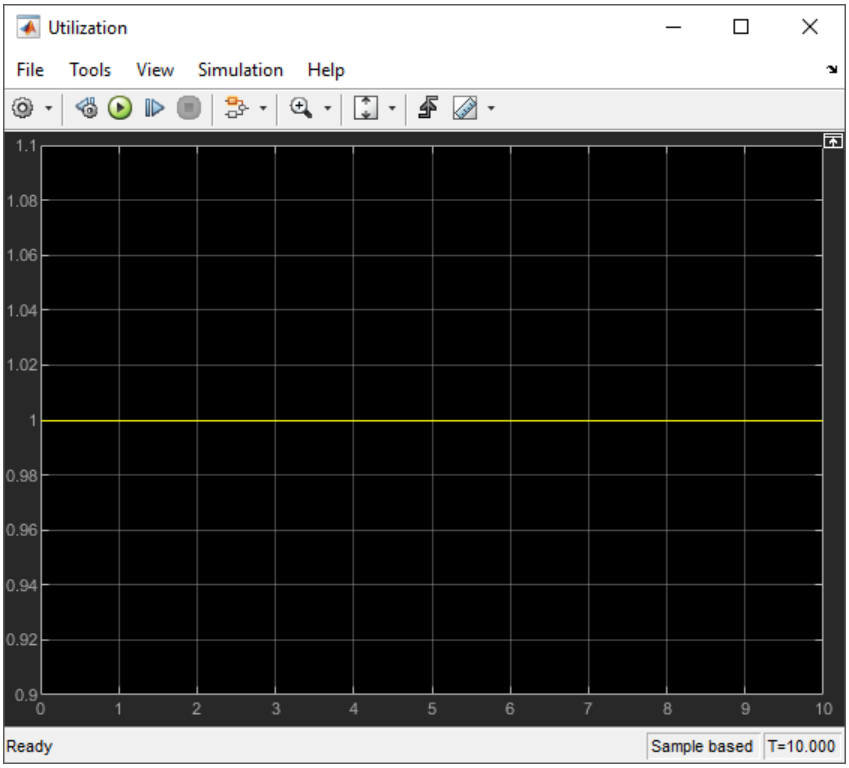



### Observations from Plots

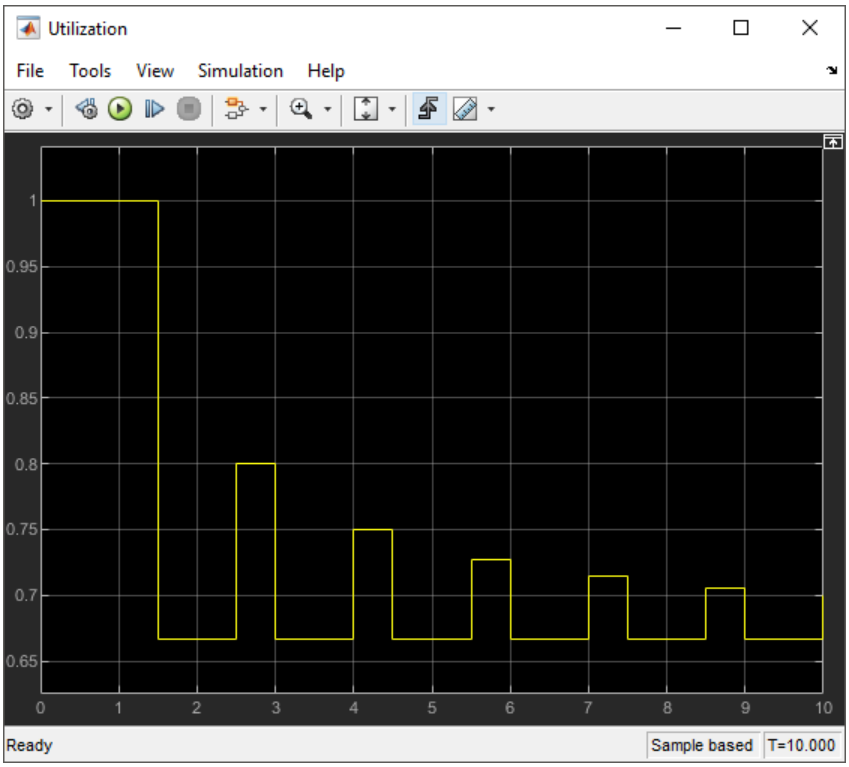
- For intergeneration time `0.3` or `1.1`, the average wait time **w** in the Server block does not change after the first departure from the block because the service time is fixed for all departed entities. The average waiting time statistic does not include partial wait times for entities that are in the server but have not yet departed.



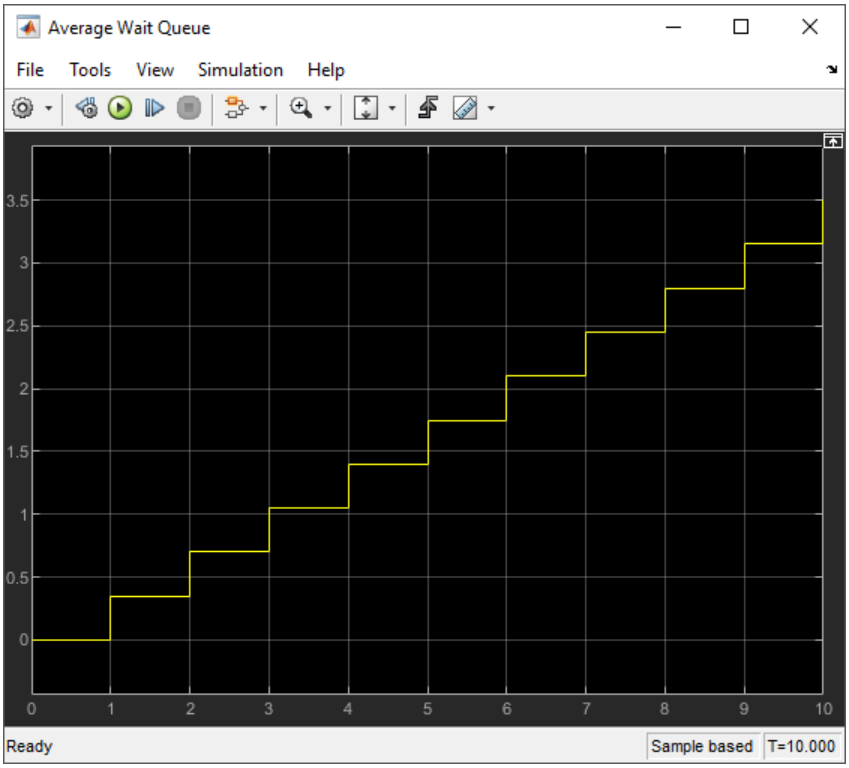

- For intergeneration time `0.3`, the utilization of the server **util** is nondecreasing because the server is constantly busy once it receives the first entity.



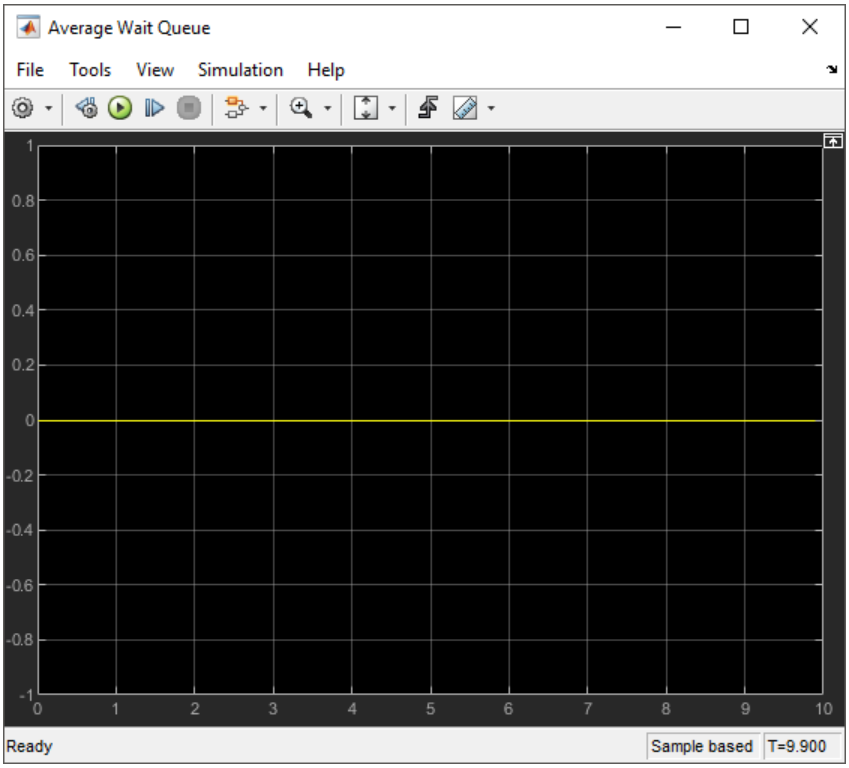
- For intergeneration time `1.5`, which is larger than the service time (1), the utilization may decrease because the server has idle periods between entities.




- For intergeneration time `0.3`, the average waiting time **w** in the queue increases throughout the simulation because the queue gets longer and longer.




- For intergeneration time `1.1`, which is larger than the service time (1), the average waiting time **w** in the queue is zero because every entity that arrives at the queue is able to depart immediately.



## See Also

Entity Server | Entity Terminator | Entity Generator | Entity Queue

## Related Examples

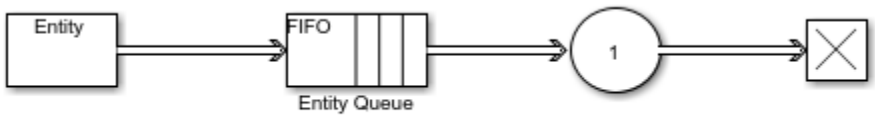
- “Manage Entities Using Event Actions” on page 2-13
- “Create a Hybrid Model with Time-Based and Event-Based Components” on page 5-2
- “Interpret SimEvents Models Using Statistical Analysis”

# Manage Entities Using Event Actions

This example shows how to control entity generation rate and write event actions to change entity attributes in a simple queuing system. In a discrete-event simulation, an event is an observation of an instantaneous incident that may change a state variable, an output, or the occurrence of other events. SimEvents allows you to create custom actions when an event occurs. These actions are called event actions. Events can have corresponding actions. You can write event actions to change entity attributes by using MATLAB code or Simulink functions.

## Start with a Simple Queuing System

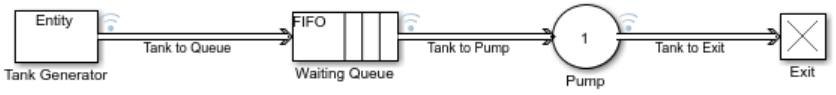
This is a simple queueing system with Entity Generator, Entity Queue, Entity Server, and Entity Terminator blocks. In this example, an entity represents a tank of a truck that arrives at a gas station. The attribute of an entity represents the current gas level in a tank. Event actions represent the changes of the gas level in a tank. Tanks are randomly generated, queued, and they are serviced with a pump which transfers a constant amount of gas for one second. Tanks depart from the station with their new total gas amount.






## Modify the Model

1. Select the whole model or the entity paths originating from the Entity Generator, Entity Queue, and Entity Server blocks and right-click to select **Log Selected Signals**. Simulation Data Inspector is used to visualize the flow of tanks and their gas level in the model. For more information, see “Inspect Simulation Data” (Simulink).
2. Rename the Entity Generator block as Tank Generator, the Entity Queue block as Waiting Queue, the Entity Server block as Pump, and the Entity Terminator block as Exit.



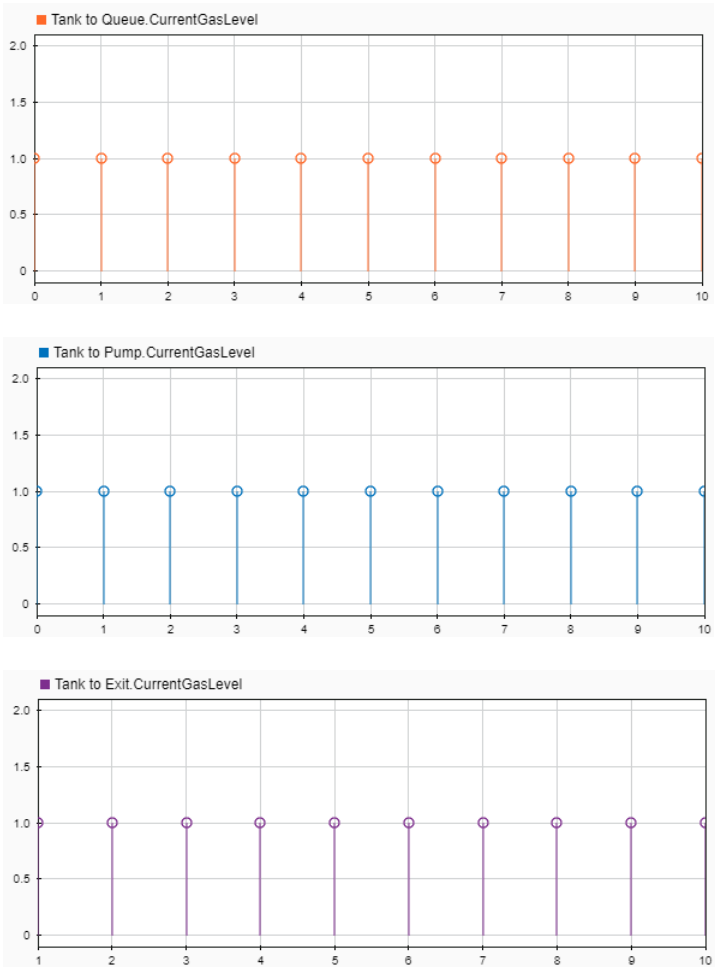

3. Rename the path originating from the Tank Generator block as Tank to Queue, the Waiting Queue block as Tank to Pump, and the Pump block as Tank to Exit.

## Configure and Simulate Model

1. Double-click the Tank Generator, and select the **Entity type** tab. Change the **Entity type name** to `Tank`, and the **Attribute Name** to `CurrentGasLevel`.

   The entity attribute `CurrentGasLevel` represents the existing amount of gas in each tank.
2. Simulate the model. Open the **Simulation Data Inspector**. Observe that the tanks approach the Waiting Queue, the Pump, and the Exit with the same rate.

   Tanks leave the station with their initial gas amount `1` which is the **Attribute Initial Value**.

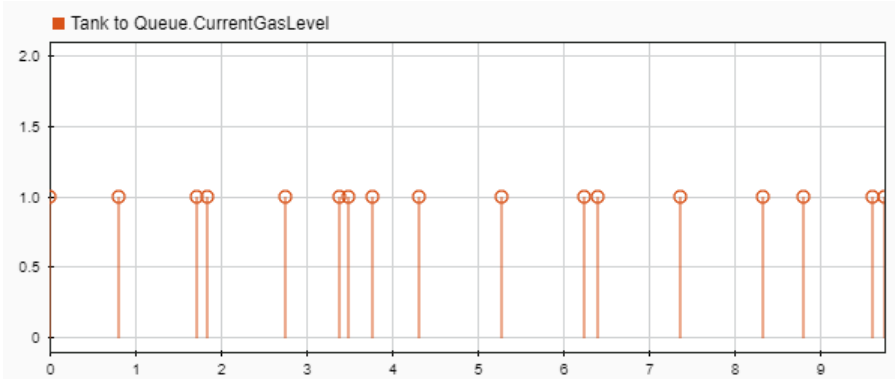



3 Open the Tank Generator block parameters dialog box. In the **Entity generation** tab, set **Time source** to `Matlab action`. Observe the default MATLAB code.

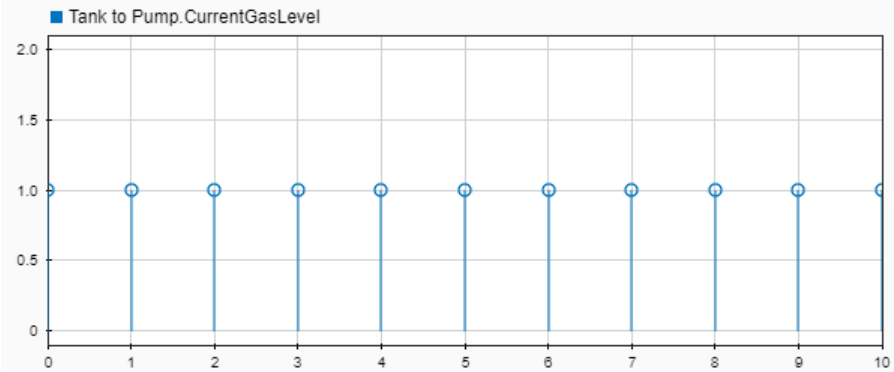
```
dt = rand(1,1);
```

The code randomizes the entity intergeneration time parameter `dt` to represent random tank arrivals.

4 Simulate the updated model. In the **Simulation Data Inspector**, observe that tanks arrive randomly with the same initial gas amount 1.



Observe that the tanks are generated randomly but they approach the pump with a regulated fixed rate because service time for the Pump is 1.

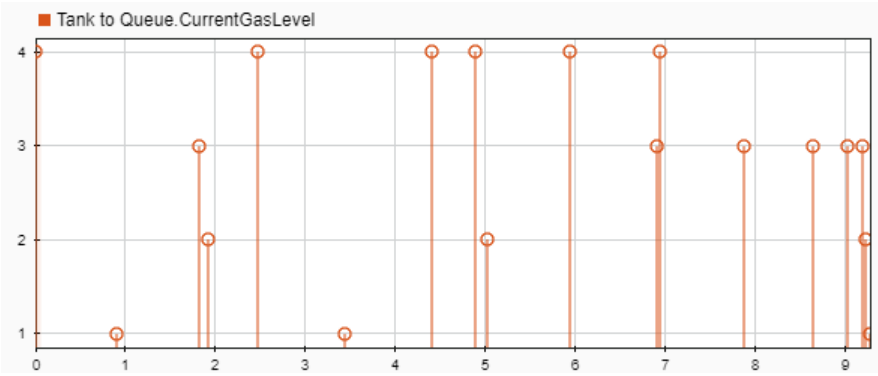



5 Open the Tank Generator block dialog box. In the **Event actions** tab, in the **Generate action** field, enter the code.

```
entity.CurrentGasLevel = randi([1,4]);
```

Tanks arrive at the station with a random gas amount that ranges from 1 to 4.

6 Simulate the updated model. In the **Simulation Data Inspector**, observe that the tanks arrive with random amounts of gas.




7 For the Pump block, set these parameters:

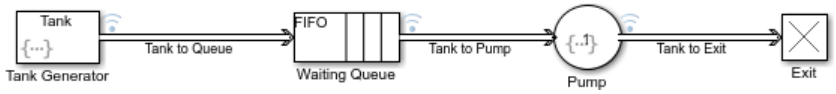
a In the **Event actions** tab, select **Service complete**.

b For the **Service complete action** field, enter the code.

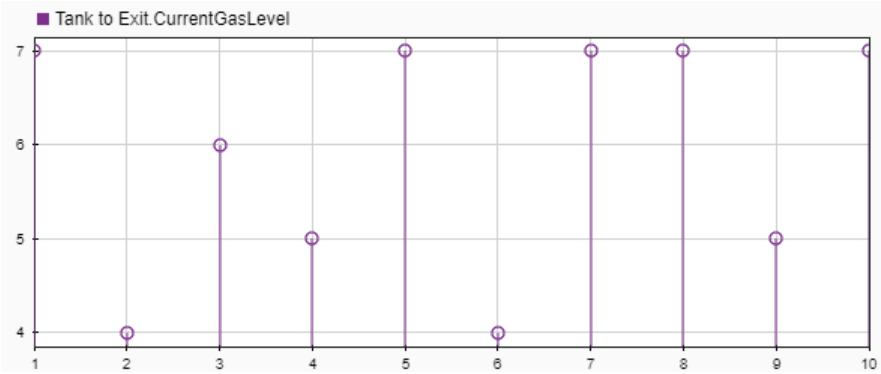
```
entity.CurrentGasLevel = entity.CurrentGasLevel + 3;
```

Each tank is filled with 3 units of gas for 1s duration, and then it departs the pump.

Observe that the Tank Generator and the Pump blocks update with the event action icon {...} indicating that the blocks define an event action.

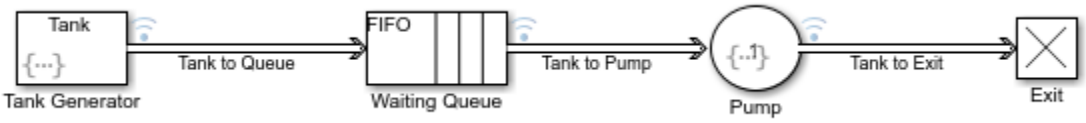



8 Simulate the updated model. In the **Simulation Data Inspector**, observe that each tank leaves the station with 3 additional units of gas.




## Modified Model to Manage Entities in a Queueing System

This is the modified model after configuring the simple queueing system.




## See Also

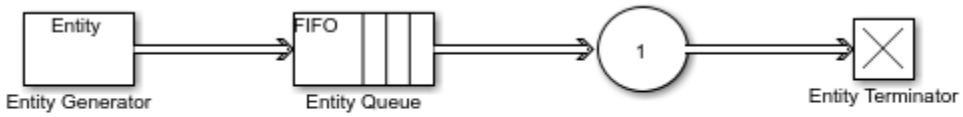
Entity Generator | Entity Queue | Entity Server | Entity Terminator

## Related Examples

- “Create a Hybrid Model with Time-Based and Event-Based Components” on page 5-2
- “Create a Discrete-Event Model” on page 2-2
- “Explore Statistics and Visualize Simulation Results” on page 2-6

# Trigger Simulink Components with Discrete Events in SimEvents

This example shows how to use Simulink Function blocks to timestamp entities, pass entity attributes to Simulink® components, and create notification events for routing. You can Use Simulink Function blocks to grab entity attributes pass them to Simulink® components for processing and then pass them back to your SimEvents® model.





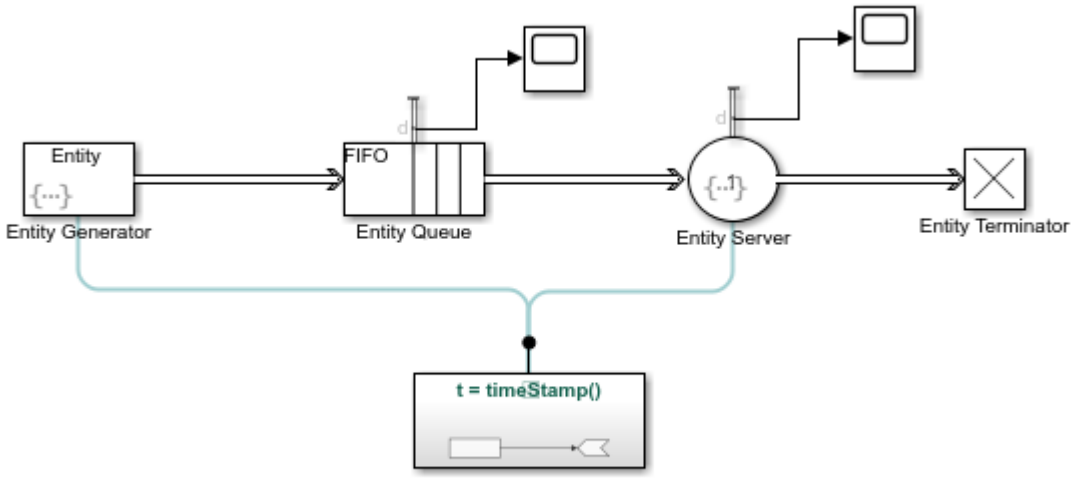

This is a simple discrete-event queueing system constructed by using Entity Generator, Entity Queue, Entity Server, and Entity Terminator blocks. To learn how to construct this model, see “Create a Discrete-Event Model” on page 2-2.

In this example, we use this simple SimEvents® model and Simulink Function blocks to show how to:

- Timestamp entities and measure the time between an entity's generation and service completion. You can use this workflow to track how much time is required to processes entities in a queueing system.
- Extract entity attributes and use the attribute values in a Simulink component. You can use this workflow to pass entity attributes to a Simulink® algorithm.
- Import data from a spreadsheet to a SimEvents® model and specify entity generation intervals. You can use this workflow to set block and entity parameters from existing data.
- Extract entity attributes and pass them to another entity with different type.

### Timestamp Entities Using a Simulink Function Block

This model shows how to track time for entities flowing through your system. A Simulink Function block is used to timestamp entity generation and service times and to compute the total duration between these two processes.

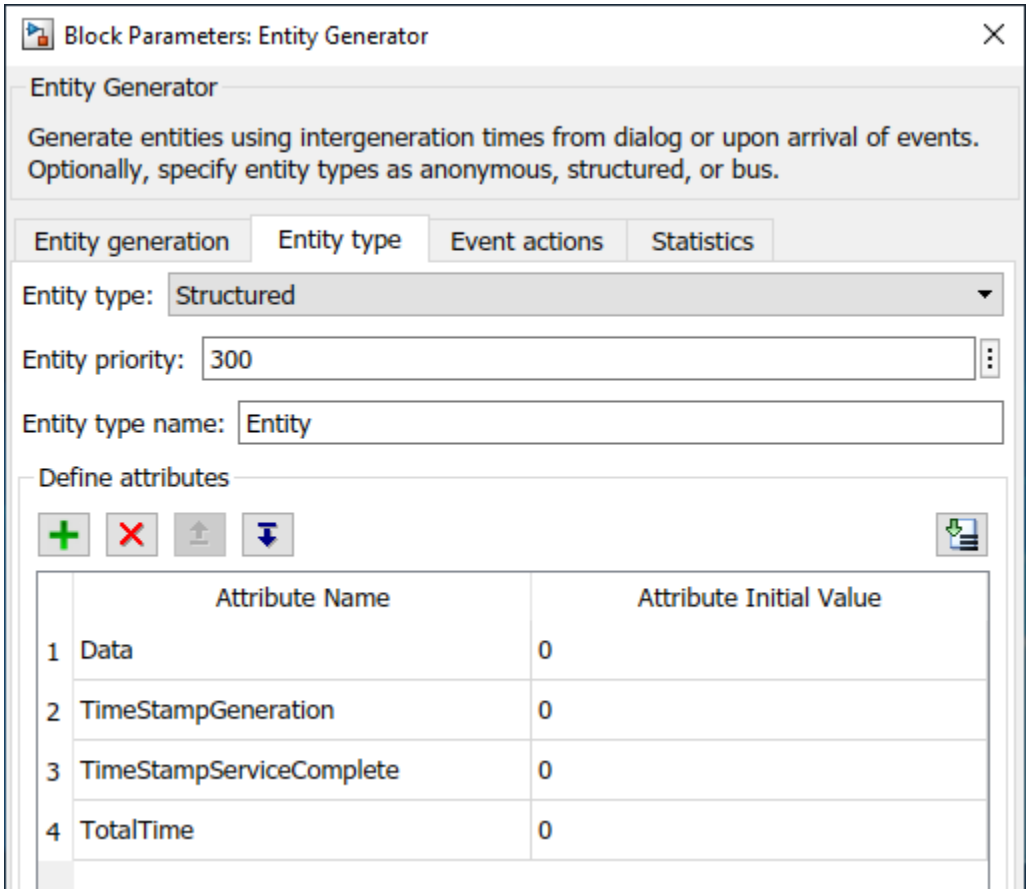





To open the model, use this code:

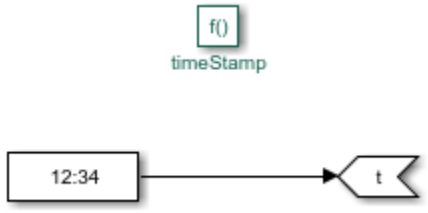
```
open_system('TimeStampEntitiesUsingSimulinkFunctionModel');
```

In this model, the entity intergeneration intervals are generated from a uniform distribution on the open interval `(0,1)`. The entities have four attributes and all attributes have an initial value of `0`:

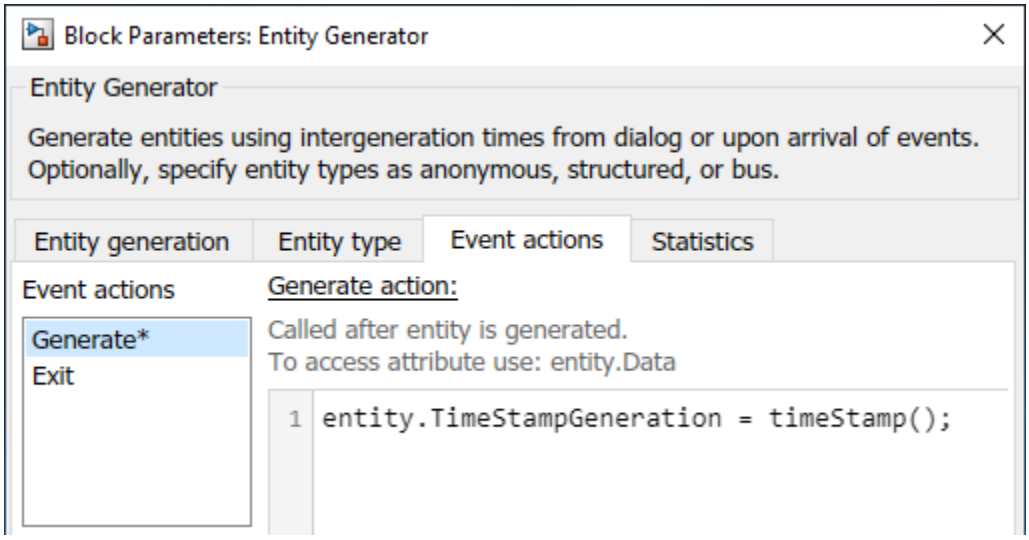



- `Data` represents the data the entities carry.
- `TimeStampGeneration` stores entity generation time.
- `TimeStampServiceComplete` stores entity service completion time.
- `TotalTime` is the time between an entity's generation and service.

In the Simulink Function block, the Digital Clock block timestamps the entity generation time.




You can timestamp when the entity is generated by calling the `timeStamp()` function in the Simulink Function from the Entity Generator block.




To call the function, in the Entity Generator block, in the **Event actions** tab, in the **Generate** field, use this code:

```
entity.TimeStampGeneration = timeStamp();
```

The code calls the `timeStamp()` function and assigns the value from the Digital Clock block to the `TimeStampGeneration` attribute when the entity is generated.

Similarly, to timestamp service completion and calculate the time difference between entity generation and service, open the Entity Server block, and in the **Event actions** tab, click the **Service complete**. Enter this code.

```
% Stamp the service completion by calling the |timeStamp()|
% function.
entity.TimeStampServiceComplete = timeStamp();
% Calculate the difference between generation and service
% completion time.
entity.TotalTime = entity.TimeStampServiceComplete - entity.TimeStampGeneration;
% Display the entity attribute for the time difference.
disp(entity.TotalTime);
```

The service completion time, which is acquired by calling `timeStamp()`, is assigned to the `entity.TimeStampServiceComplete` attribute. Then the duration between the entity's generation and service completion is calculated and assigned to `entity.TotalTime`. The code displays `entity.TotalTime` values for each entity in the Diagnostic Viewer.

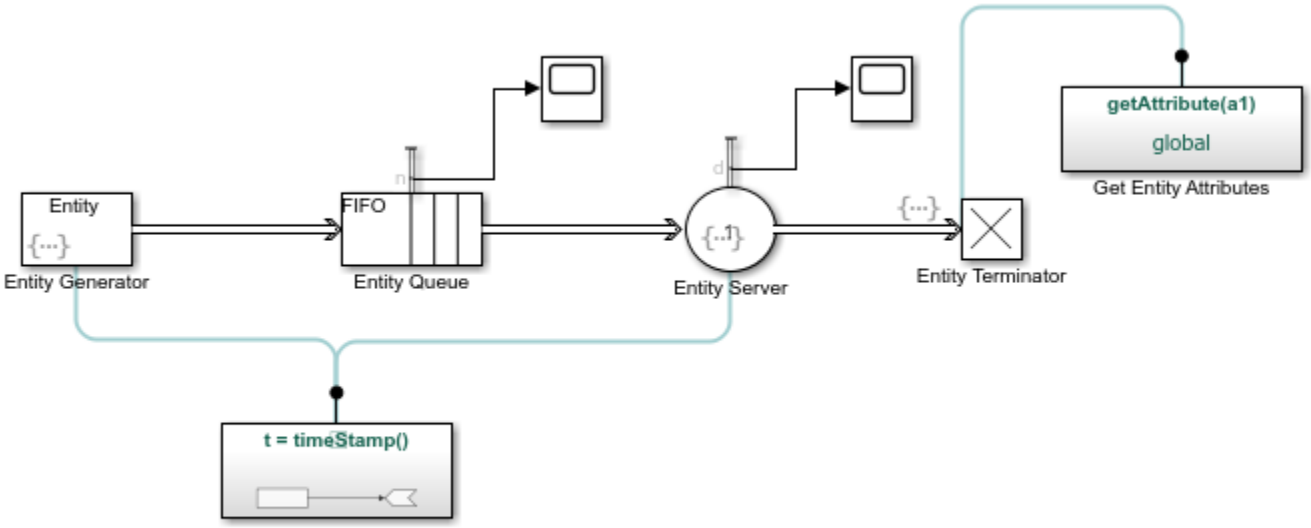
Simulate the model. In the Diagnostic Viewer, you can observe the `entity.TotalTime` values for `10` entities. The duration between entity generation and service increases because entities wait in the Entity Queue block for their turn of service.

```
 1
1.1853
1.2795
2.1525
2.2391
2.6068
3.5092
4.2307
4.6838
4.7263
```

Increase the simulation time to `1000` and observe that `entity.TotalTime` converges to approximately 26.

### Pass `TotalTime` Attribute to a Simulink Component

Suppose that you want to pass `entity.TotalTime` values to a Simulink® component. This model shows how to pass the attribute value to a Simulink Function block when an entity arrives at the Entity Terminator block.



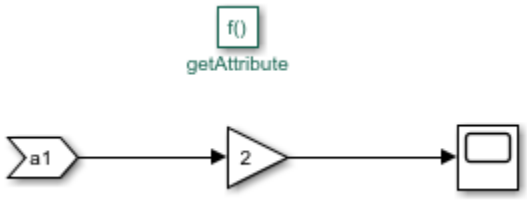



To open this model, use this code:

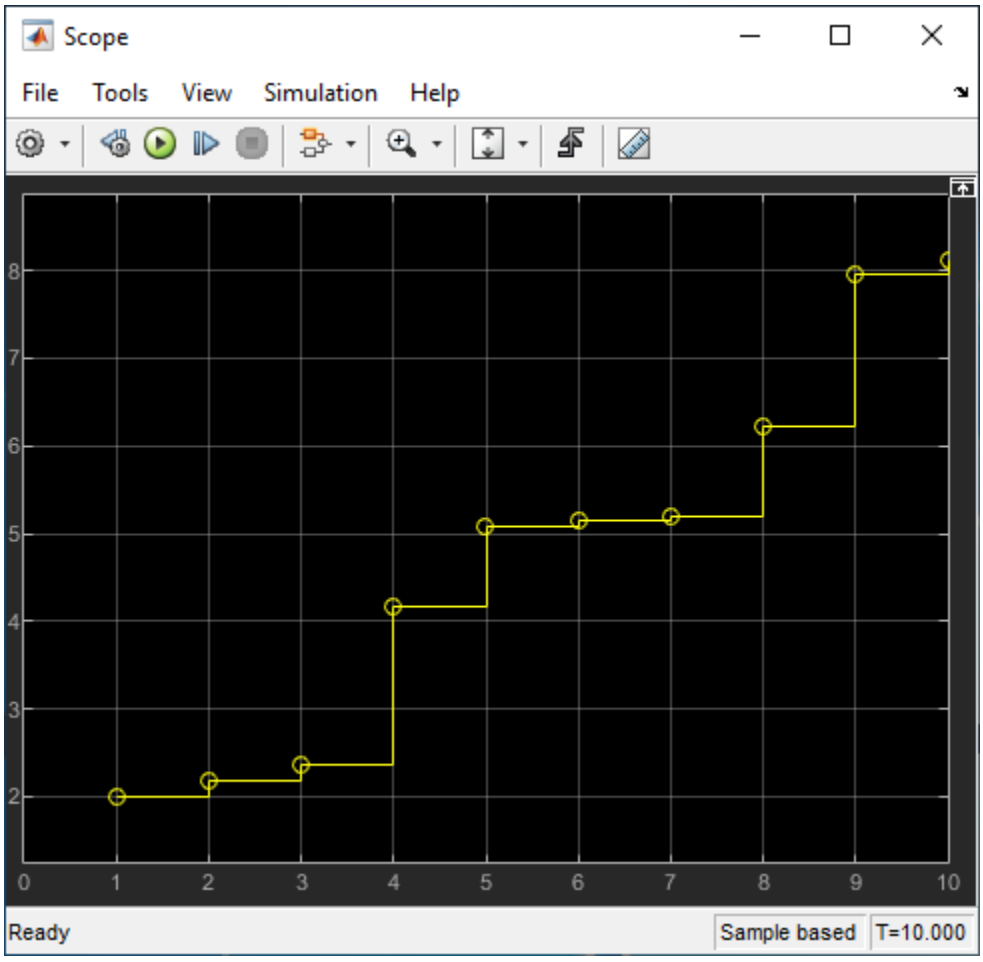
```
open_system('GetEntityAttributesSimulinkFunctionModel');
```

To achieve this behavior, open the Entity Terminator block, in the **Event actions** tab, click **Entry** and call the `getAttribute(entity.TotalTime)` function.

The Get Entity Attributes block takes `entity.Totaltime` as the input argument and uses a Gain block to amplify its values by multiplying them by 2.



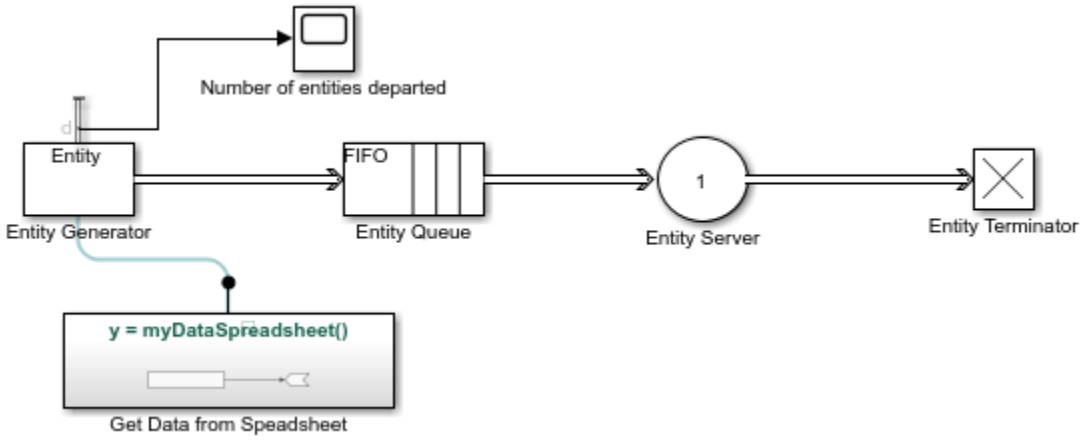

Simulate the model. Observe the Scope block that shows the amplified attribute values.




### Import Data from a Spreadsheet to Specify Entity Intergeneration Times

Suppose that you want to incorporate data from a spreadsheet to your simulation. Using a spreadsheet, you can specify various parameters in your model, such as entity intergeneration time, entity attributes, or service time.

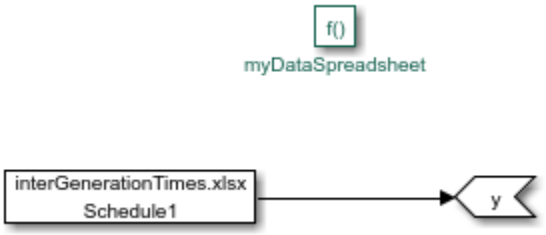
This example model shows how to import data from a spreadsheet to a SimEvents® model and use data to specify entity intergeneration times.




To open this model, use this code:

```
open_system('ImportDataSimEventsModel');
```

In the model, a From Spreadsheet block is inside the Simulink Function block and acquires values from the `interGenerationTimes.xlsx` spreadsheet. The spreadsheet has five values — 1, 2, 3, 4, and 5 — to be used as entity intergeneration times.

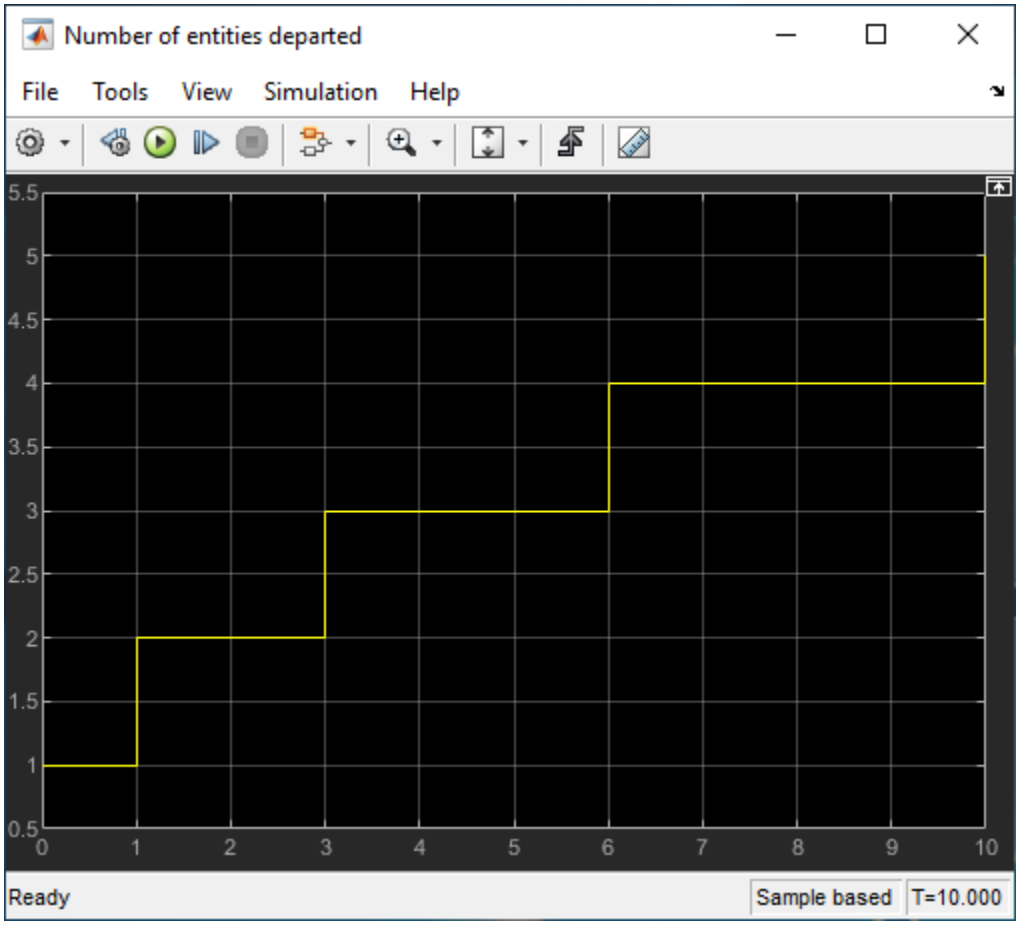



To call the `myDataSpreadsheet()` function, in the Entity Generator block, in the **Intergeneration time action** field, use this code:

```
dt = myDataSpreadsheet();
```

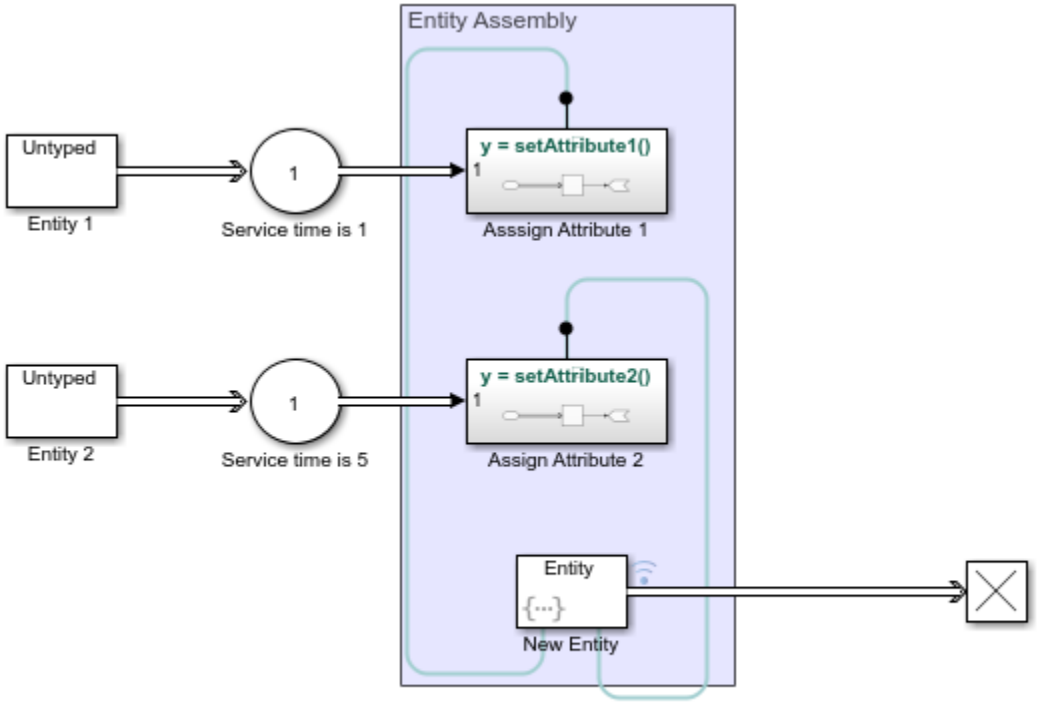
`dt` is the variable that specifies the intergeneration times for entities. The code assigns `dt` values by calling the Simulink function `myDataSpreadsheet()`, which acquires values from the spreadsheet.

Simulate the model. Observe the Scope block that displays when entities are generated and depart the block. The intervals between entity generation are the same as the data from the spreadsheet.




### Pass Entity Attributes Between Different Entity Types

In SimEvents, you can create a model that has different entity types and pass the attributes between entities using a Simulink Function block.



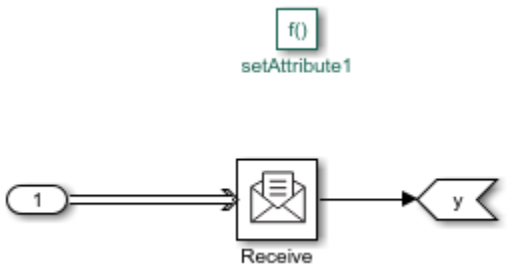



To open this model, use this code:

```
open_system('AssignEntityAttributeSimulinkFunctionModel');
```

In the model, two Entity Generator blocks generate entities. Entity 1 generates entities with a constant value of 2 and are serviced for 1 simulation time. After the service is complete, the entities arrive at the Simulink Function block labeled Assign Attribute 1.

In Assign Attribute 1, entities are received by a Receive block with an internal queue of size 16. The Receive block converts the entity data to signal values.

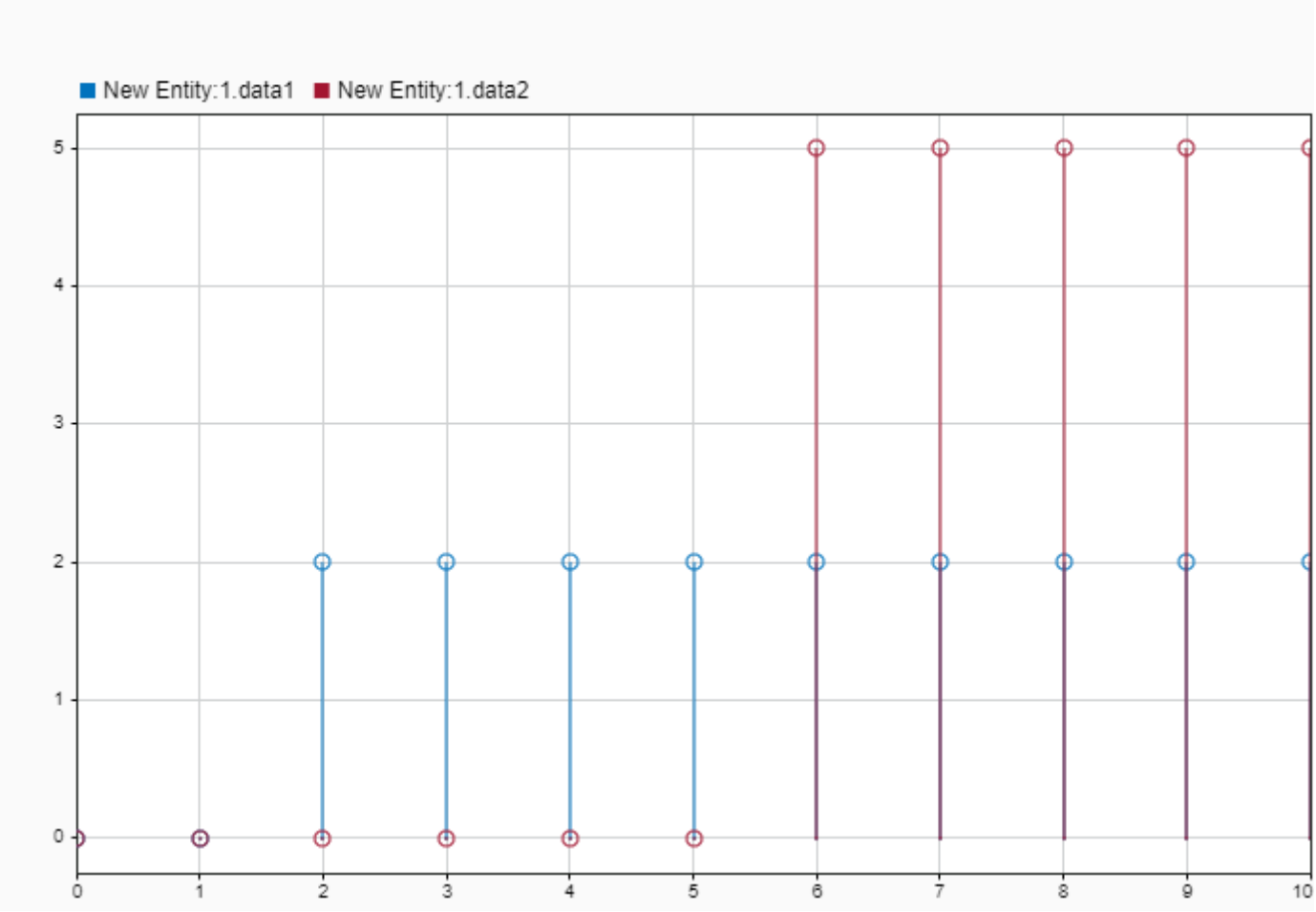



Similarly, Entity 2 generates entities that carry data of value 2 and are serviced for 5 simulation time. After the service is complete, entities arrive at Assign Attribute 2.

The entity data is passed to another Entity Generator block labeled New Entity. The New Entity block generates entities carrying two attributes, `data1` and `data2`, whose values are acquired by calling `setAttribute1()` and `setAttribute2()`, respectively.

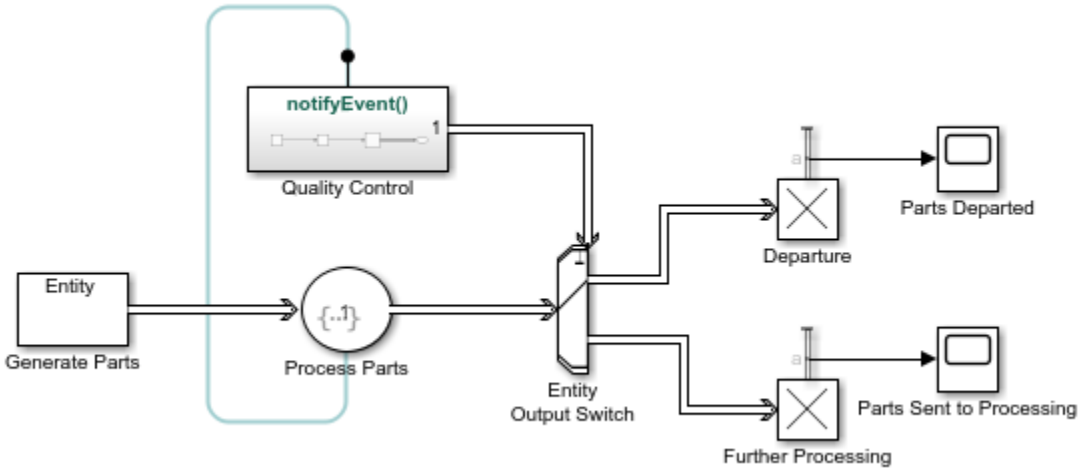
```
entity.data1 = setAttribute1();
entity.data2 = setAttribute2();
```

Simulate the model and open the Data Inspector. Observe that the values of `data1` and `data2` values of the new entity are `0` until simulation time `2`. This is because entities are serviced and there is no attribute pass between the entities. At time `2`, `data1` is `2`, which is the value that is passed by `setAttribute1()`. At time `6`, `data2` starts to acquire values from `setAttribute2()`. This delay is due to the difference between the service times of the entities.




### Create a Notification Event for Routing

This model shows how to use a Simulink Function block to create an event to notify a routing block when an entity's processing is complete.



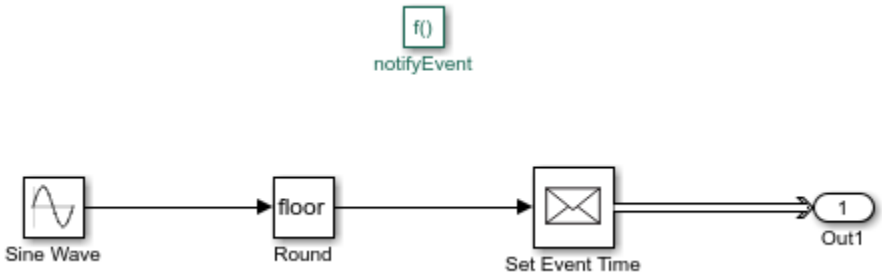

To open this model, use this code:

```
open_system('NotifyEventSimulinkFunctionCallModel');
```

In the model, an Entity Generator block generates entities that represent parts in a facility. The entities are then processed by an Entity Server block. If a part passes quality control, the Entity Output Switch block routes the parts to Departure. Otherwise, the parts are sent to Further Processing.

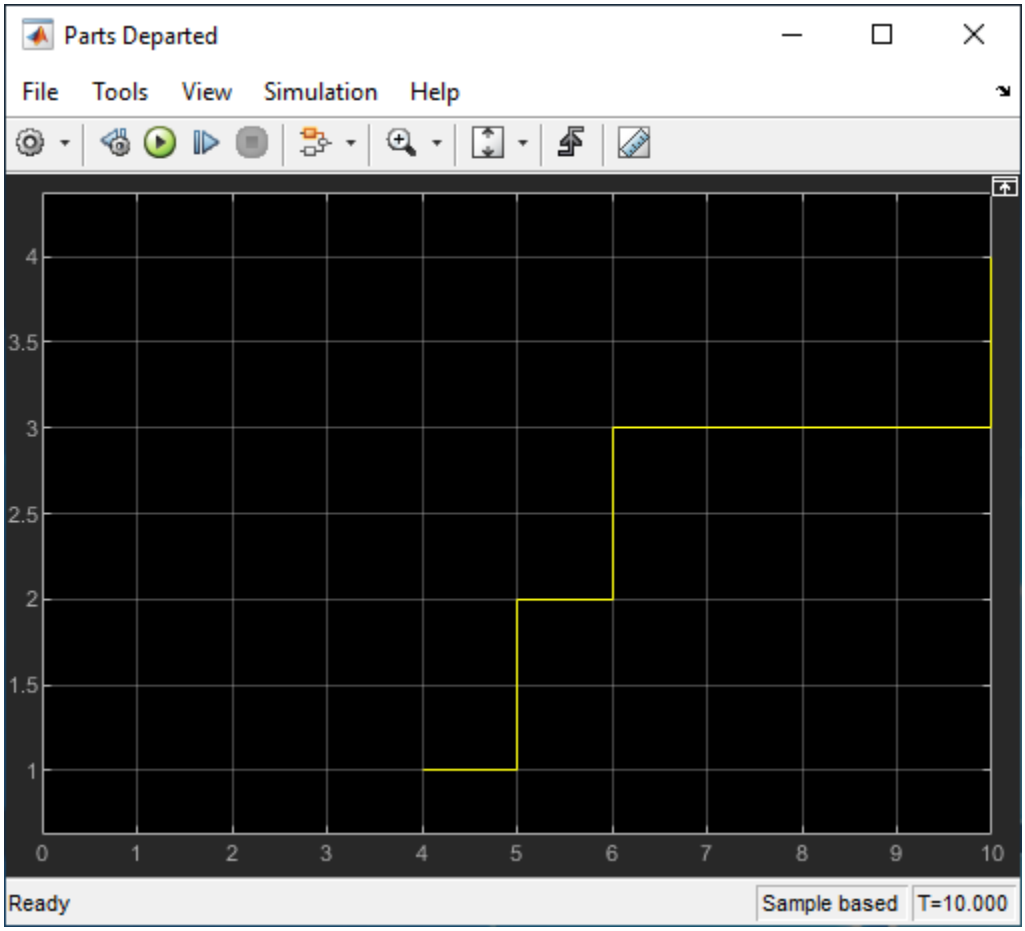
To create a notification event after an entity is processed, in the Entity Server block, in the **Event actions** tab, in the **Service complete action** field, call the `notifyEvent()` function.

In the Quality Control block, a Sine Wave block is used to generate a signal. A Round block is used to round the signal values to the nearest integer less than or equal to that value. The output signal from the Round block takes the value 1 or 2. The signal is converted to a message by the Set Event Time block.

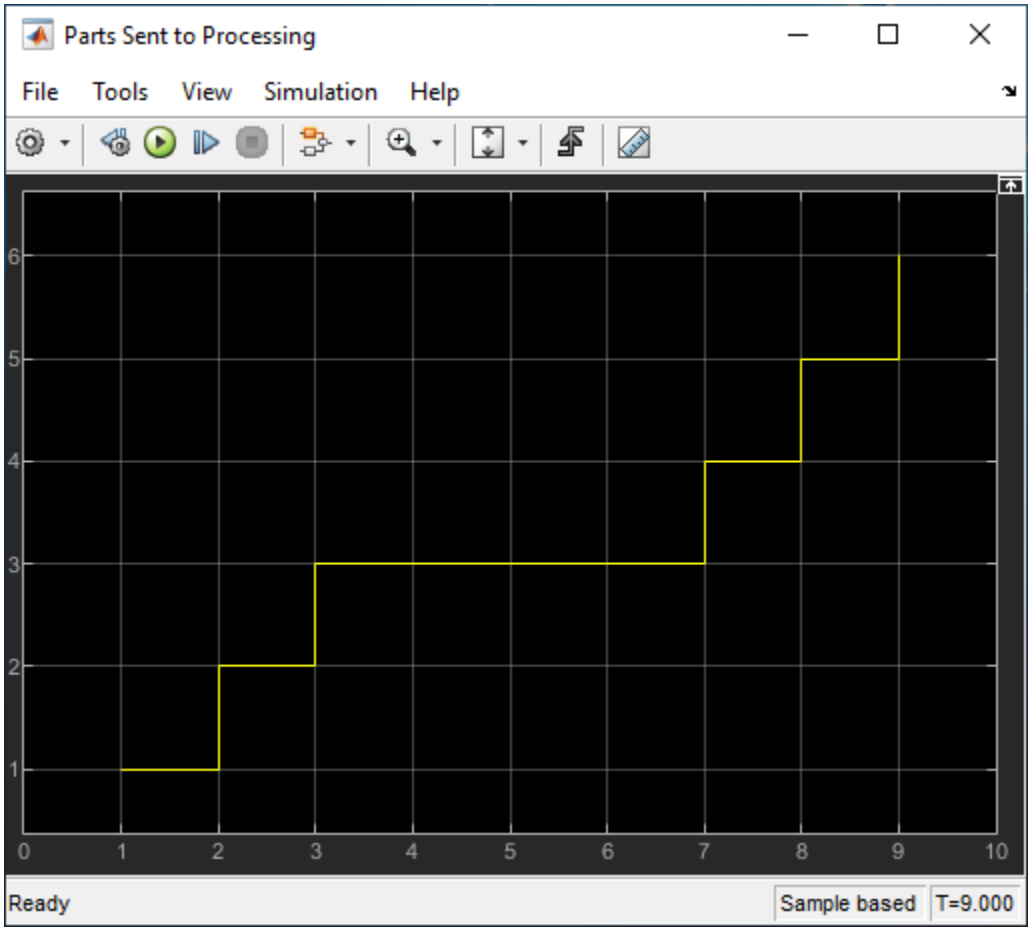



The message data value from the Quality Control block specifies which output port is selected when entities depart the Entity Output Switch block. If the message carries value 1, output port 1 is selected for entity departure. If the message carries value 2, output 2 is selected for entity departure.

Simulate the model and observe the Scope block labeled Parts Departed. Four parts depart the facility.




Observe the Scope block labeled Parts Sent to Processing, which shows that six parts are sent to further processing.




## See Also

Gain | Bias | Simulink Function | Entity Generator | Entity Queue | Entity Server | Entity Terminator

## Related Examples

- “Manage Entities Using Event Actions” on page 2-13
- “Create a Hybrid Model with Time-Based and Event-Based Components” on page 5-2
- “Create a Discrete-Event Model” on page 2-2
- “Explore Statistics and Visualize Simulation Results” on page 2-6

# Key Concepts in SimEvents Software

# Entities in a SimEvents Model

| In this section... |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

Entities are discrete items of interest in a discrete-event simulation. By definition, these items are called entities in SimEvents software. Entities can pass through a network of queues, servers, gates, and switches during a simulation. Entities can carry data, known in SimEvents software as attributes.

SimEvents models typically contain at least one source block that generates entities. Other SimEvents blocks in the model process the entities that the source block generates.

**Note** Entities are not the same as events. Events are instantaneous discrete incidents that change a state variable, an output, and/or the occurrence of other events. See “Events and Event Actions” for details.

## Meaning of Entities in Different Applications

You determine what an entity signifies, based on what you are modeling. The table lists example entity representations in various applications.

| Context of Sample Application | Entities |
| --- | --- |
| Airport with a queue for runway access | Airplanes waiting for access to runway |
| Communication network | Packets, frames, or messages to transmit |
| Bank of elevators | People traveling in elevators |
| Conveyor belt for assembling parts | Parts to assemble |
| Computer operating system | Computational tasks or jobs |

## Vary the Interpretation of Entities

A single model can use entities to represent different kinds of items. For example, if you are modeling a factory that processes two different kinds of parts, you can:

- Use two Entity Generator blocks to create the two kinds of parts.
- Use one Entity Generator block and subsequently assign an attribute to indicate what kind of part each entity represents and another attribute to represent a property.

**Note** SimEvents entities are fundamentally the same as Simulink and Stateflow messages.

## Visualize Entities

Entities do not appear explicitly in the model window. A graphical block can represent a component that processes entities, but entities themselves do not have a graphical representation. However, you can gather information about entities using Simulink scopes. You cannot branch an entity connection line. If your application requires an entity to arrive at multiple blocks, use Entity Replicator block to create copies of entities.

## Storing Entities

These blocks are capable of holding an entity:

- Entity Generator
- Entity Queue
- Multicast Receive Queue
- Entity Server
- Entity Terminator
- Discrete Event Chart
- MATLAB Discrete Event System
- Entity Replicator
- Resource Acquirer
- Resource Releaser

These blocks permit an entity arrival but must output or destroy the entity at the same value of the simulation clock:

- Entity Input Switch
- Entity Output Switch
- Entity Multicast
- Entity Gate
- Composite Entity Creator
- Composite Entity Splitter
- Resource Pool

## Entity Types

An entity type is the identification tag associated with any block that creates entities in your model. For the Entity Generator block, you assign a name to the entity type on the Entity type tab of the generation block. From this block, each new entity receives this tag. For example, the name of the entity type associated with an Entity Generator in your model might be `Customer`. Each entity that originates in that block receives this entity type. A Composite Entity Creator block also generates new entities by combining two or more existing entities to form a new composite entity. You can assign a new entity type name to the entity type (named `Combined` by default).

---

**Note** The Entity Replicator block also generates new entities by outputting copies of an incoming entity. However, because the incoming entity already possesses an entity type, the block does not create new entity types for the copies.

---

As an entity progresses through your model, its type does not change. Even if the entity acquires attribute, timeout, or timer data that give it a more complex structure, the entity type remains the same. Although a Composite Entity Creator block forms new composite entities with a new entity type, the underlying entity types remain the same.

By default, each new entity type that SimEvents creates in your model uses the name `Entity`.

The Entity Generator block can generate these entity types:

- Anonymous — Unstructured entity with no name. You can specify only entity priority and initial data value for anonymous entity types.
- Structured — Structured entity type that you define in this block dialog box. You can name entities, specify priorities, and specify attributes for the entity in the **Define attributes** section of the Entity Generator block. Attributes are data carried by entities. Creating a structured entity in this tab is a convenient way to create an entity without having to create an associated bus object in Simulink.
- Bus object — Entity type that you define using Simulink bus objects. You can name entities, specify priorities, and specify attributes for the entity. To specify this entity type, you must have an existing bus object, created in Simulink, and use that bus object name as the name of the entity type. This bus object:
  - Must be a valid bus object with one or more bus elements at a single level.
  - Cannot contain variable-size elements. This limitation is also true for entities registered as bus objects through the Composite Entity Creator block.

## Data and Role of Entity Attributes

You can optionally attach data to entities. Such data is stored in one or more attributes of an entity. You define names and numeric values for attributes. For example, if your entities represent a message that you are transmitting across a communication network, you might assign data called `length` that indicates the length of each particular message. You can read or change the values of attributes during the simulation.

Entities and attributes can be of any data type that Simulink supports, including enumerated types. For more information, see “Data Types Supported by Simulink” (Simulink).

Data types supported by MATLAB but not supported by Simulink may not be passed between the Simulink model and event actions. You can use these data types in event actions as local variables.

You can optionally specify the structure of an entity using a Simulink bus object. This capability is useful when defining complex entity structures that need to be defined once, but used in multiple locations in a model. In addition, the MATLAB Discrete-Event System and Discrete Event Chart blocks require that you specify entities as bus objects. For more information on bus objects, see “Specify Bus Properties with Simulink.Bus Objects” (Simulink).

## Create Entities in a SimEvents Model

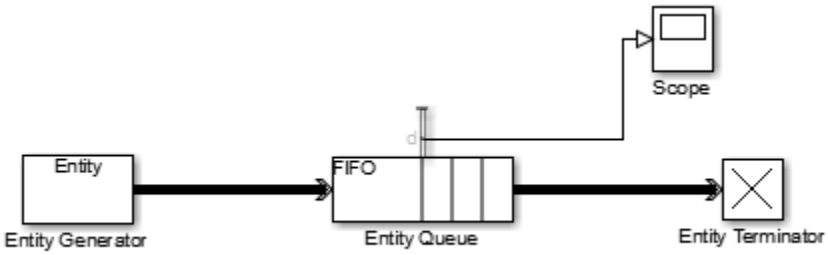
An Entity Generator block can be used to generate entities. By default, the block creates time-based entities. You can change the **Time Source** parameter to select the time source for the entity generation. You can create time-based entities using:

- The **Period** parameter value. For more information, see “Create Time-Based Entities” on page 3-5.
- A signal port. You can then connect a Simulink source block, such as a Repeating Sequence block, to the signal port. The time value cannot be a negative number. For more information, see “Specify Intergeneration Times for Entities”.
- MATLAB code. For more information, see “Create Randomized Entities” on page 3-6.

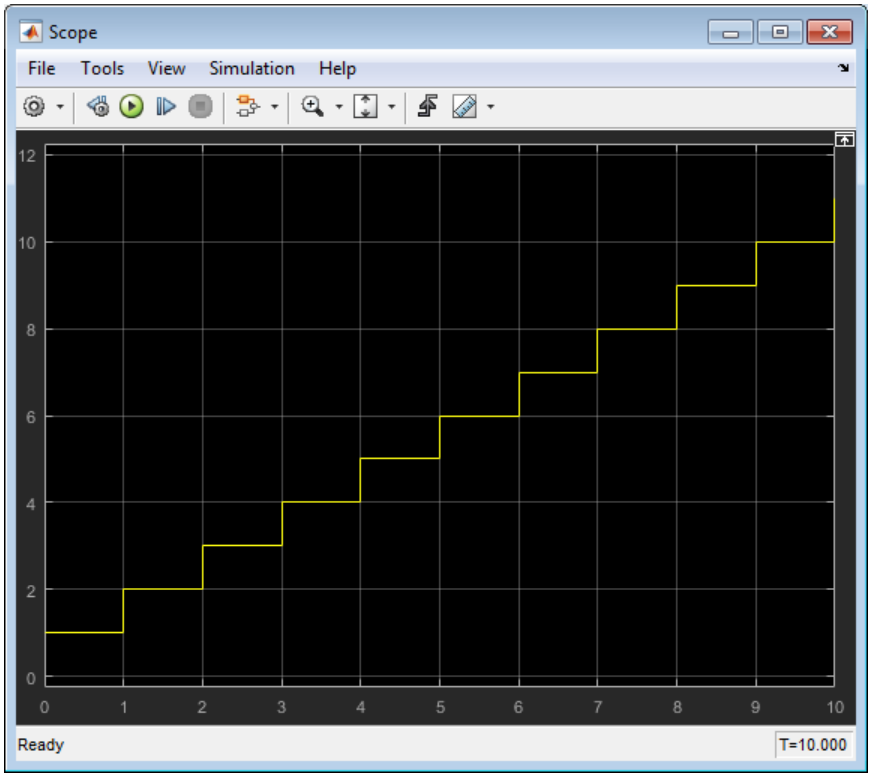
### Create Time-Based Entities

Use the Entity Generation block to create time-based entities. The Entity Generation lets you specify a period at which it creates entities.

1. Open the SimEvents block library. You can use the Simulink browser or type `simevents` at the MATLAB Command Window.
2. Create a new model.
3. From the SimEvents library, drag the Entity Generator block to the new model.
4. From the SimEvents library, drag the Entity Queue block to the new model.
   - Connect the Entity Generator block to the input of the Entity Queue.
   - In the Entity Queue block, select **Number of entities departed, d**.
5. From the Simulink Sinks library, drag a Scope block to the new model. Connect the Scope block to the d port of the Entity Queue block.
6. From the SimEvents library, drag an Entity Terminator block to the new model. Connect the output of the Entity Queue block to the input of the Entity Terminator block.




Upon simulation, the scope displays the entities that depart the queue.



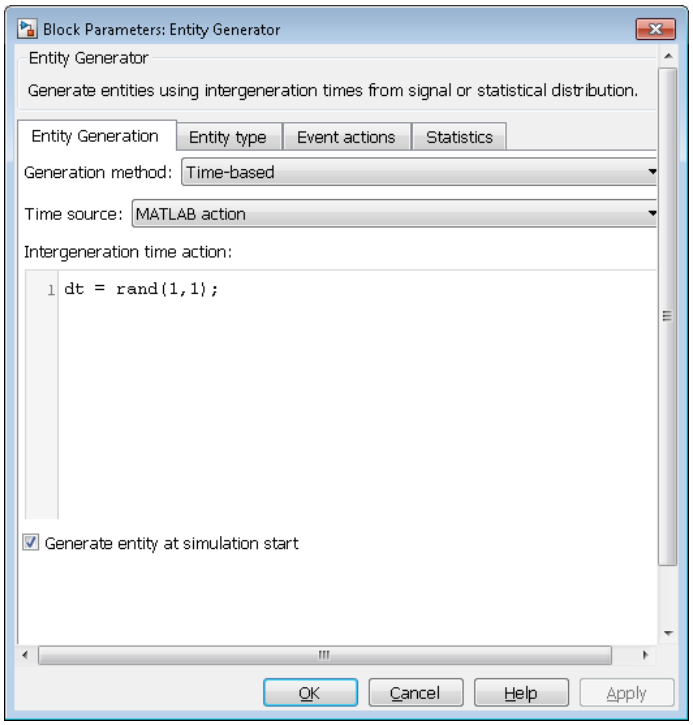

**Note** You cannot connect a scope to a SimEvents line, as denoted by the thick double arrow line.

### Create Randomized Entities

Use the Entity Generation block to create time-based entities. The Entity Generation lets you specify a randomization operation (such as the MATLAB `rand` function) to create entities at random times.

1. Open the SimEvents block library. You can use the Simulink browser or type `simevents` at the MATLAB Command Window.
2. Create a model.
3. From the SimEvents library, drag the Entity Generator block to the new model.
   a. Double-click the block and set the **Time source** parameter to `MATLAB action`.
   b. In the **Intergeneration time action** parameter, enter a call to a randomizer function, such as `rand`. For example:

```
dt = rand(1,1);
```



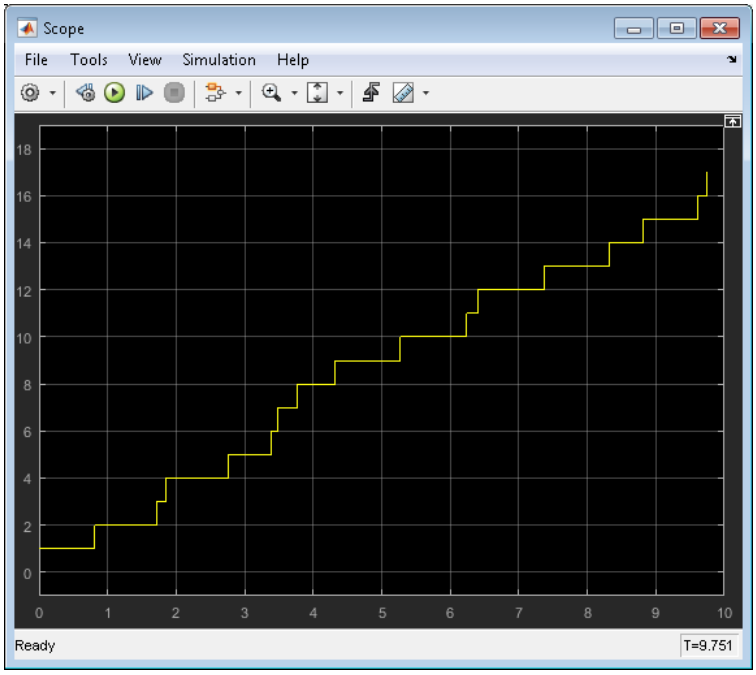

4 From the SimEvents library, drag the Entity Queue block to the new model.

   - Connect the Entity Generator block to the input of the Entity Queue
   - In the Entity Queue block, select **Number of entities departed, d**.

5 From the Simulink Sinks library, drag a Scope block to the new model. Connect the Scope block to the d port of the Entity Queue block.

6 From the SimEvents library, drag an Entity Terminator block to the new model. Connect the output of the Entity Queue block to the input of the Entity Terminator block.

Upon simulation, the scope displays the entities that depart the queue.




## See Also

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

## Related Examples

- “Generate Entities When Events Occur”
- “Specify Intergeneration Times for Entities”
- “Working with Entity Attributes and Entity Priorities”
- “Inspect Structures of Entities”

# Role of Entity Ports and Paths

| In this section... |
| --- |
|  |
|  |
|  |
|  |

## Entity Ports and Paths

An entity output port provides a way for an entity to depart from a block. Conversely, an entity input port provides a way for an entity to arrive at a block.

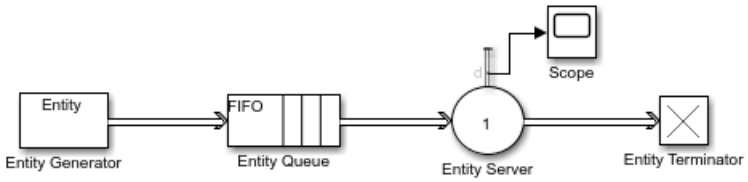
A connection line indicates a path along which an entity can potentially advance. However, the connection line does not imply that any entities actually advance along that path during a simulation. For a given entity path and a given time instant during the simulation, any of the following could be true:

- No entity is trying to advance along that path.
- An entity has tried and failed to advance along that path. For some blocks, it is normal for an entity input port to be unavailable under certain conditions. As a result, the entity fails in its attempt to advance along that path, even though the path is intact (that is, even though the ports are connected). An entity that tries and fails to advance is called a pending entity.
- One or more entities successfully advance along that path. This occurs only at a discrete set of times during a simulation.

**Note** The simulation could also have one or more times at which one or more entities successfully advance along a given entity path. Simultaneously, one or more different entities try and fail to advance along that same entity path. For example, an entity departs from a queue and, simultaneously, the next entity in the queue tries and fails to depart.

## Definition of Entity Paths

An entity path is a connection from an entity output port to an entity input port, depicted as a line connecting the entity ports of two SimEvents blocks. An entity path represents the equivalence between an entity's departure from the first block and arrival at the second block. For example, in the model shown below, any entity that departs from the Entity Queue block's output port equivalently arrives at the Entity Server block's input port.

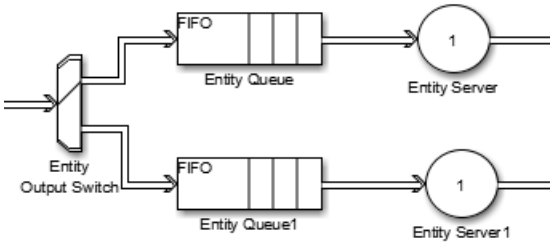


The existence of the entity path does not guarantee that any entity actually uses the path. For example, the simulation could be so short that no entities are ever generated. Even when an entity path is used, it is used only at a discrete set of times during the simulation.

## Implications of Entity Paths

In some models, you can use the entity connection lines to infer the full sequence of blocks at which a given entity arrives, throughout the simulation.

In many discrete-event models, however, the set of entity connection lines does not completely determine the sequence of blocks at which each entity arrives. This example shows two queues in a parallel arrangement, preceded by a block that has one entity input port and two entity output ports.




By looking at the entity connection lines alone, you cannot tell which queue block's **IN** port an entity arrives at. Instead, you need to know more about how the one-to-two block (Output Switch) behaves and understand the outcome of certain run-time decisions.

## Designing Paths Using Input, Output, and Entity Combiner Blocks

You design entity paths by choosing or combining entity paths using the Entity Input Switch, Entity Output Switch, and Entity Combiner blocks of the SimEvents library. These blocks have extra entity ports that let you vary the model's topology (that is, the set of blocks and connection lines).

Typical reasons for manipulating entity paths are:

- To describe an inherently parallel behavior in the situation you are modeling — for example, a computer cluster with two computers that share the computing load. You can use the Entity Output Switch block to send computing jobs to one of the two computers. You might also use the Entity Input Switch block if computing jobs share a common destination following the pair of computers.
- To design nonlinear topologies, such as feedback loops — repeating an operation if quality criteria such as quality of service (QoS) are not met. You can use the Entity Input Switch block with the **Active port selection** parameter set to `All` to combine the paths of new entities and entities that require a repeated operation.
- To incorporate logical decision-making into your simulation — for example, to determine scheduling protocols. You might use the Entity Input Switch block to determine which of several queues receive attention from a server.
- To incorporate logic for activation or deactivation of an entity path, use the Entity Gate block. For example, you can activate an entity path for one entity when a condition is fulfilled in your model.
- To model routing of copies of an entity to multiple remote locations in the model, consider using the Entity Multicast and Multicast Receive Queue blocks.

Other libraries in the SimEvents library set contain a number of blocks whose secondary features, such as preemption from a server or timeout from a queue or server, give you opportunities to design paths.

## See Also

Entity Input Switch | Entity Output Switch

## Related Examples

- “Control Output Switch with Event Actions and Simulink Function”

## More About

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

# Storage with Queues and Servers

| In this section... |
| --- |
|  |
|  |
|  |
|  |

Queue and server blocks are storage blocks that hold entities.

- Queues order entities and sort them according to queue policies.
- Servers delay entities until certain conditions are met.

Storage blocks can have event actions based on events influencing entities in the corresponding storage blocks. Each block has a set of events that invoke a specific event action. For more information on event actions, see “Events and Event Actions”.

## Behavior and Features of Queues

In a discrete-event simulation, a queue stores entities for a length of time that cannot be determined in advance. The queue attempts to output entities when it can, but its success depends on whether the next block accepts new entities. An everyday example of a queue is when you stand in a line with other people to wait for some type of service to address your needs and you cannot determine in advance how long you must wait.

The distinguishing features of queues include:

- Capacity — The number of entities the queue can store simultaneously
- Queue Policy — Which entity departs first, if the queue stores multiple entities

The Entity Queue block uses these queue policies:

- FIFO — The block processes the entity as first in first out.
- LIFO — The block processes the entity as last in first out.
- Priority — The block reads the priority from the **Priority Source** parameter. This parameter is a particular attribute value that the block stores based on the value of the number.

### Physical Queues and Logical Queues

Sometimes, a queue in a model is similar to an aspect of a real-world system. This kind of queue is sometimes called a physical queue. For example, you might use a queue to represent a sequence of:

- People standing in line
- Airplanes waiting to access a runway
- Messages waiting to be sent
- Parts waiting to be assembled in a factory
- Computer programs waiting to be executed

In other cases, a queue in a model does not arise in an obvious way from a real-world system, but instead is included for modeling purposes. This kind of queue is sometimes called a logical queue. For

example, you might use a queue to provide a temporary storage area for entities that might otherwise have nowhere to go. Using logical queues can prevent deadlocks or simplify the simulation.

Use the Entity Queue block to model queues.

## Behavior and Features of Servers

In a discrete-event simulation, a server stores entities for a length of time, called the service time, and then attempts to output the entity. 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.

The service time for each entity is computed when it arrives. If, however, the next block does not accept the arrival of an entity that has completed its service, the server is forced to hold the entity longer.

The distinguishing features of servers include:

- The number of entities it can serve simultaneously, which could be finite or infinite
- The characteristics of, or the method of computing, the service times of arriving entities
- Whether the server permits arriving entities to preempt entities that are already stored in the server

---

**Tip** In the absence of preemption, a finite-capacity server does not accept new arrivals when it is already full. You can place a queue before each finite-capacity server to establish a place for entities to stay while they are waiting for the server to accept them. Otherwise, the waiting entities might be stored in various different locations in the model and the situation might be more difficult for you to predict or analyze.

---

### Types of Servers

In some cases, a server in a model is analogous to a real-world system. For example, you might use a server to represent:

- A person (such as a bank teller) who performs a transaction with each arriving customer
- A transmitter that processes and sends messages
- A machine that assembles parts in a factory
- A computer that executes programs

In some cases, a server does not represent a real-world system. A common modeling technique involves a delay of duration zero(that is, an infinite server whose service time is zero) to provide a place for an entity to reside to manipulate its attributes.

Use the Entity Server block to model queues.

Common server use cases of a server include:

- Modeling the processing unit in a production line application
- Representing the processor in a network application

## Modeling with Queues and Servers

You can combine Entity Queue and Entity Server blocks to model different situations:



### Serial Queue-Server Pairs

Connecting two queue-server pairs in series can represent successive operations on an entity. For example, you can model how the parts on an assembly line are processed sequentially by two machines.

You can alternatively model the situation as a pair of servers without a queue between them. However, the absence of the queue means that if the first server completes service on an entity before the second server is available:

- The entity must stay in the first server past the end of service.
- The first server cannot accept a new entity for service until the second server becomes available.

### Parallel Queue-Server Pairs as Alternatives

Connecting two queue-server pairs in parallel, in which entities are routed to one or the other queue-server pair, can represent alternative operations. For example, you can model how vehicles wait in line for one of several tollbooths at a toll plaza. In this case, the model must have decision logic, possibly in the form of a switch that precedes this pattern.

### Parallel Queue-Server Pairs in Multicasting

Connecting two queue-server pairs in parallel, in which a copy of each entity arrives at both, can represent a multicasting situation, such as sending a message to multiple recipients. Note that copying entities might not make sense in some applications.

### Serial Connection of Queues

Connecting two queues in series might be useful if you are using entities to model items that physically experience two distinct sets of conditions while in storage. For example, additional inventory items that overflow one storage area have to stay in another storage area where a less well-regulated temperature affects the items’ long-term quality. Modeling the two storage areas as distinct queue blocks facilitates viewing the average length of time that entities stay in the overflow storage area.

### Parallel Connection of Queues

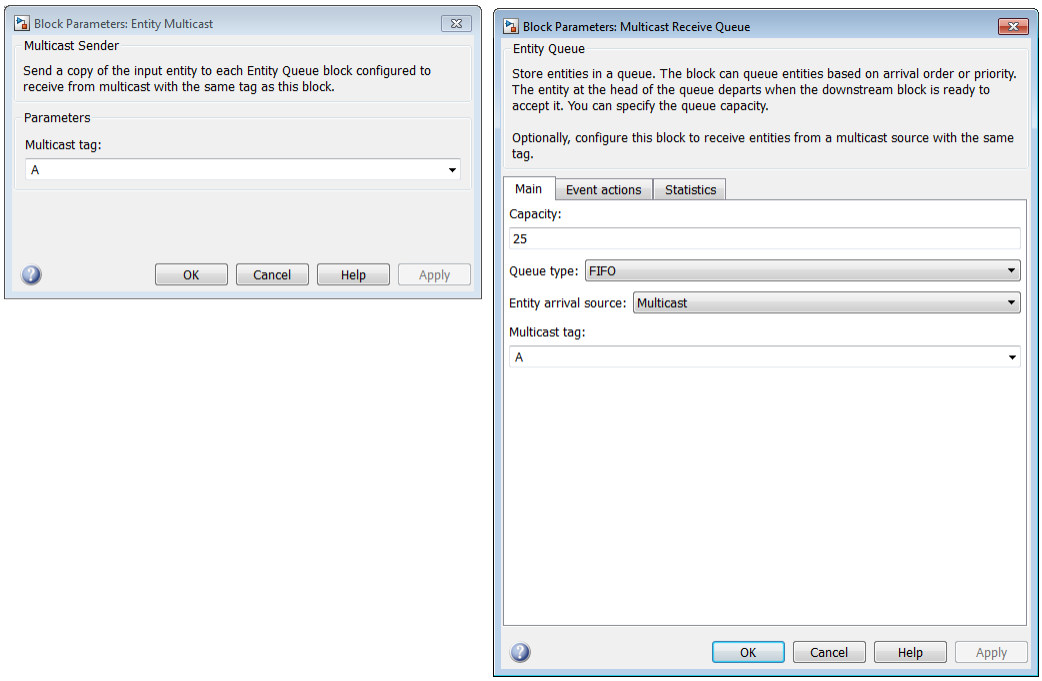
Connecting two queues in parallel, in which each entity arrives at one or the other, can represent alternative paths for waiting. The paths might lead to different operations, such as a line of vehicles waiting for a tollbooth and a line of vehicles waiting on a jammed exit ramp of the freeway. You might model the tollbooth as a server and the traffic jam as a gate.

## Broadcast Entities Using Multicast Mode

Multicast mode enables multiple queues to receive entities from one Entity Multicast block. The receiving block for an Entity Multicast blocks is a Multicast Receive Queue block whose **Tag** parameters have the same value. The Multicast Receive Queue block is essentially the Entity Queue block with the **Entity Arrival source** parameter set to `Multicast`.

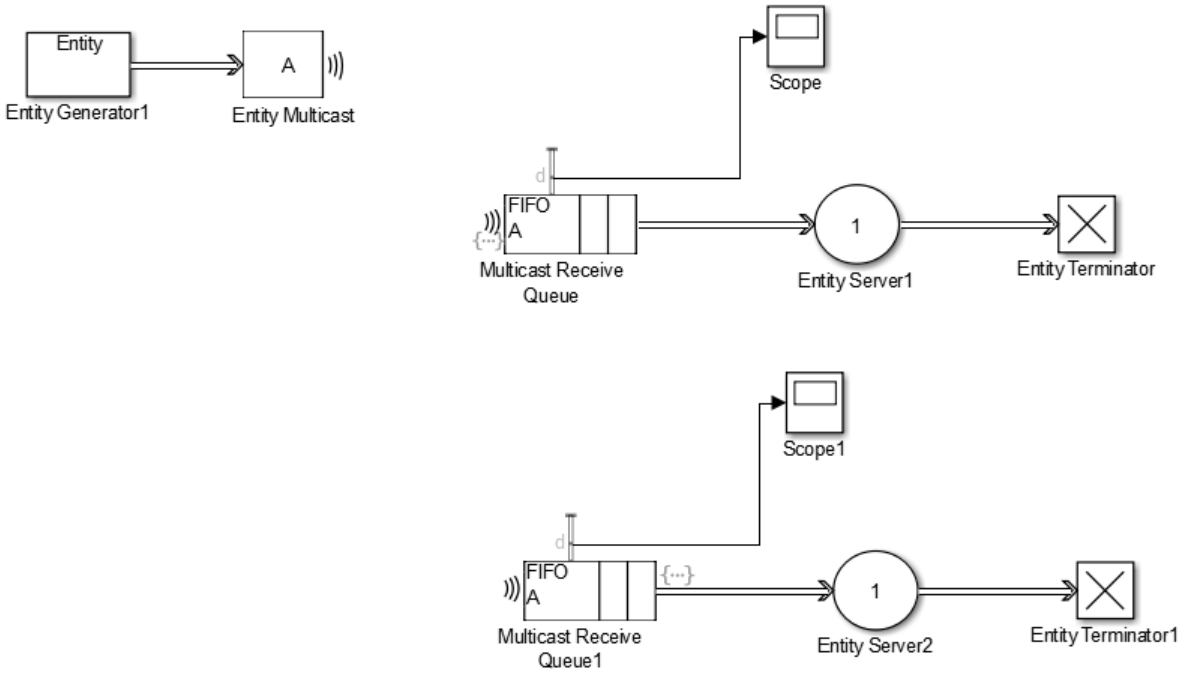
Using the Entity Multicast block requires no connecting lines. The **Tag** parameters just need to match.

1. From the SimEvents library, drag the Entity Multicast and Multicast Receive Queue blocks.
2. In both dialog boxes, in the **Multicast tag** parameters, enter the same text. For example, A.




The software uses these tags to match the broadcaster and broadcastees.

This is example shows entities broadcast to two queues. Notice that the FIFO blocks for both queues have the **A** tag.




## See Also

Entity Multicast | Entity Queue | Entity Server | Multicast Receive Queue | Resource Acquirer | Resource Pool | Resource Releaser

## Related Examples

- “Model Basic Queuing Systems”
- “Model Using Resources”

## More About

- “Events and Event Actions”

# Inspect Statistics

# Count Entities

| In this section... |
| --- |
|  |
|  |
|  |
|  |

Using statistics, you can count entities across the simulation and per time instant.

## Count Departures Across the Simulation

Use the **d** or **a** output from a 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.

## Count Departures per Time Instant

Suppose that you want to visualize entity departures from a particular block, and you want to reset (that is, restart) the counter at each time instant. Visualizing departures per time instant can help you:

- Detect simultaneous departures
- Compare the number of simultaneous departures at different time instants
- Visualize the departure times while keeping the plot axis manageable

For an example of counting simultaneous departures from a server in a cumulative way, see “Count Simultaneous Departures from a Server”.

For an example of noncumulative counting of entity arrivals, see “Noncumulative Counting of Entities”.

## Reset a Counter upon an Event

Suppose that you want to count entities that depart from a particular block, and you want to reset the counter at arbitrary times during the simulation. Resetting the counter can help you compute statistics for evaluating your system over portions of the simulation.

During the simulation, the block counts departing entities and resets its counter whenever the input signal satisfies your specified event criteria.

## Associate Each Entity with Its Index

To associate an entity with its index, in the initialization section of the Entity Generator block, you can associate an entity with its generation time:

1. Use a Simulink Function block with a clock block, such as Digital Clock, to create a Simulink function.

This function returns the current time.

2. In the Entity Generator block, create an attribute and associate it with the current time that the Simulink function returns.

For an example, see **Time stamp entities upon generation** in the SimEvents Design Patterns sublibrary.

## See Also

Entity Queue

## Related Examples

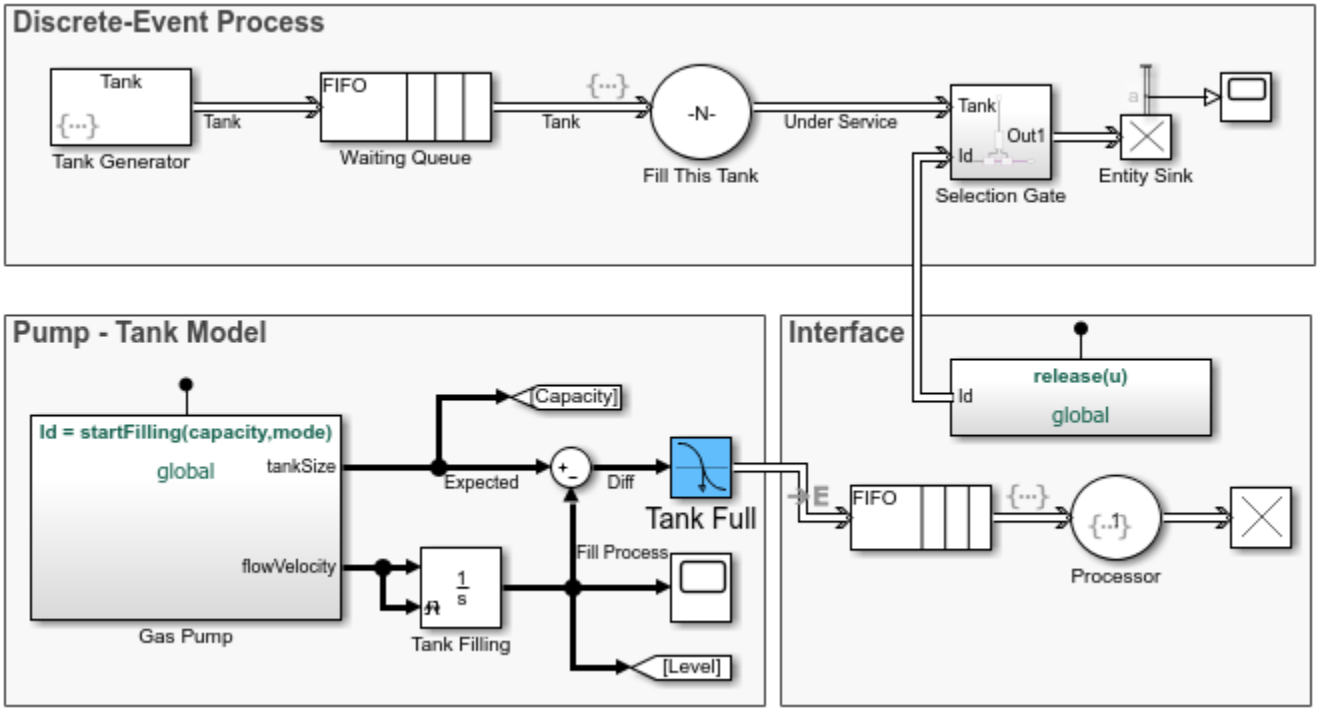
- “Explore Statistics and Visualize Simulation Results” on page 2-6
- “Count Simultaneous Departures from a Server”

# Simulate Multidomain Models

# Create a Hybrid Model with Time-Based and Event-Based Components

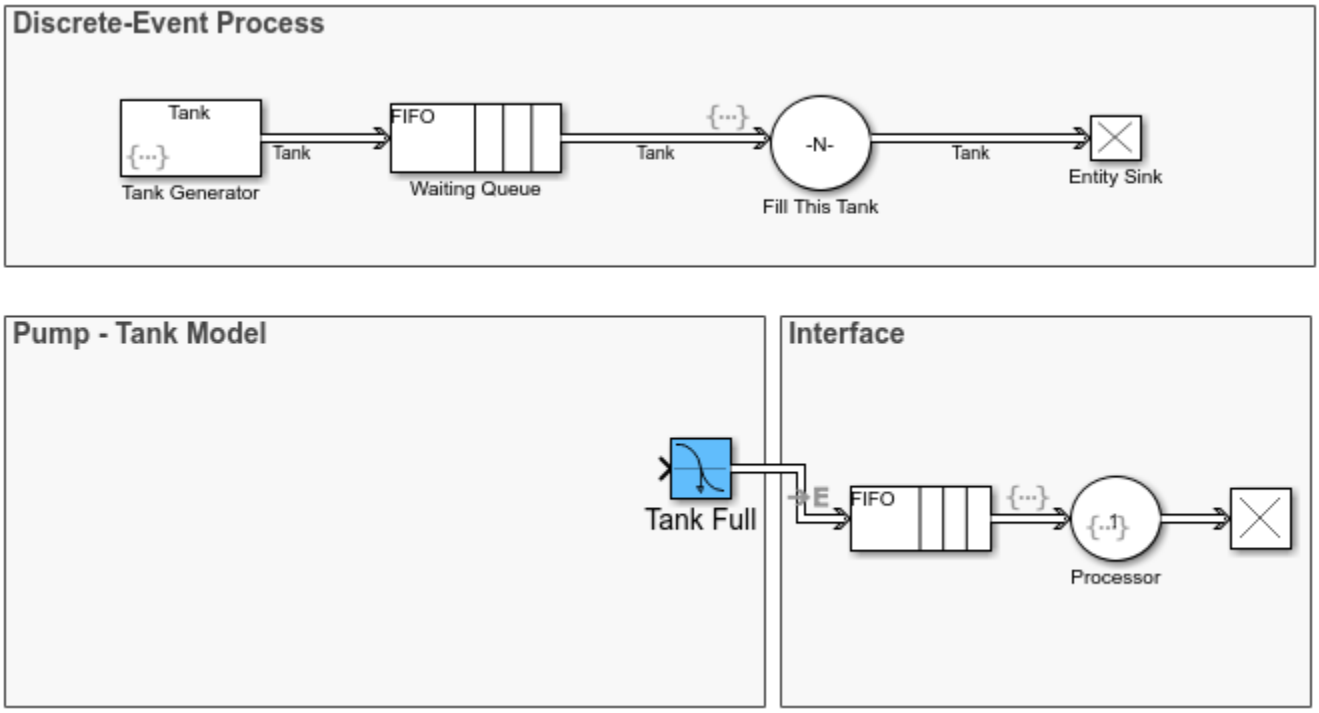
| In this section... |
| --- |
|  |
|  |
|  |
|  |
|  |

The example `seExampleTankFilling` models tanks queuing up to be filled. In the example, SimEvents component models event-based behavior while the Simulink component models time-based dynamics.




## Communication between SimEvents and Simulink components

Without the Selection Gate block and Simulink Function blocks, the flow of tanks in Discrete-Event Process follows their generation, queuing, service, and termination. For more information about building the SimEvents component of the model, see “Create a Discrete-Event Model” on page 2-2. To learn more about writing event actions for the same model, see “Manage Entities Using Event Actions” on page 2-13.




The Pump -Tank model is the Simulink component that represents the time-driven tank filling process. When a tank is full, it generates a SimEvents message through the Hit Crossing block and the message follows a similar flow of generation, queuing, service, and termination. The badge →E denotes the transition between time-based and event-based behavior.

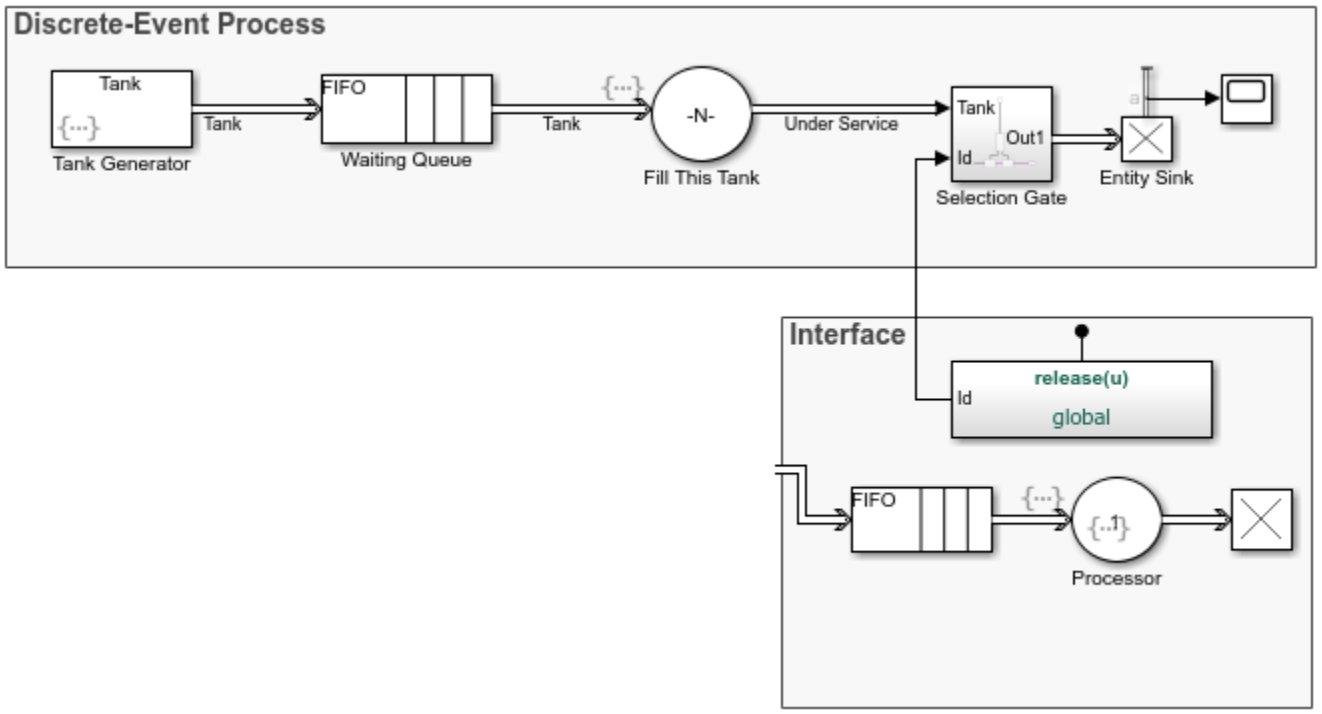
The arrival of a tank at the Entity Server block triggers the filling process in Pump-Tank model. When a tank is full, Hit Crossing block labeled Tank Full generates a message . Arrival of this message at the Processor in Interface component triggers the Simulink Function block to release the Selection Gate for the full tank's departure.

Next, SimEvents and Simulink components of the model are presented in detail.

## SimEvents Part of Model

The SimEvents part models the flow of tanks.

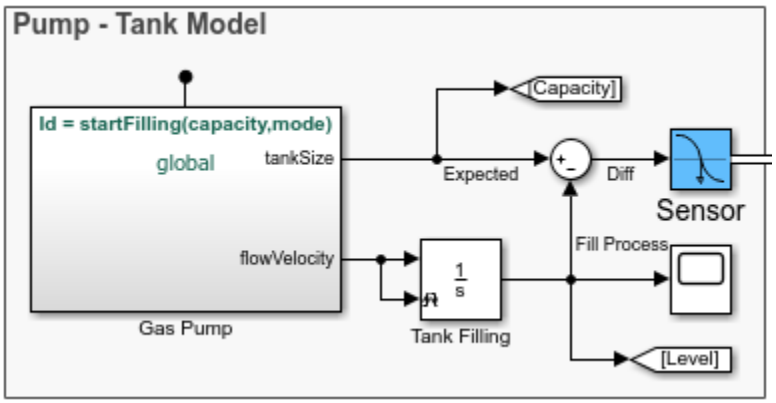
- The Entity Generator block generates the tanks.
- The Entity Queue block queues each tank in first-in-first-out (FIFO) mode.
- The Entity Server block calls the `startFilling` Simulink function to fill each tank. Several tanks can be served at the same time.
- The Entity Server block in the Interface processes the SimEvents message generated by the Hit Crossing block and calls the Simulink function to enable the Selection Gate subsystem for a specific tank. The block also calls the Simulink function to reinitialize the Integrator block for the next fill.




## Simulink Part of Model

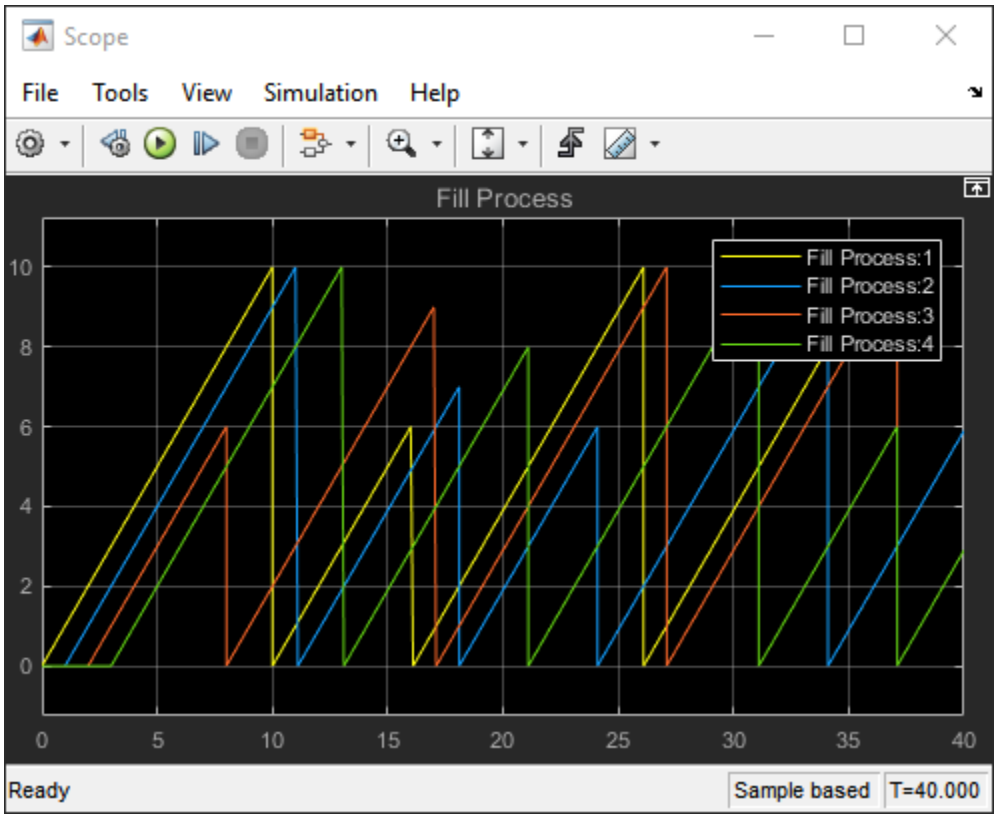
The Simulink part models the time-driven process of filling tanks.

- This component contains the logic to fill the tanks.
- Each tank has a `Capacity` attribute. The continuous time part models the process of filling up a tank, modeled by the Integrator block. When a tank is filled to its capacity, the Selection Gate subsystem releases the tank and the tank departs.
- This component also contains the Simulink function `startFilling`.
- The Hit Crossing block detects the completion of the tank filling process and sends a SimEvents message regarding this event. This message is processed in the Interface, which triggers the release of the tank by the Selection Gate and the reinitialization of the Integrator block for the next fill.

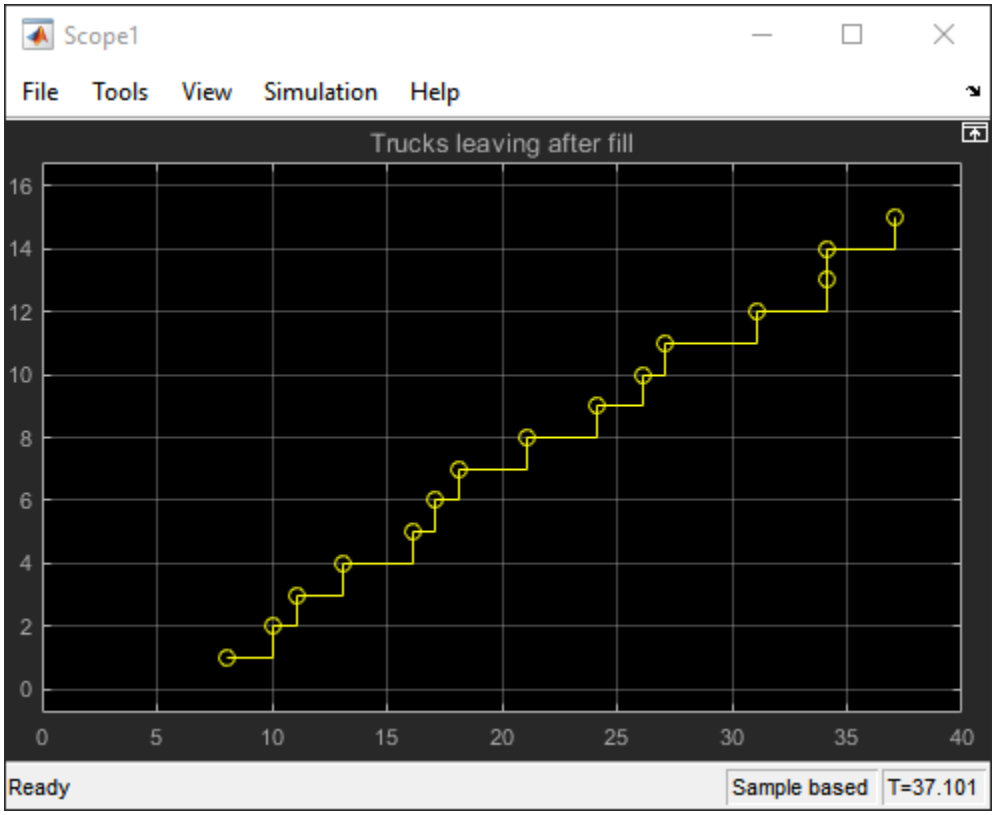


## Simulate the Hybrid Model

Run the `seExampleTankFilling` model. In the first scope, observe the fill process for each pump.




In the second scope, observe the number of trucks leaving after being filled. The plot displays that there are 15 trucks leaving the facility after their gas tanks were filled.




## Event-Based and Time-Based Dynamics in the Simulation

In the `seExampleTankFilling` model, the time-based dynamics of the tank fill coexist with the event-based dynamics of the tank flow system. When you run the simulation, the solver and the event calendar both play a role. Upon major time steps of the solver, the simulation solves the ordinary differential equations that represent the dynamics of the tank fill system. Solving the event-based dynamics entails scheduling and processing events, such as service completion and entity generation, on the SimEvents event calendar. Because the model uses a variable-step solver, when events occur in the discrete-event system, the solver has a major time step.

To learn more about solvers, see “Solvers for Discrete-Event Systems”. To learn more about creating event-based and time-based models, see “Working with SimEvents and Simulink”.

## See Also

Entity Generator | Entity Queue | Entity Server

## More About

- “Generate Entities When Events Occur”
- “Model Basic Queuing Systems”
- “Model Using Resources”
- “Solvers for Discrete-Event Systems”