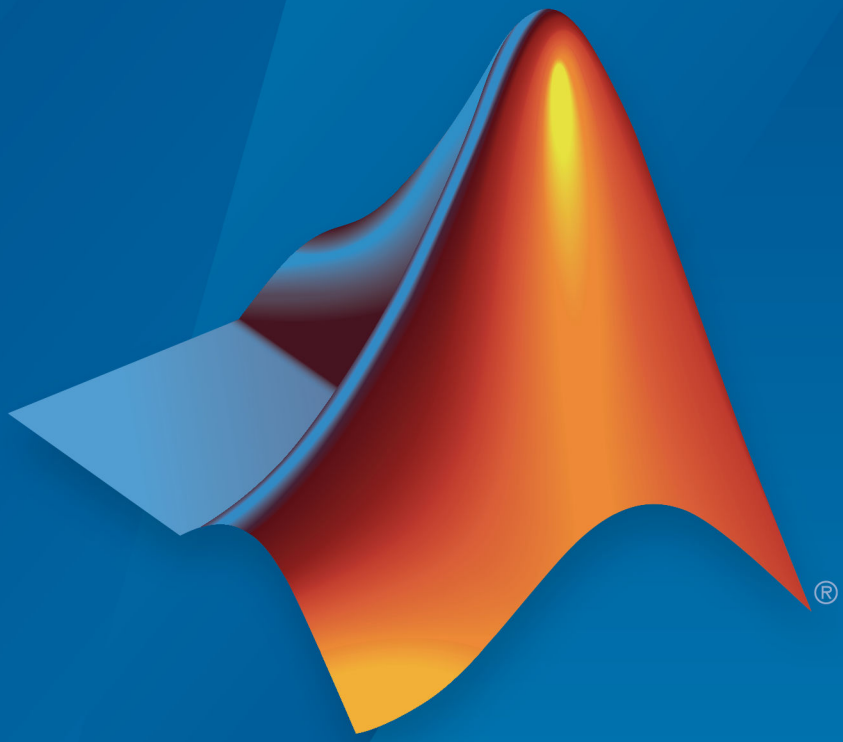
SimEvents®

User's Guide



MATLAB® & SIMULINK®

R2018b

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.
3 Apple Hill Drive
Natick, MA 01760-2098

SimEvents® User's Guide

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)

Contents

1 Working with Entities

2 Modeling Queues and Servers

3 Routing Techniques

4 Work with Resources

5 Visualization, Statistics, and Animation

6 Learning More About SimEvents Software

7 Use SimEvents with Simulink

8 Build Discrete-Event Systems Using Charts

9 Build Discrete-Event Systems Using System Objects

10 Custom Visualization

11 Migrating SimEvents Models

12 Troubleshoot SimEvents Models

Working with Entities

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.

Entity Data Type Support

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). Entities and attributes cannot be a fixed-point data type.

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.

See Also

Composite Entity Creator | 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 | Resource Acquirer | Resource Pool | Resource Releaser

Related Examples

- “Generate Entities When Events Occur” on page 1-11
- “Specify Intergeneration Times for Entities” on page 1-15
- “Manipulate Entity Attributes” on page 1-39
- “Inspect Structures of Entities” on page 1-43
- “Generate Multiple Entities at Time Zero” on page 1-22
- “Count Simultaneous Departures from a Server” on page 1-29
- “Combine Entities” on page 1-46
- “Replicate Entities on Multiple Paths” on page 1-47

More About

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

- “Attribute Value Support” on page 1-54
- “When to Use Bus Objects” (Simulink)

Events and Event Actions

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. You can create event actions to occur when entities change state, for example, when an entity exits a block. For a list of blocks and the actions they can have, see “Storage Actions”.

An event calendar tracks upcoming events for a model during a discrete-event simulation. For more information on the event calendar, see “Event Calendar” on page 6-2.

The event actions assistant helps you create repeated sequence of event actions or random event actions according to a statistical distribution. For more information on the event actions assistant, see “Event Actions Assistant for Events”.

Create Event Actions

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

Entity Generator	Entity Queue	Entity Server	Entity Terminator	Resource Acquirer	Entity Batch Creator
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

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

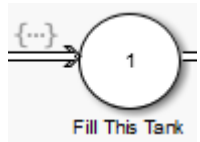
The `seExampleTankFilling` example has two event actions defined, in the Entity Generator and Entity Server blocks. This example recreates the event action in the Entity Server block.

1. In a new model, from the SimEvents library, drag the Entity Server and Simulink Function blocks.
2. In the Entity Server block:
   - Click the **Entity actions** tab.
   - To create an action on entity entry, click **Entry**.
   - In the **Entry action** section, type:

     ```
     startFilling(entity.Capacity);
     ```

     This command calls the function, `startFilling`.

     The ingoing line to the Entity Server block icon updates with the event action icon (`{...}`) indicating that the block defines an event action.

3. In the Simulink Function block:
   a. In Trigger Port, enter `startFilling` in the **Function name** parameter.
   b. Drag in an Inport block and rename it to cap.
   c. Rename the *u* input to `capacity` and connect it to cap.
   d. Remove the *y* output.
   e. Drag in a MATLAB Function block and an Outport block.
   f. In the MATLAB Function, enter the code:

      ```
      function y = toggle()
      %#codegen
      persistent u

      if isempty(u)
          u = -1;
      end
      ```

```
if u == -1
    u = 1;
else
    u = -1;
end

y = u;
```

g Connect the *y* output of the MATLAB Function block to the Outport block and rename the Outport block to reset.

You have now defined the `startFilling` function for the event action. To optionally visualize the connection between the Entity Server block and the Simulink Function block, in the Editor, select **Display > Function Connectors**.

See Also

Composite Entity Creator | 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 | Resource Acquirer | Resource Pool | Resource Releaser

Related Examples

- “Generate Entities When Events Occur” on page 1-11

More About

- “Role of Events in a SimEvents Model”
- “Entities in SimEvents Models”
- “Event Action Languages” on page 1-8
- “Run Computations on Events” on page 1-14
- “Event Calendar” on page 6-2

Event Action Languages

In this section...

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.

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 1-11

More About

- “Role of Events in a SimEvents Model”
- “Mask Editor Overview” (Simulink)

Generate Entities When Events Occur

In this section...

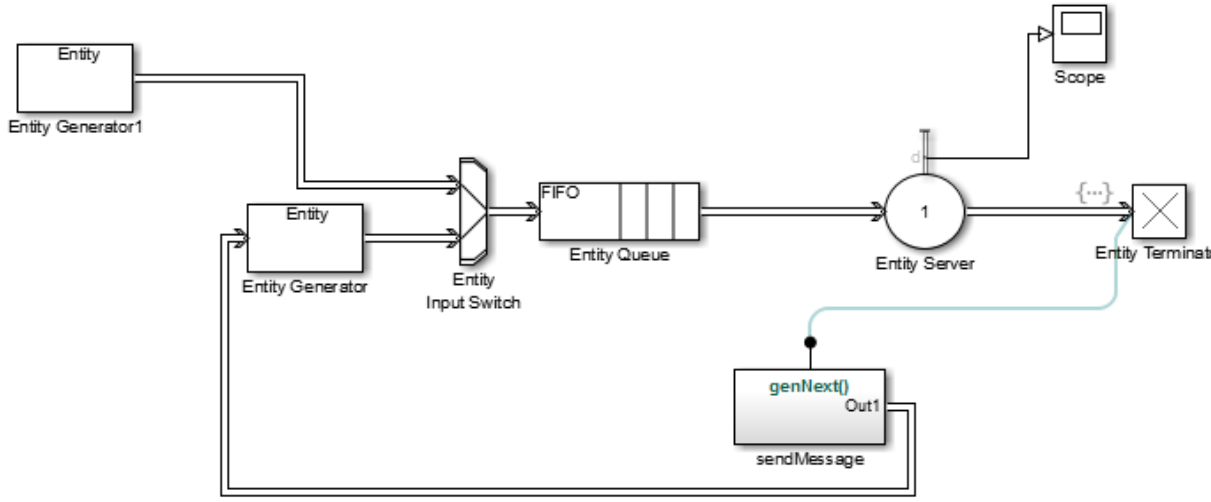
In addition to time-based entity generation, the Entity Generator block enables you to generate entities in response to events that occur during the simulation. In event-based generation, a new entity is generated whenever a message arrives at the input port of the Entity Generator block.

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

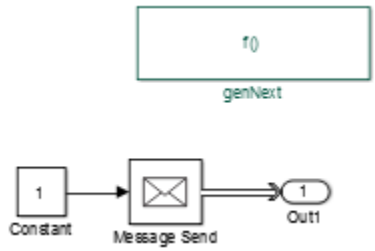
Generating entities when events occur is appropriate if you want the dynamics of your model to determine when to generate entities.

Generate Entity When First Entity is Destroyed

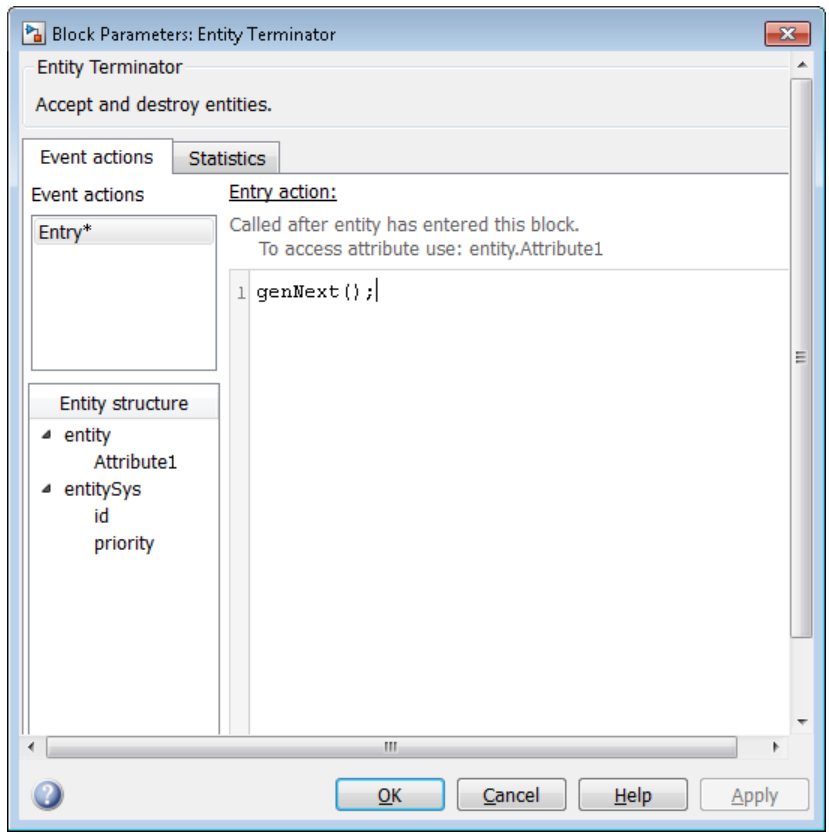
To generate an entity when the first entity is destroyed, use two Entity Generator blocks and a Simulink Function block. The Entity Terminator block calls the Simulink Function after destroying the first entity.



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 of an example that uses an Excel® spreadsheet, see `Generating and Initializing Entities`.

See Also

Composite Entity Creator | 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 | Resource Acquirer | Resource Pool | Resource Releaser

Related Examples

- “Specify Intergeneration Times for Entities” on page 1-15
- “Manipulate Entity Attributes” on page 1-39
- “Inspect Structures of Entities” on page 1-43
- “Generate Multiple Entities at Time Zero” on page 1-22
- “Count Simultaneous Departures from a Server” on page 1-29
- “Combine Entities” on page 1-46
- “Replicate Entities on Multiple Paths” on page 1-47

More About

- “Entities in SimEvents Models”
- “Role of Entity Ports and Paths”
- “Attribute Value Support” on page 1-54

Run Computations on Events

You can run computations on events using event actions by:

- Writing event actions using MATLAB code that perform computations.
- Using the Simulink Function block to call a function that performs computations.

With either of these methods, you can use attribute-defined data to perform the computations.

See Also

More About

- “Role of Events in a SimEvents Model”
- “Event Calendar” on page 6-2

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`

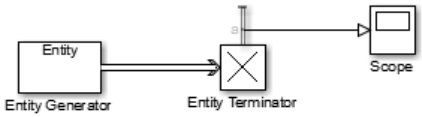
  Uses the **Period** parameter to periodically vary the intergeneration times.
- `Signal port`

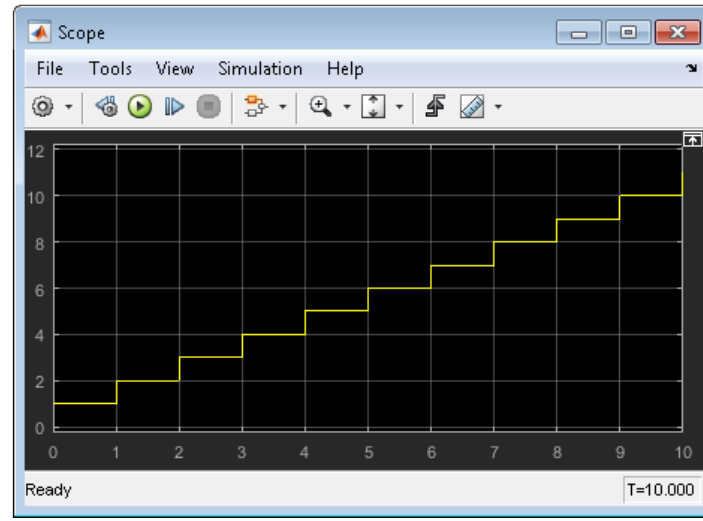
  Uses a signal from an external block, such as the Sine wave block, to vary the intergeneration times.
- `MATLAB action`

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

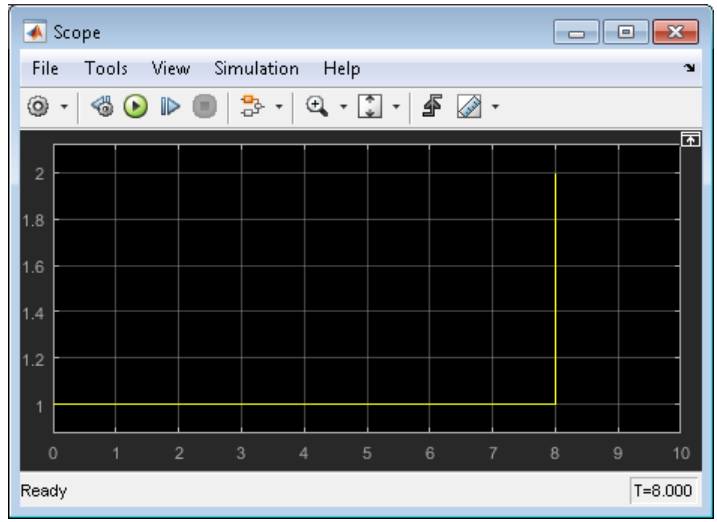
Periodically Vary the Intergeneration Times

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.





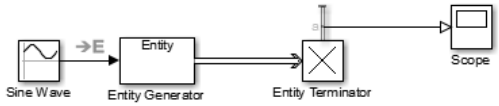
5. Vary the period to 8 and simulate the model again. Observe the change in the scope.

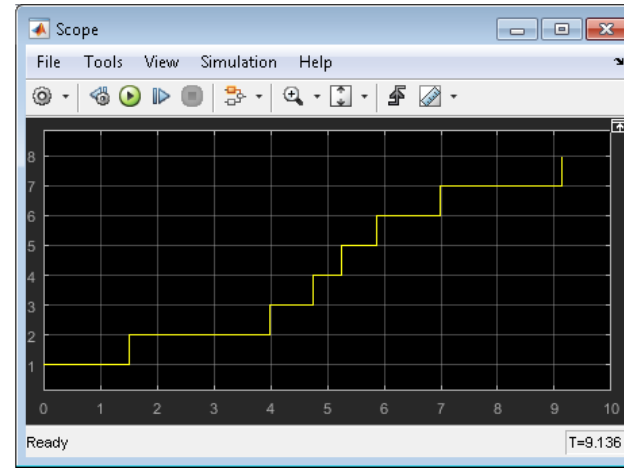


Use a Signal to Vary the Intergeneration Times

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

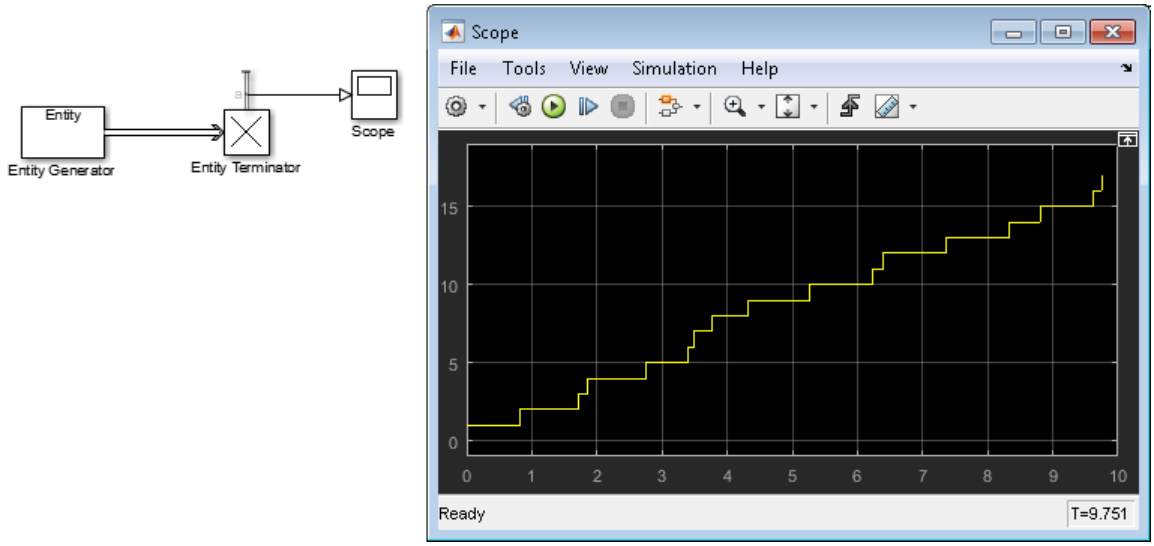
1. In a new model, from the SimEvents library, drag the Entity Generator, Entity Terminator, and Scope blocks.
2. In the **Entity Generation** tab of the Entity Generator, set the **Time source** parameter to `MATLAB action`.

   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();
   ```

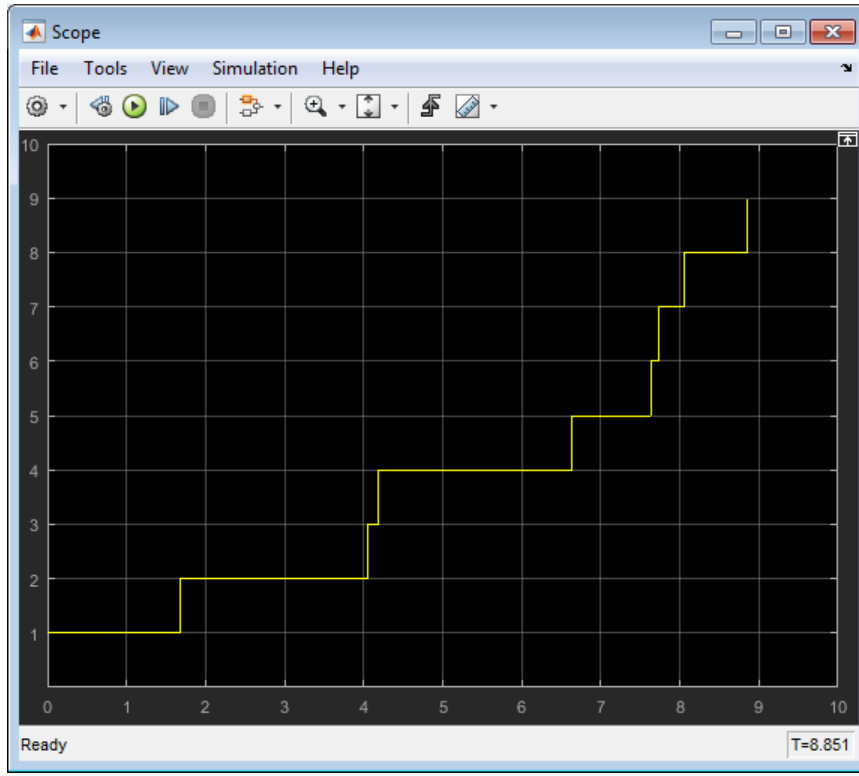
   Note For intergeneration times, you must set the fixed name, *dt*. You cannot set any other variable name for this value.
4. In the Statistics tab of the Entity Terminator block, select the **Number of entities arrived** check box.

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 1-11
- “Manipulate Entity Attributes” on page 1-39
- “Inspect Structures of Entities” on page 1-43
- “Generate Multiple Entities at Time Zero” on page 1-22
- “Count Simultaneous Departures from a Server” on page 1-29

- “Combine Entities” on page 1-46
- “Replicate Entities on Multiple Paths” on page 1-47

More About

- “Entities in SimEvents Models”
- “Role of Entity Ports and Paths”
- “Attribute Value Support” on page 1-54

Generate Multiple Entities at Time Zero

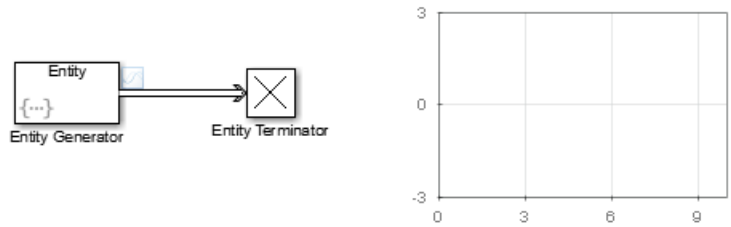
In a discrete-event simulation, an event is an observation of an instantaneous incident that may change a state variable, an output, and/or the occurrence of other events.

Suppose that you want to:

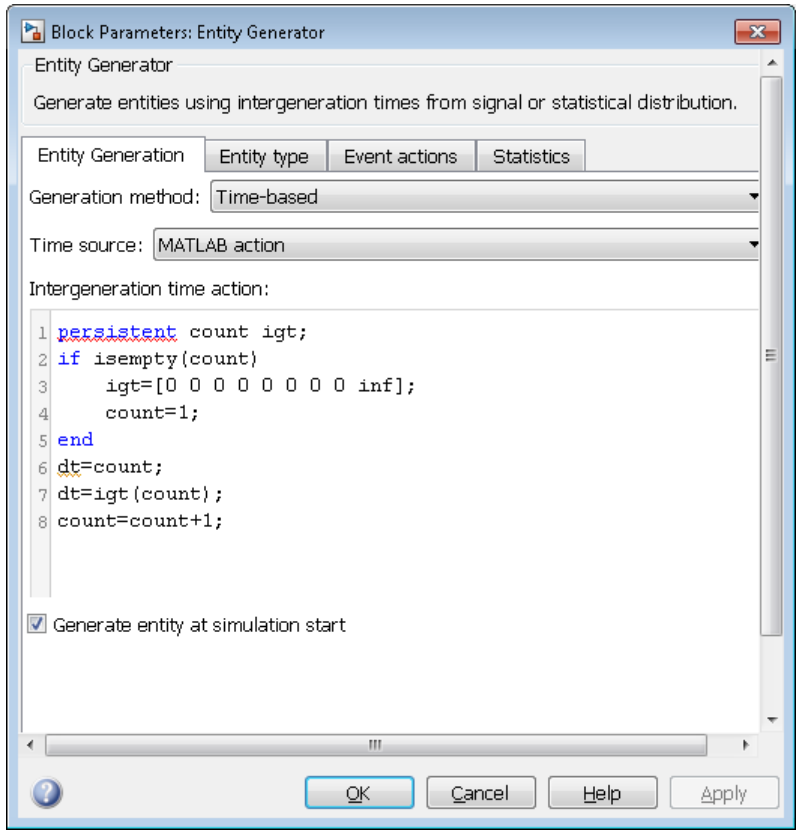
- Preload a queue or server with entities at the start of the simulation, before you analyze queueing or processing delays.
- Initialize the capacity of a shared resource before you analyze resource allocation behavior.

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.

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



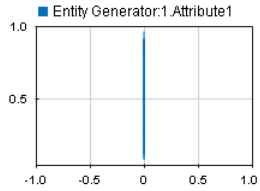
1. In a new model, from the SimEvents library, drag the Entity Generator, Entity Terminator, and Dashboard Scope blocks.
2. Double-click the Entity Generator block.
3. From the **Time source** drop-down list, select `MATLAB action`.
4. In the **Intergeneration time action** field, use MATLAB code to 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

Composite Entity Creator | 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 | Resource Acquirer | Resource Pool | Resource Releaser

Related Examples

- “Generate Entities When Events Occur” on page 1-11
- “Specify Intergeneration Times for Entities” on page 1-15
- “Manipulate Entity Attributes” on page 1-39
- “Inspect Structures of Entities” on page 1-43
- “Count Simultaneous Departures from a Server” on page 1-29
- “Combine Entities” on page 1-46
- “Replicate Entities on Multiple Paths” on page 1-47

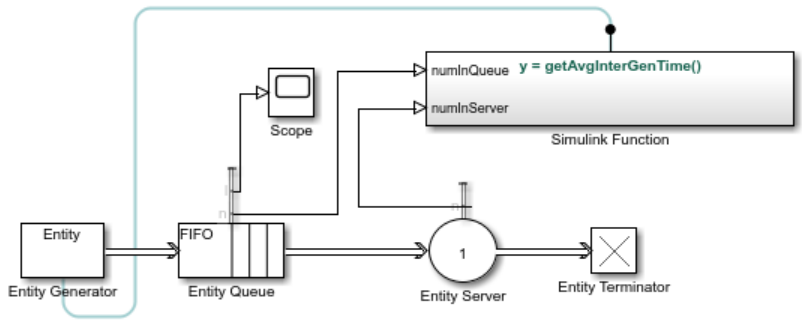
More About

- “Entities in SimEvents Models”
- “Role of Entity Ports and Paths”
- “Attribute Value Support” on page 1-54

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.

1 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());
```

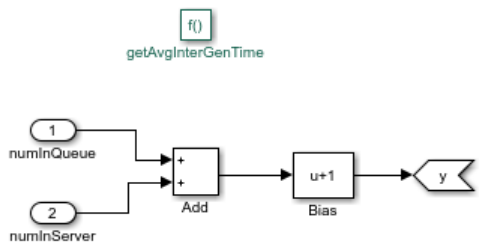
The entity intergeneration time `dt` is generated from an exponential distribution with mean `mu`, which is determined by the function `getAvgInterGenTime()`.

2. 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.
3. 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.

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



   a. Add two In1 blocks and rename them as `numInQueue` and `numInServer`.

      `numInQueue` represents the current number of entities accumulated in the queue and `numInServer` represents the current number of entities accumulated in the server.

b Add a Simulink Add block to add these two inputs.

c Add a Simulink Bias block and set the **Bias** parameter as `1`.

The constant bias `1` is to guarantee a nonzero intergeneration time.

Optionally, select **Display > Function Connections** from the main menu to display the feedback loop from the Simulink Function block to the Entity Generation block.

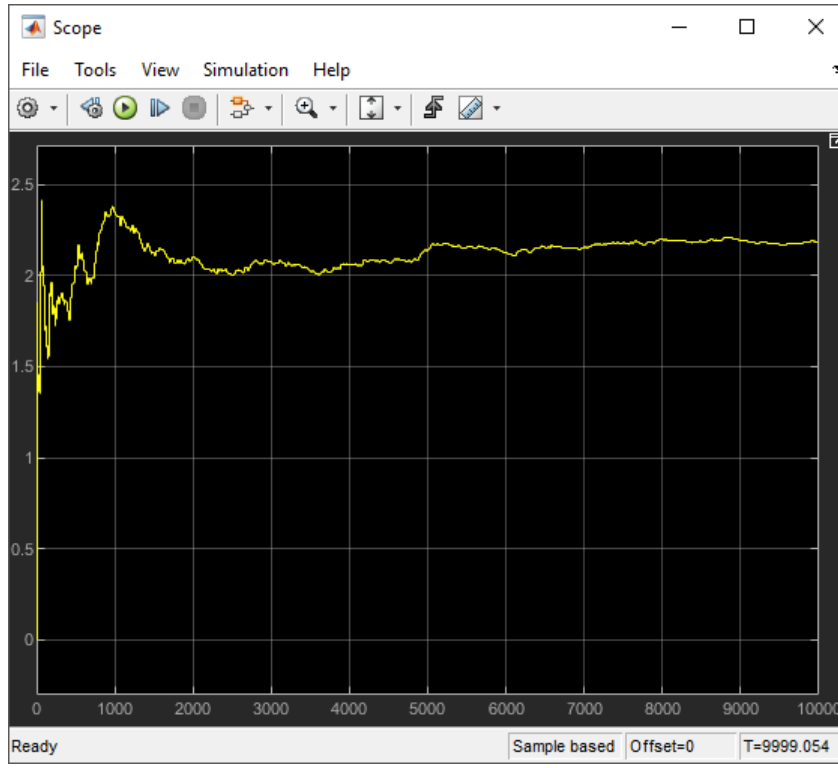
7 In the parent model, connect the **Number of entities in block, n** statistics from the Entity Queue and Entity Server blocks to the Simulink Function block.

8 Connect a Simulink Scope block to the **Average queue length, l** statistic from the Entity Queue block.

The goal is to investigate the average queue length.

9 Increase the simulation time to `10000` and simulate the model.

10 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

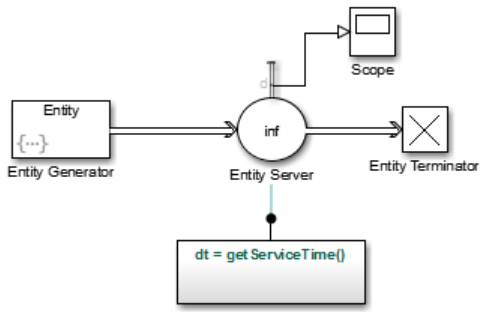
Entity Generator | Entity Queue | Entity Server | Entity Terminator

Related Examples

- “Generate Entities When Events Occur” on page 1-11
- “Generate Multiple Entities at Time Zero” on page 1-22
- “Count Simultaneous Departures from a Server” on page 1-29
- “Replicate Entities on Multiple Paths” on page 1-47

Count Simultaneous Departures from a Server

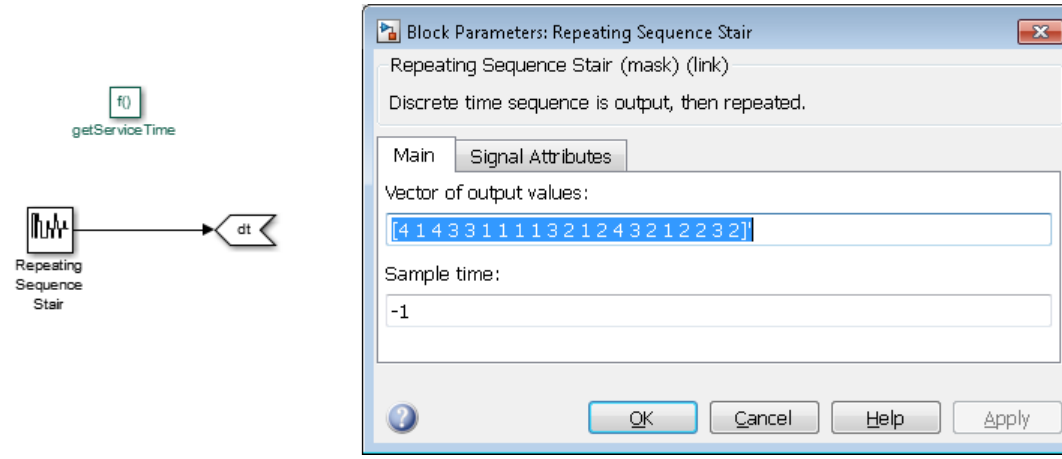
This example shows how to count the simultaneous departures of entities from a server. Use the **d** output from the Entity Server block to learn how many entities have departed (or arrived at) the block. The output signal also indicates when departures occurred. This method of counting is cumulative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



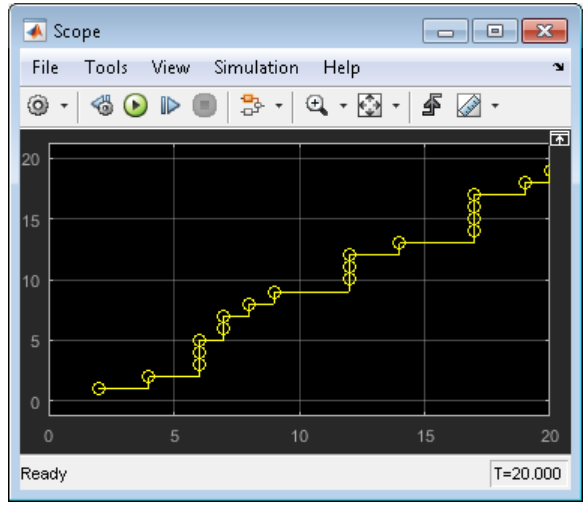
1. In a new model, from the SimEvents library, drag the Entity Generator, Entity Server, Entity Terminator, and Simulink Function blocks. Add a Simulink Scope block.
2. Double-click the Entity Generator block.
   - In the **Event actions** tab, to generate random attribute values, enter:

     ```
     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.



See Also

Composite Entity Creator | 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 | Resource Acquirer | Resource Pool | Resource Releaser

Related Examples

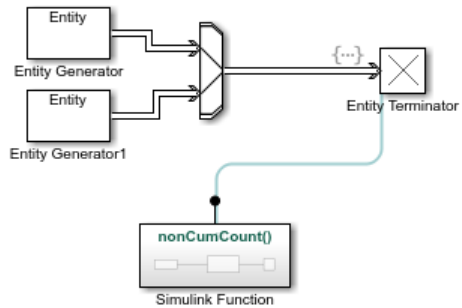
- “Generate Entities When Events Occur” on page 1-11
- “Specify Intergeneration Times for Entities” on page 1-15
- “Manipulate Entity Attributes” on page 1-39
- “Inspect Structures of Entities” on page 1-43
- “Generate Multiple Entities at Time Zero” on page 1-22
- “Combine Entities” on page 1-46
- “Replicate Entities on Multiple Paths” on page 1-47

More About

- “Entities in SimEvents Models”
- “Role of Entity Ports and Paths”
- “Attribute Value Support” on page 1-54

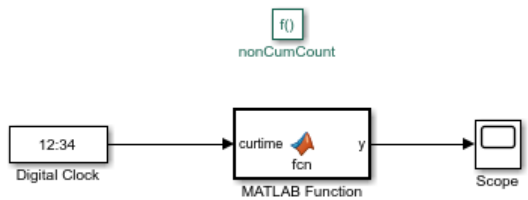
Noncumulative Counting of Entities

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.



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.

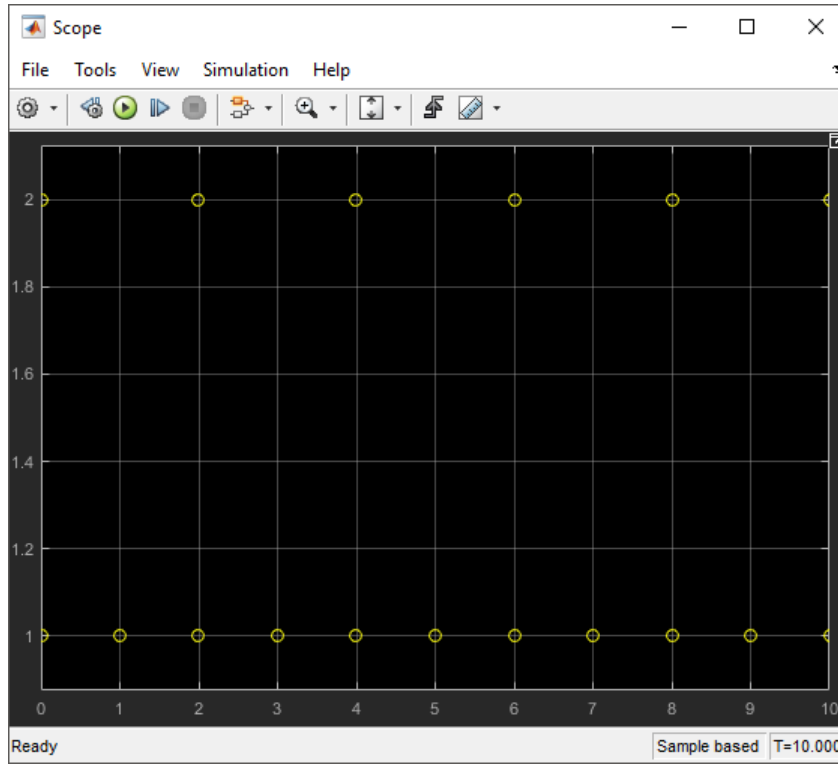
8 In the parent model, double-click the Entity Terminator block. In the **Entry action** field of the **Event actions** tab, enter this code.

```
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 1-29
- “Generate Entities When Events Occur” on page 1-11
- “Specify Intergeneration Times for Entities” on page 1-15
- “Generate Multiple Entities at Time Zero” on page 1-22

More About

- “Entities in SimEvents Models”
- “Role of Entity Ports and Paths”
- “Attribute Value Support” on page 1-54

Working with Entity Attributes

In this section...

You can attach data to an entity using one or more attributes of the entity. 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 also use attributes to specify the amount of a resource for your model. For more information, see “Model with Resources” on page 4-2.

Attach Attributes

To attach attributes to an entity, use the Entity Generator block. Attribute attachments can create new attributes or change the values of existing attributes. You can attach attributes such as:

- Constant value
- 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 that you write
- Values of a signal
- Outputs of a function defined in Simulink or Stateflow® environment that you write.

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:

- Add an attribute manually to attach to the entity.
- Modify an attribute that you added to the table from the **Available Attributes** list to attach to the entity.

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

Button	Action	Notes
	Add a template 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.

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 (when the attribute comes from the dialog box).	Double-click the value, and then type the value you want to assign.

See Also

Discrete Event Chart | Entity Generator | MATLAB Discrete Event System

Related Examples

- “Manipulate Entity Attributes” on page 1-39

More About

- “Entities in SimEvents Models”
- “Attribute Value Support” on page 1-54
- “Combine Entities” on page 1-46

Manipulate Entity Attributes

The attributes table describes some ways that you can use data that you have attached to an entity.

- Create a signal
- Create a plot
- Compute a different attribute value
- Help specify behavior of a block that supports the use of attribute values for block parameters. Examples are the service time for a server and the selected port for an output switch.

Suppose that your entity possesses an attribute with one of these quantities:

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

When an entity with one of these attribute quantities arrives at a server or switch block, you can directly reference the attribute using an option on the server or switch block dialog box. This approach is better than creating a message or signal with the value and delivering it before the entity arrives.

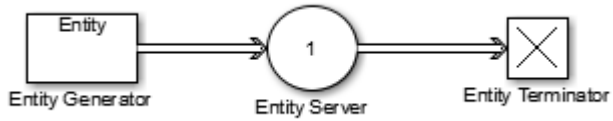
Write Functions to Manipulate Attributes

To manipulate attributes using code, use the **Event actions** tab of a block. In this tab, you can write MATLAB code to manipulate the attribute. To access the attribute, use the notation *`entityName.attributeName`*. For example:

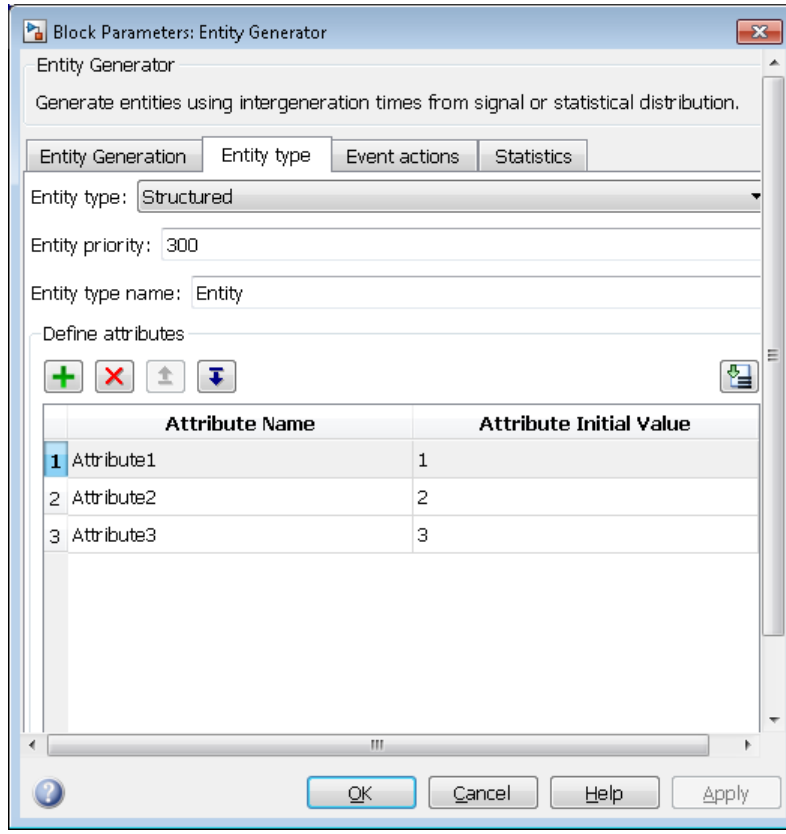
```
entity.Attribute1=5;
```

For example, you might want to manipulate the attributes for service completion.

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

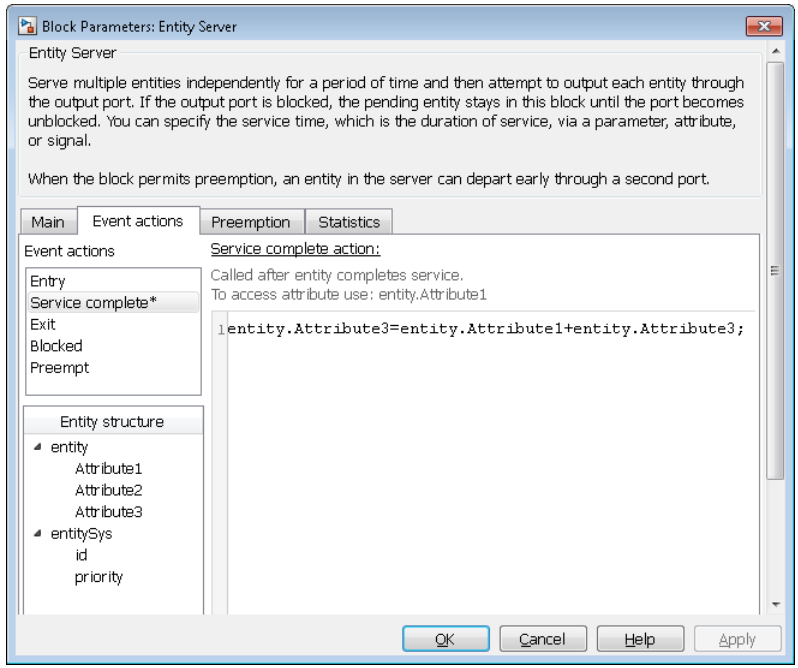


2 Double-click Entity Generator and, in the **Entity type** tab, add three attributes to the attributes table.



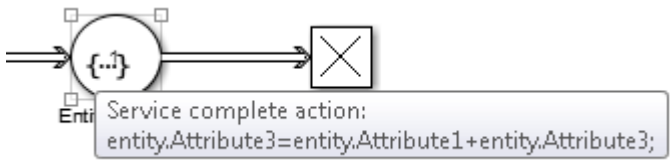
3 In the Entity Server block, click the **Event actions** tab.

4 For the `Service complete` action, enter MATLAB code to manipulate the entity attributes you added in the Entity Generator block. For example:



This code updates the Entity Server block with the event action icon.

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



See Also

Discrete Event Chart | Entity Generator | MATLAB Discrete Event System

Related Examples

- “Manipulate Entity Attributes” on page 1-39

More About

- “Entities in SimEvents Models”
- “Working with Entity Attributes” on page 1-36
- “Attribute Value Support” on page 1-54

Inspect Structures of Entities

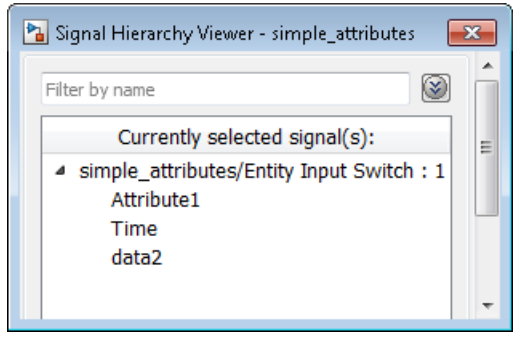
You can inspect entity structures using these methods:

- On a signal line, using the Signal Hierarchy Viewer (for more information, see “Display Entity Types” on page 1-43).
- In a block at run-time, using the Entity Inspector

Display Entity Types

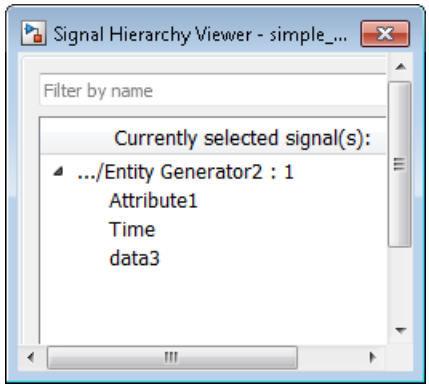
To show entity types in your model, in the model editor, right-click a line and select **Signal Hierarchy**. The Signal Hierarchy Viewer interactively displays about entities, signals, and bus objects. For more information on the Signal Hierarchy Viewer, see “Signal Hierarchy Viewer” (Simulink).

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.

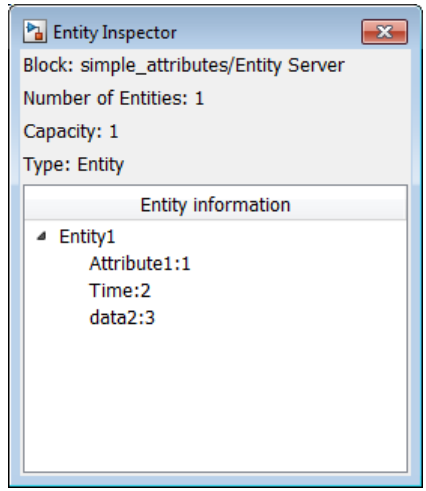
If you now modify the second Entity Generate block path to change `data2` to `data3`, the structure of entities on the second path becomes unique. You must specify a new entity type name for the second Entity Generator block.



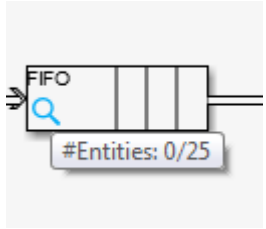
Inspect Entities at Run Time

To inspect entities at run-time, use the Entity Inspector. Inspect 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

- “Entities in SimEvents Models”
- “Entity Types” on page 1-2
- “Role of Entity Ports and Paths”
- “Attribute Value Support” on page 1-54

Combine Entities

You can combine entities from different paths using the Composite Entity Creator block. The entities that you combine, called component 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 and its surrounding blocks automatically detect 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 SimEvents Models”

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 SimEvents Models”

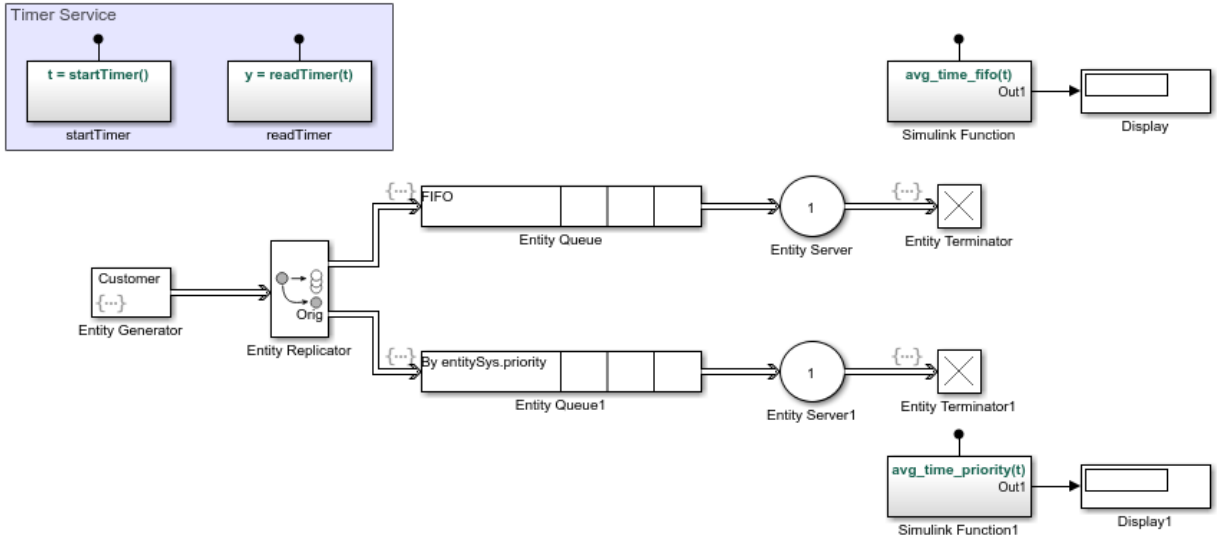
Measure Point-to-Point Delays

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.

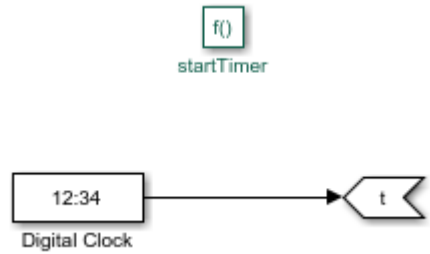
Basic Example Using Timer Blocks

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.

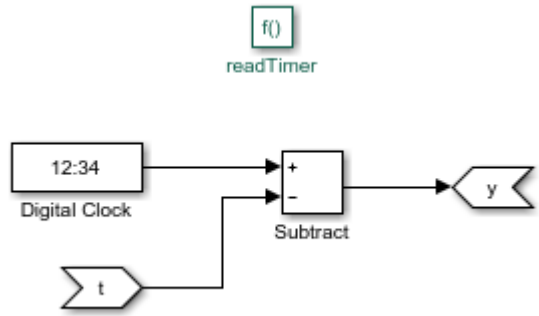


This example has four Simulink Function blocks. Two define timer functions, startTimer and readTimer. The other functions calculate average times.

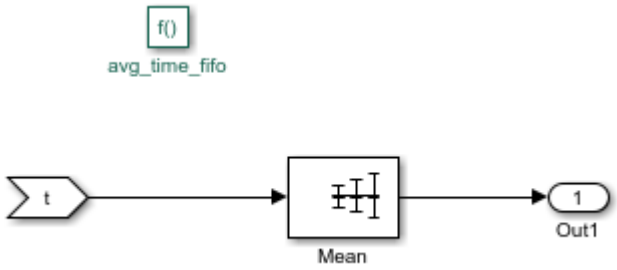
1. In a new model, drag the blocks shown in the example and relabel and connect them as shown.
2. For the startTimer block, define:



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:

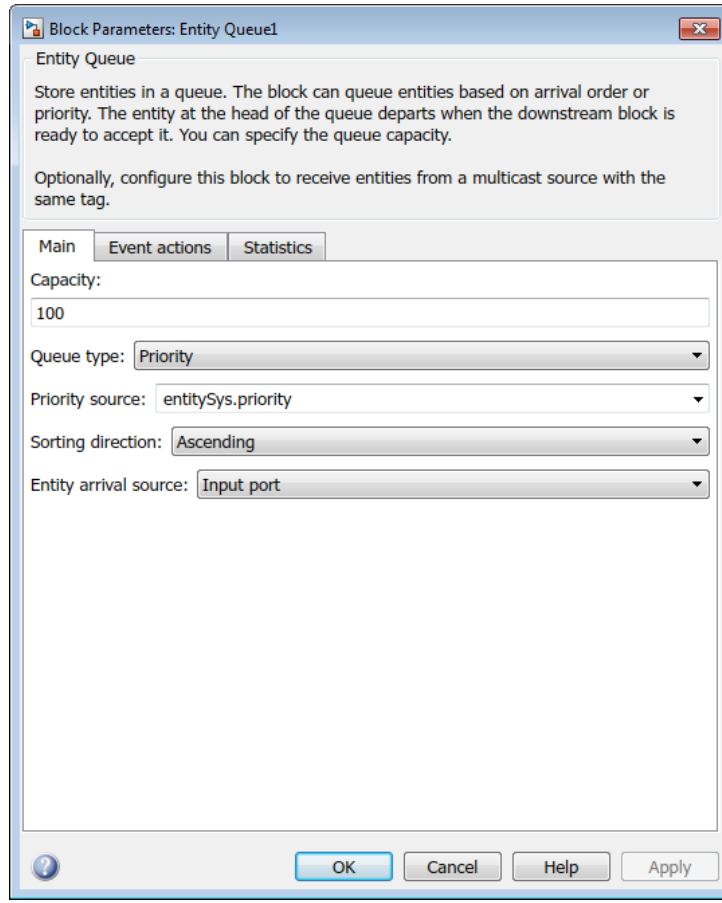
   a In the **Main** tab, set **Queue type** to FIFO.

   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:

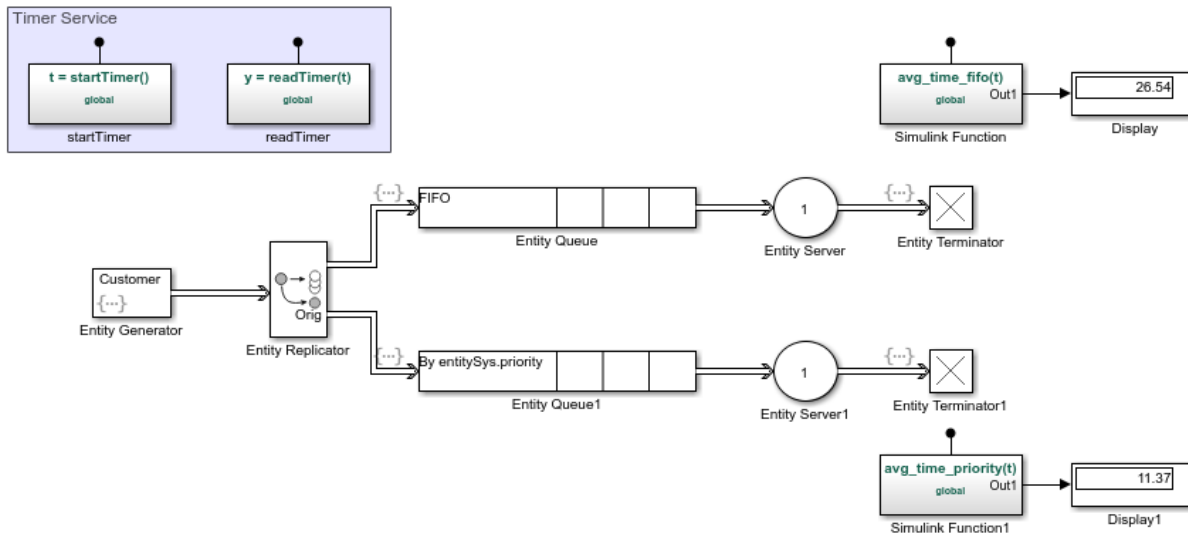
```
% Read timer
elapsedTime = readTimer(entity.Timer);
```

```
% 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 SimEvents Models”

Attribute Value Support

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

Permitted Characteristics of Attribute Values

- Real or complex
- Array 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.

Not Permitted as Attribute Values

- Structure
- Bus
- Variable-size signals or variable-size arrays
- Frame

See Also

Discrete Event Chart | Entity Generator | MATLAB Discrete Event System

Related Examples

- “Manipulate Entity Attributes” on page 1-39

More About

- “Entities in SimEvents Models”
- “Working with Entity Attributes” on page 1-36

Modeling Queues and Servers

Model Basic Queuing Systems

In this section...

Example of a Logical Queue

Suppose that you are modeling a queue that can physically hold 100 entities and you want to determine what proportion of the time the queue length exceeds 10. You can model the long queue as a pair of shorter queues connected in series. The shorter queues have length 90 and 10.

Although the division of the long queue into two shorter queues has no basis in physical reality, it enables you to gather statistics related to one of the shorter queues. In particular, you can view the queue length (**n**) of the queue having length 90. If the signal is positive over a nonzero time interval, then the length-90 queue contains an entity that cannot advance to the length-10 queue. This means that the length-10 queue is full. As a result, the physical length-100 queue contains more than 10 items. Determining the proportion of time the physical queue length exceeds 10 is equivalent to determining the proportion of time the queue length signal of the logical length-90 queue exceeds 0.

Vary the Service Time of a Server

You can vary the service time of a server using one of the following methods:

- Constant source, where you vary the constant
- Randomized source
- Arbitrary source
- Time-based source

Use the **Service time source** parameter of the Entity Server block to apply these methods. You can select from:

- `Dialog`

  Enter the constant value in the **Service time value** parameter.

- `Signal port`

  Connect a time source to the resulting signal port.

- `Attribute`

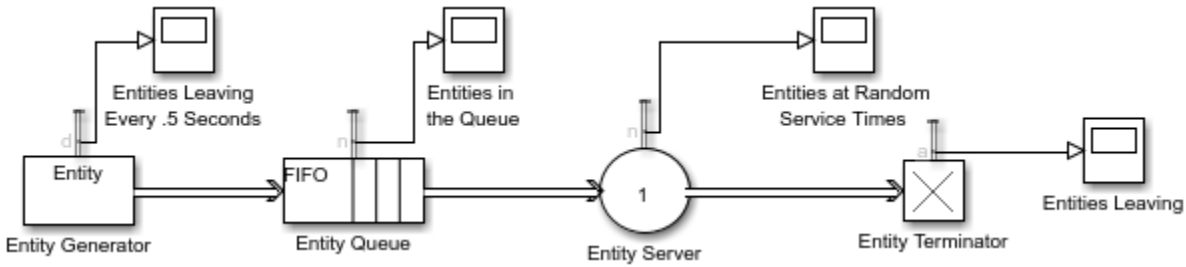
  Enter the name of the attribute that contains data to be interpreted as service.

- `MATLAB action`

  In the **Service time action** section, enter MATLAB code to vary the service time. Assign the variable *dt*, which the model uses as service time.

Random Service Times

This example is a simple queuing system in which entities arrive at a fixed deterministic rate. They then wait in a queue and advance to a server that services the entities at random intervals. It illustrates use of the `Service time from random distribution design` pattern.



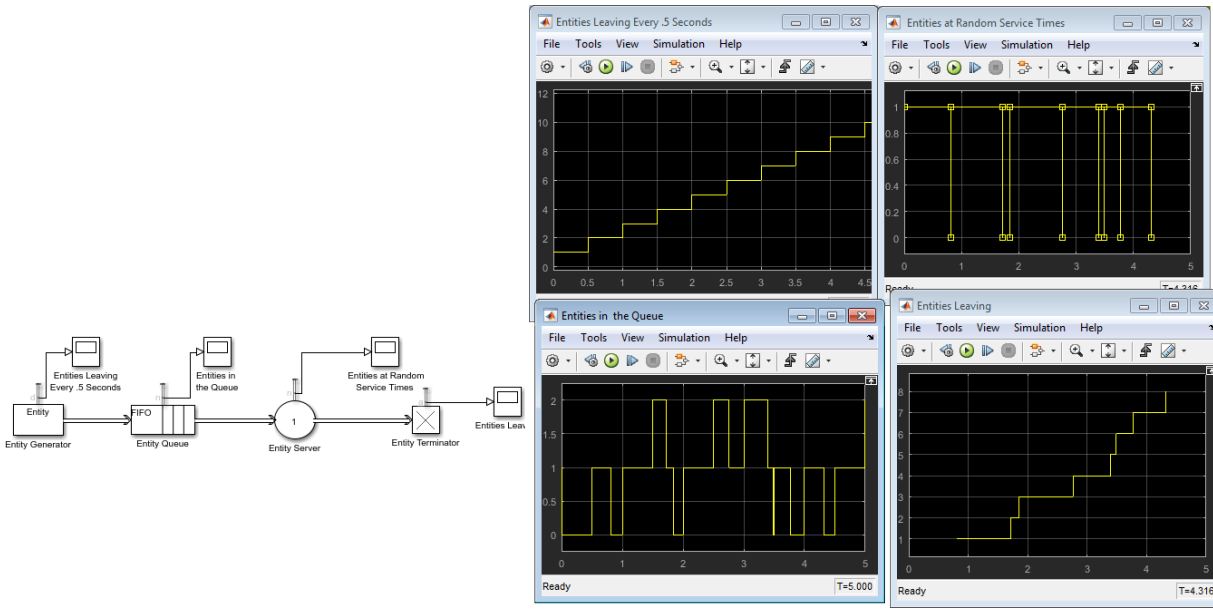
1. In a new model, drag the blocks shown in the example and relabel and connect them as shown. For convenience, start with the `Service time from random distribution` design pattern
2. To generate entities every .5 seconds, in the Entity Generator block:
   a. In the **Entity Generation** tab, change the **Period** to `.5`.
   b. In the **Statistics** tab, select **Number of entities departed, d**.
3. In the Entity Queue block, select **Number of entities in block, n**.
4. In the Entity Server block:

a Verify that the server is configured for random service time. If not, copy the Server block from the `Service time from random distribution` design pattern.

b In the **Statistics** tab, select **Number of entities in block, n**.

5 In the Entity Terminator block, in the Statistics tab, select **Number of entities arrived, a**.

6 Save and run the model. In particular, observe the pattern of the entities leaving the Entity Generator block and the entities at random service times.



See Also

Entity Queue | Entity Server

Related Examples

- “Sort by Priority” on page 2-6
- “Task Preemption in a Multitasking Processor” on page 2-8
- “Determine Whether a Queue Is Nonempty” on page 2-11

- “Model Server Failure” on page 2-12

More About

- “Storage with Queues and Serves”

Sort by Priority

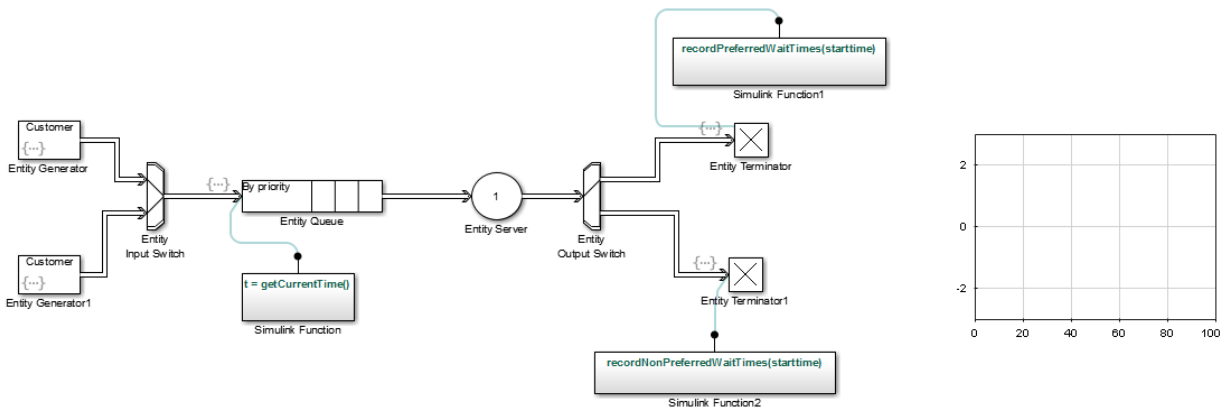
In this section...

Behavior of Priority Mode of Entity Queue Block

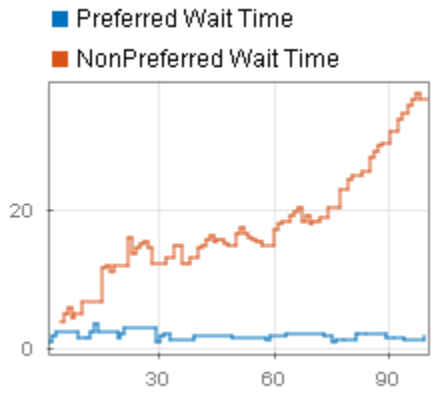
The `Priority` mode of the Entity Queue block supports queuing in which entities positions in the queue are based primarily on specific attribute values. Arrival times are relevant only when attribute values are equal. You specify the attribute and the sorting direction using the **Priority source** and **Sorting direction** parameters in the block dialog box.

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.



You can see from the plots that despite the shorter service time, the average system time for the nonpreferred customers is much longer than the average system time for the preferred customers.



Comparison with Unsorted Behavior

If the queue used a FIFO discipline for all customers instead of a priority sorting, then the average system time would decrease slightly for the nonpreferred customers and increase markedly for the preferred customers.

See Also

Entity Queue | Entity Server

Related Examples

- “Model Basic Queuing Systems” on page 2-2
- “Task Preemption in a Multitasking Processor” on page 2-8
- “Determine Whether a Queue Is Nonempty” on page 2-11
- “Model Server Failure” on page 2-12

More About

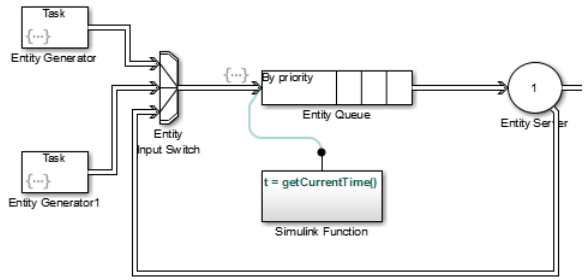
- “Storage with Queues and Serves”

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.

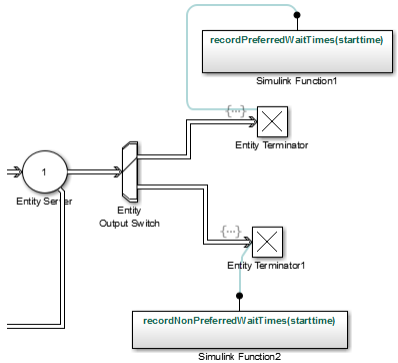
The example shows preemption—replacement—of low priority tasks by a high priority task in a multitasking processor. An Entity Server block represents the task processor presented with a capacity to process multiple concurrent tasks.

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.

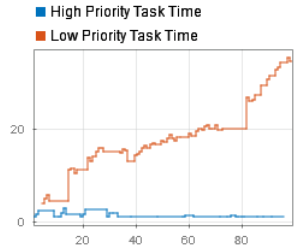
An Entity Server block represents a multitasking processor with capacity for multiple tasks.



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 scope plots show the simulation results.



See Also

Entity Queue | Entity Server

Related Examples

- “Model Basic Queuing Systems” on page 2-2
- “Sort by Priority” on page 2-6
- “Determine Whether a Queue Is Nonempty” on page 2-11
- “Model Server Failure” on page 2-12

More About

- “Storage with Queues and Serves”

Determine Whether a Queue Is Nonempty

To determine whether a queue is storing any entities, use this technique:

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.

See Also

Entity Queue | Entity Server

Related Examples

- “Model Basic Queuing Systems” on page 2-2
- “Sort by Priority” on page 2-6
- “Task Preemption in a Multitasking Processor” on page 2-8
- “Model Server Failure” on page 2-12

More About

- “Storage with Queues and Serves”

Model Server Failure

In this section...

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.

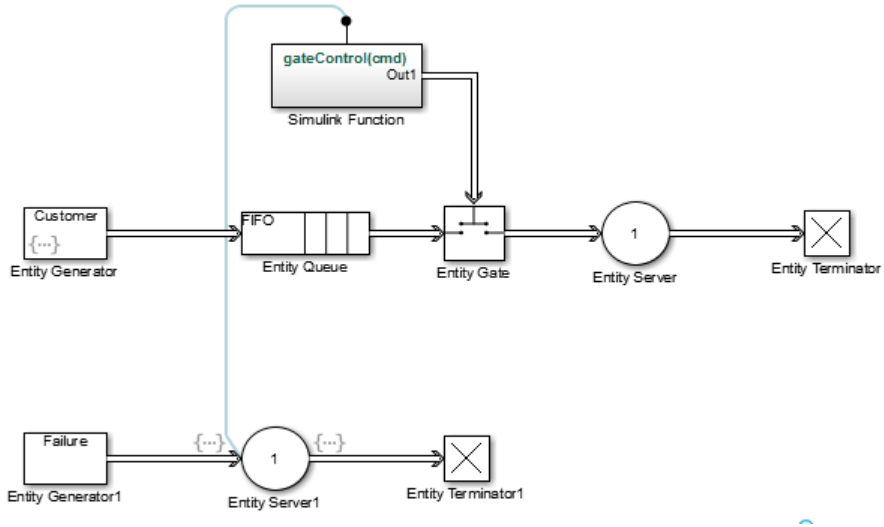
Server as Communication Channel	Server as Machine	Server as Human Processor
Transmitting message	Processing part	Working
Connected but idle	Waiting for new part to arrive	Waiting for work
Unconnected	Off	Off duty
Holding message (pending availability of destination)	Holding part (pending availability 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 A 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
- "Sort by Priority" on page 2-6
- "Task Preemption in a Multitasking Processor" on page 2-8
- "Determine Whether a Queue Is Nonempty" on page 2-11

More About

- “Storage with Queues and Serves”

Routing Techniques

Role of Paths in SimEvents Models

In this section...

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, any entity that departs from the output port of an Entity Queue block set to FIFO mode equivalently arrives at an Entity Server block 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 that a given entity arrives at, throughout the simulation.

In many discrete-event models, however, the set of entity connection lines does not completely determine the sequence of blocks that each entity arrives at.

By looking at entity connection lines alone, you cannot tell which queue block's input port an entity will arrive at. Instead, you need to know more about how the Entity Output Switch block behaves and you might even need to know the outcome of certain run-time decisions.

Overview Blocks for Designing Paths

You design entity paths by choosing or combining entity paths using these blocks:

- Entity Input Switch
- Entity Output Switch
- Entity Replicator

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 — for example, repeating an operation if quality criteria such as quality of service (QoS) are not met. You can use the Entity Input Switch block to combine the paths of new entities and entities that require a repeated operation.
- To incorporate logical decision making into your simulation — for example, determining scheduling protocols. You might use the Entity Input Switch block to determine which of several queues receives attention from a server.

Other blocks in the SimEvents library have secondary features, such as preemption from a server, that give you opportunities to design paths.

See Also

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

Related Examples

- “Select Departure Path Using Entity Output Switch” on page 3-5
- “Select Arrival Path Using Entity Input Switch” on page 3-8
- “Combine Entity Paths” on page 3-10
- “Use Messages To Route Entities” on page 3-12
- “Enable a Gate for a Time Interval” on page 3-22

More About

- “Use Attributes to Route Entities” on page 3-19
- “Role of Gates in SimEvents Models” on page 3-20

Select Departure Path Using Entity Output Switch

In this section...

Role of the Entity Output Switch

The Entity Output Switch block selects one among a number of entity output ports. The selected port can change during the simulation. You have several options for criteria that the block uses to select an entity output port.

When the selected port is not blocked, an arriving entity departs through this port.

Sample Use Cases

Here are some scenarios in which you might use an output switch:

- Entities advance to one of several queues based on efficiency or fairness concerns. For example, airplanes advance to one of several runways depending on queue length, or customers advance to the first available cashier out of several cashiers.

  Comparing different approaches to efficiency or fairness, by testing different rules to determine the selected output port of the output switch, might be part of your goal in simulating the system.
- Entities advance to a specific destination based on their characteristics. For example, parcels advance to one of several delivery vehicles based on the locations of the specified recipients.
- Entities use an alternate route in case the preferred route is blocked. For example, a communications network drops a packet if the route to the transmitter is blocked and the simulation gathers statistics about dropped packets.

The topics listed below illustrate the use of the Entity Output Switch block.

Topic	Features of Example
“Select the First Available Server” on page 3-6	`First port that is not blocked` switching criterion
“Use an Attribute to Select an Output Port” on page 3-6	Attribute-based switching, where the attribute value is random

Select the First Available Server

Assume an example where entities arriving at the Entity Output Switch block depart through the first entity output port that is not blocked, as long as at least one entity output port is not blocked. An everyday example of this approach is a single queue of people waiting for service by one of several bank tellers, cashiers, call center representatives, etc. Each person in the queue wants to advance as soon as possible to the first available service provider without preferring one over another.

You can implement this approach by setting the **Switching criterion** parameter in the Entity Output Switch block to `First port that is not blocked`.

Use an Attribute to Select an Output Port

Consider the situation in which parcels are sorted among several delivery vehicles based on the locations of the specified recipients. If each parcel is an entity, then you can attach data to each entity to indicate the location of its recipient. To implement the sorting, set the **Switching criterion** parameter in the Entity Output Switch block to `From attribute`.

See Also

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

Related Examples

- “Select Arrival Path Using Entity Input Switch” on page 3-8
- “Combine Entity Paths” on page 3-10
- “Use Messages To Route Entities” on page 3-12
- “Enable a Gate for a Time Interval” on page 3-22

More About

- “Role of Paths in SimEvents Models” on page 3-2
- “Use Attributes to Route Entities” on page 3-19
- “Role of Gates in SimEvents Models” on page 3-20

Select Arrival Path Using Entity Input Switch

In this section...

Role of the Input Switch

The Entity Input Switch chooses among a number of entity input ports. This block selects exactly one entity input port for potential arrivals and makes all other entity input ports unavailable. The selected entity input port can change during the simulation. You have several options for criteria that the block uses for selecting an entity input port.

A typical scenario in which you might use an input switch is when multiple sources of entities feed into a single queue, where the sequencing follows specific rules. For example, users of terminals in a time-shared computer submit jobs to a queue that feeds into the central processing unit, where an algorithm regulates access to the queue so as to prevent unfair domination by any one user.

Round-Robin Approach to Choosing Inputs

In a round-robin approach, an input switch cycles through the entity input ports in sequence. After the last entity input port, the next selection is the first entity input port. The switch selects the next entity input port after each entity departure. When the switch selects an entity input port, it makes the other entity input ports unavailable, regardless of how long it takes for an entity to arrive at the selected port.

You can implement a round-robin approach by

1. Setting the **Active port selection** parameter to `Switch`.
2. Setting the **Switching criterion** parameter to `Round robin`.

See Also

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

Related Examples

- “Select Departure Path Using Entity Output Switch” on page 3-5
- “Combine Entity Paths” on page 3-10
- “Use Messages To Route Entities” on page 3-12
- “Enable a Gate for a Time Interval” on page 3-22

More About

- “Role of Paths in SimEvents Models” on page 3-2
- “Use Attributes to Route Entities” on page 3-19
- “Role of Gates in SimEvents Models” on page 3-20

Combine Entity Paths

In this section...

Using Entity Input Switch to Combine Paths

You can merge multiple paths into a single path using the Entity Input Switch block with the **Active port selection** parameter set to `All`. Merging entity paths does not change the entities themselves, just as merging lanes on a road does not change the vehicles that travel on it. In particular, the Entity Input Switch block does not create aggregates or batches.

Here are some scenarios in which you might combine entity paths:

- Attaching different data — Multiple entity generator blocks create entities having different values for a particular attribute. The entities then follow a merged path but might be treated differently later based on their individual attribute values.
- Merging queues — Multiple queues merge into a single queue.
- Connecting a feedback path — A feedback path enters the same queue as an ordinary path.

Sequence Simultaneous Pending Arrivals

The Entity Input Switch block does not experience any collisions, even if multiple entities attempt to arrive at the same time. The categories of behavior are as follows:

- If the entity output port is not blocked when the entities attempt to arrive, then the sequence of arrivals depends on the sequence of departure events from blocks that precede the Entity Input Switch block.

  Even if the departure time is the same for multiple entities, the sequence might affect the system's behavior. For example, if the entities advance to a queue, the departure sequence determines their positions in the queue.
- If pending entities are waiting to advance to the Entity Input Switch block when its entity output port changes from blocked to unblocked, then the entity input ports are

notified of the change sequentially. The change from blocked to unblocked means that an entity can advance to the Entity Input Switch block.

If at least two entities are waiting to advance to the Entity Input Switch block via distinct entity input ports, then the notification sequence is important because the first port to be notified of the change is the first to advance an entity to the Entity Input Switch block.

See Also

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

Related Examples

- “Select Departure Path Using Entity Output Switch” on page 3-5
- “Select Arrival Path Using Entity Input Switch” on page 3-8
- “Use Messages To Route Entities” on page 3-12
- “Enable a Gate for a Time Interval” on page 3-22

More About

- “Role of Paths in SimEvents Models” on page 3-2
- “Use Attributes to Route Entities” on page 3-19
- “Role of Gates in SimEvents Models” on page 3-20

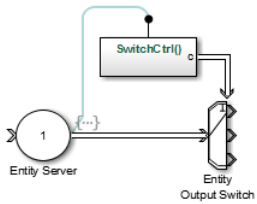
Use Messages To Route Entities

In this section...

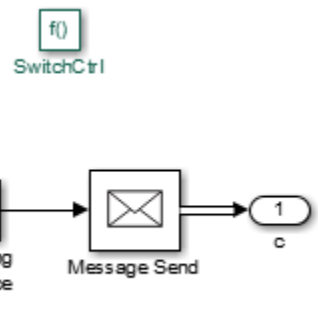
Control Output Switch with a Message

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

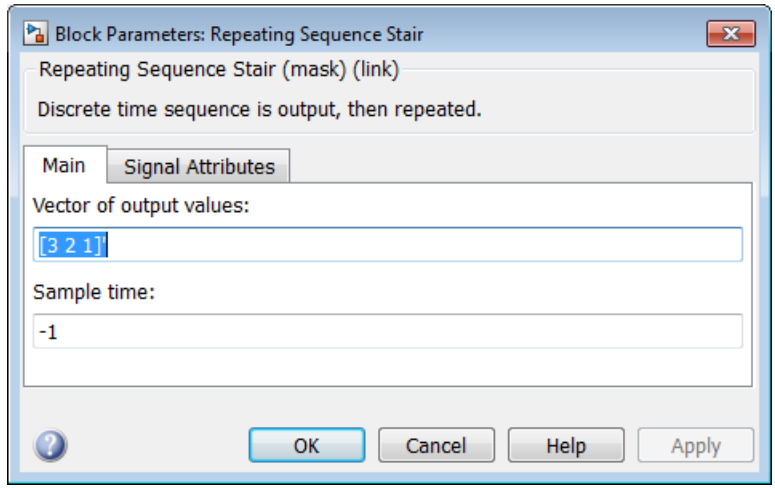
Consider the following example.



The SwitchCtrl function contains a single Repeating Sequence Stair block, whose **Sample time** parameter is set to `-1` (inherited).



When the Simulink Function block executes, it outputs the next number from a repeating sequence. In this model, the output message value is 3, 2 or 1, based on the sequence of values specified in the Repeating Sequence Stair block.



When service in the Entity Server block is complete, the entity advances to the Entity Output Switch block. 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. Use this procedure:

1. Select **From control port**.
2. Set **From control port** to the desired initial port selection. 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

- “Select Departure Path Using Entity Output Switch” on page 3-5

- “Select Arrival Path Using Entity Input Switch” on page 3-8
- “Combine Entity Paths” on page 3-10
- “Enable a Gate for a Time Interval” on page 3-22

More About

- “Role of Paths in SimEvents Models” on page 3-2
- “Use Attributes to Route Entities” on page 3-19
- “Role of Gates in SimEvents Models” on page 3-20

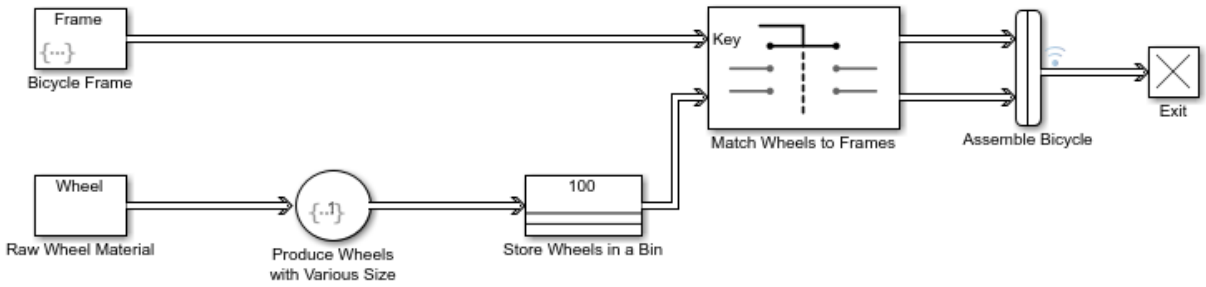
Match Entities Based on Attributes in SimEvents Models

This example shows how to build a model to store and match entities representing bicycle components based on their attributes. 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.

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 in the facility. The frames are ordered from a supplier and they arrive ready to assemble. Due to the supplier, frame arrival rate is slower than the wheel production rate and the set of wheels are stored in a bin.

Bicycle Assembly Line

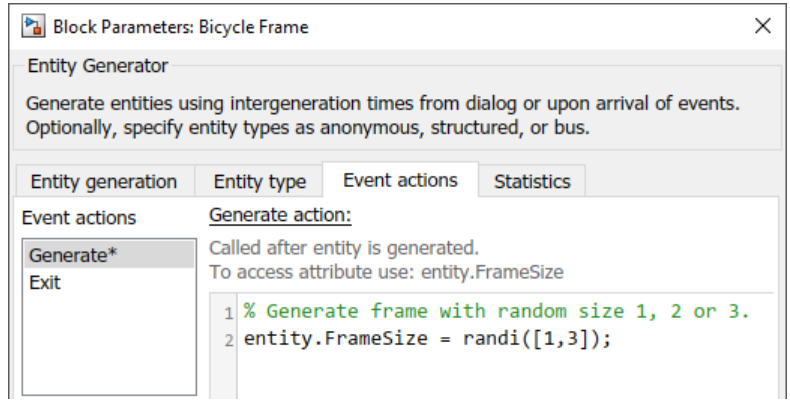
Open the model to investigate the bicycle assembly line that mathces set of wheels to the corresponding frame for assembly.



Producing Bicycle Frames and Wheels

1. Add two Entity Generator blocks, an Entity Server block, an Entity Store block, an Entity Selector block, a Composite Entity Creator block, and an Entity Terminator block to your model. Rename and connect blocks as shown.
2. In the Bicycle Frame Block Parameters dialog box, in the **Entity type name** box, enter `Frame`. In the **Attribute Name** box, enter `FrameSize`, and in the **Attribute Initial Value** box, enter `0`.
3. Select the **Event actions** tab and then select the **Generate** option in the **Event actions** box. In the **Generate action** box, enter the code.

```
% Generate frame with random size 1, 2 or 3.
entity.FrameSize = randi([1,3]);
```



4 Select the **Entity Generation** tab. Set the **Period** to 5.

Set the period to a value that is greater than 1 to represent the slow arrival rate of bicycle frames.

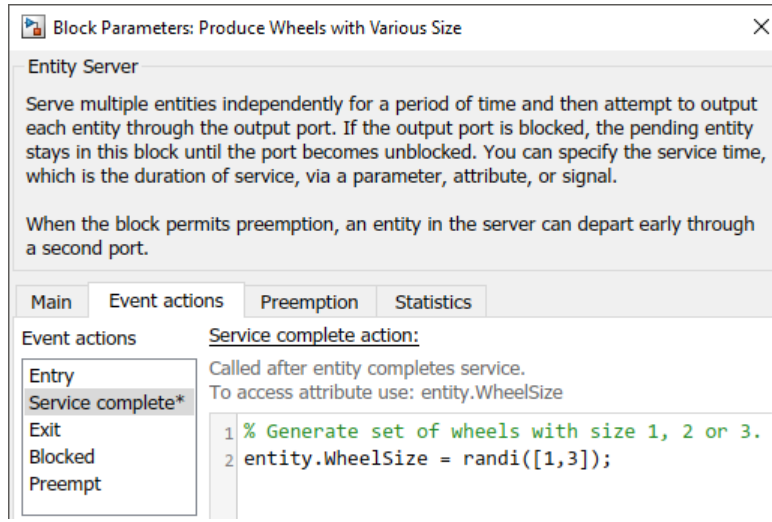
5 In the Raw Wheel Material Block Parameters dialog box, select the **Entity type** tab. In the **Entity type name** box, enter `Wheel`. In the **Attribute Name** box, enter `WheelSize` and in the **Attribute Initial Value** box, enter `0`.

Each entity departing the Raw Wheel Material block represents raw material for producing a set of bicycle wheels.

6 In the Produce Wheels with Various Size Block Parameters dialog box, select the **Event actions** tab and then select the **Service complete** option.

In the **Service complete action** box, enter the code.

```
% Generate set of wheels with size 1, 2 or 3.
entity.WheelSize = randi([1,3]);
```



Each entity represents one set of wheels to be matched to a frame.

Store and Match Wheels to Frames for Bicycle Assembly

1. In the Store Wheels in a Bin Block Parameters dialog box, set the **Capacity** to `100`.
2. Observe that in the Match Wheels to Frames Block Parameters dialog box, the **Number of matching streams** is 1. There is one incoming stream of set of wheels to be matched to a frame.
3. Set the **Key entity attribute name** to `{'FrameSize'}`.

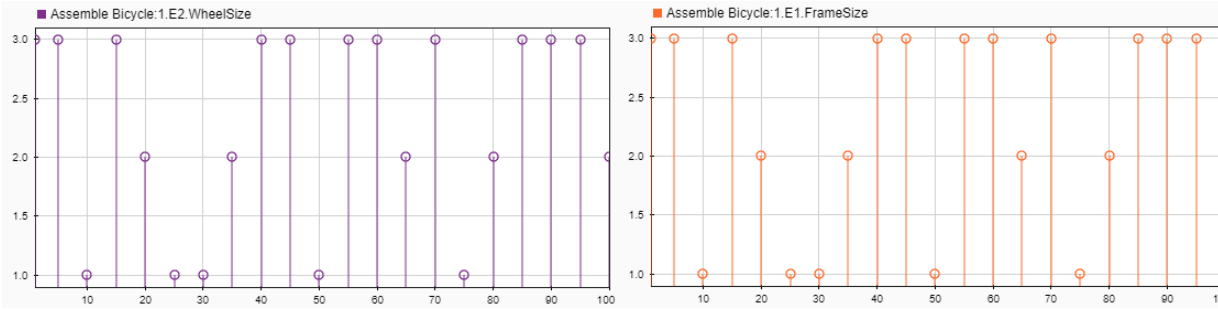
   Frame is the reference entity and its size is the reference value to be matched.
4. Set the **Matching entity attribute name(s)** to `{'WheelSize'}`.

   Entity Selector block matches a set of wheels with size equal to the reference frame size.
5. In the Assemble Bicycle Block Parameters dialog box, set the **Entity type name** to `Bicycle`.

Results

1. Set the **Simulation stop time** to 100.

2. Select the entity path originating from Bicycle Assembly block and right-click to select **Log Selected Signals**.
3. Simulate the model. Open **Simulation Data Inspector**. For more information about using the Simulation Data Inspector, see “Inspect Simulation Data” (Simulink).
4. Observe that for the bicycle assembly, the size of the set of wheels and the frames are matched. The Entity Selector block matches the size of the wheels stored in the bin to the frames.



See Also

Composite Entity Creator | Entity Selector | Entity Store | Entity Gate | Entity Input Switch | Entity Output Switch | Entity Server

Related Examples

- “Use Messages To Route Entities” on page 3-12
- “Enable a Gate for a Time Interval” on page 3-22

More About

- “Role of Paths in SimEvents Models” on page 3-2
- “Role of Gates in SimEvents Models” on page 3-20

Use Attributes to Route Entities

Suppose entities represent manufactured items that undergo a quality control process followed by 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 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.

See Also

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

Related Examples

- “Use Messages To Route Entities” on page 3-12
- “Enable a Gate for a Time Interval” on page 3-22

More About

- “Role of Paths in SimEvents Models” on page 3-2
- “Role of Gates in SimEvents Models” on page 3-20

Role of Gates in SimEvents Models

In this section...

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 signal to determine time intervals over which the gate is open or closed
- The release gate, which uses a control message 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

- “Select Departure Path Using Entity Output Switch” on page 3-5
- “Select Arrival Path Using Entity Input Switch” on page 3-8
- “Combine Entity Paths” on page 3-10
- “Use Messages To Route Entities” on page 3-12
- “Enable a Gate for a Time Interval” on page 3-22

More About

- “Role of Paths in SimEvents Models” on page 3-2
- “Use Attributes to Route Entities” on page 3-19

Enable a Gate for a Time Interval

In this section...

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:

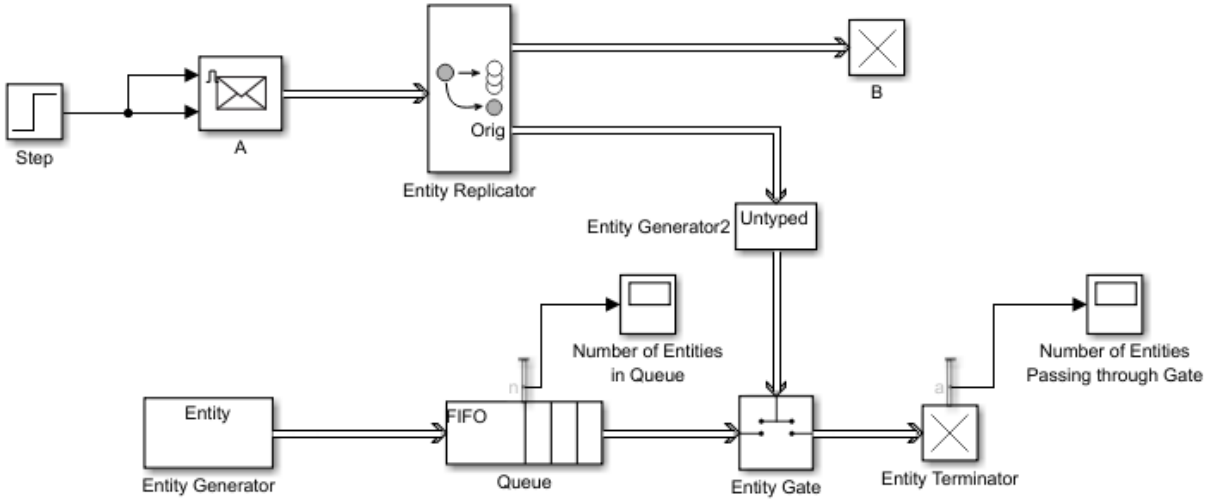
- When a message with a positive payload arrives at the enable port at the top of the block, the gate is open and an entity can arrive as long as it would be able to advance immediately to the next block.
- When a message with zero or negative payload arrives at the enable port at the top of the block, 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.

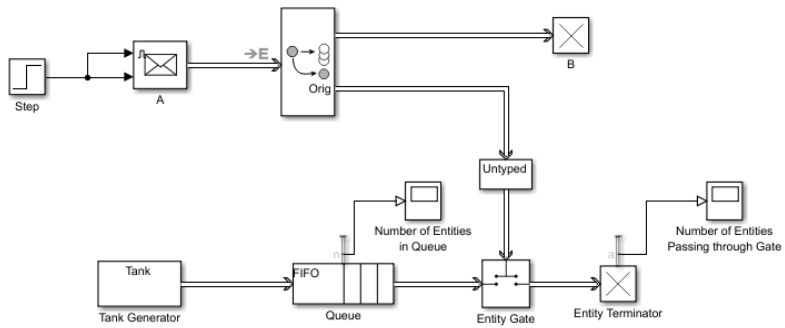
Depending on your application, the gating logic can arise from time-driven dynamics, state-driven dynamics, a SimEvents block's statistical output signal, or a computation involving various types of signals.

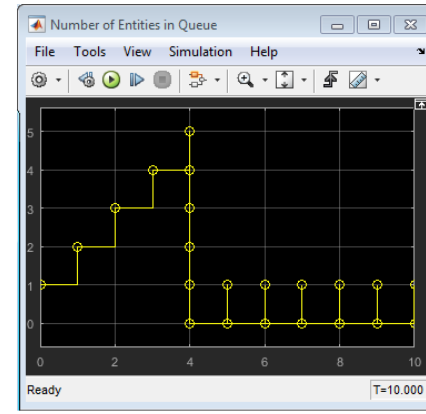
Sense an Entity Passing from A to B and Open a Gate

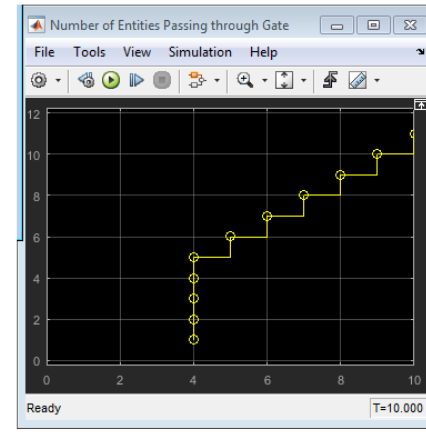
This example shows how to use the `Sense an Entity Passing from A to B and Open a Gate` design pattern. In this example, the Step block generates a step signal at time 4. This signal passes through the Message Send block A. The Entity Replicator block duplicates the entity and passes it to B. It uses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

- “Select Departure Path Using Entity Output Switch” on page 3-5
- “Select Arrival Path Using Entity Input Switch” on page 3-8
- “Combine Entity Paths” on page 3-10
- “Use Messages To Route Entities” on page 3-12

More About

- “Role of Paths in SimEvents Models” on page 3-2
- “Use Attributes to Route Entities” on page 3-19
- “Role of Gates in SimEvents Models” on page 3-20

Work with Resources

Model with Resources

In this section...

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

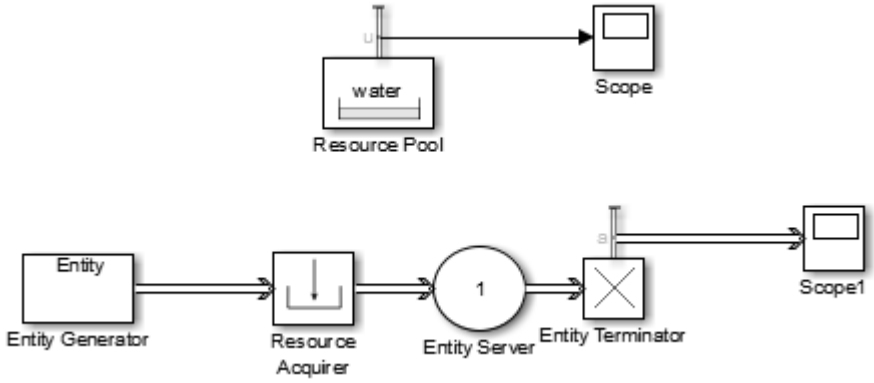
Use the **Selected Resources** table of the Resource Acquirer block to receive the resource amount definition from the block dialog box or an entity attribute. Using attributes as the source for the resource requires synchronicity between these blocks:

- 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

Find and Extract Entities in SimEvents Models

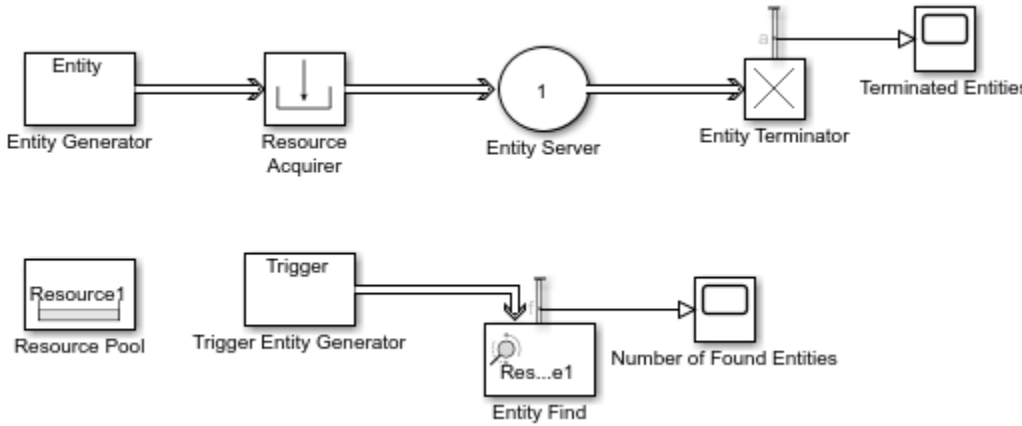
You can find entities in a SimEvents model by using an Entity Find block. The block searches and finds entities that use a particular resource from a Resource Pool block and acquire it through a Resource Acquirer block.

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.

Find and Examine 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.

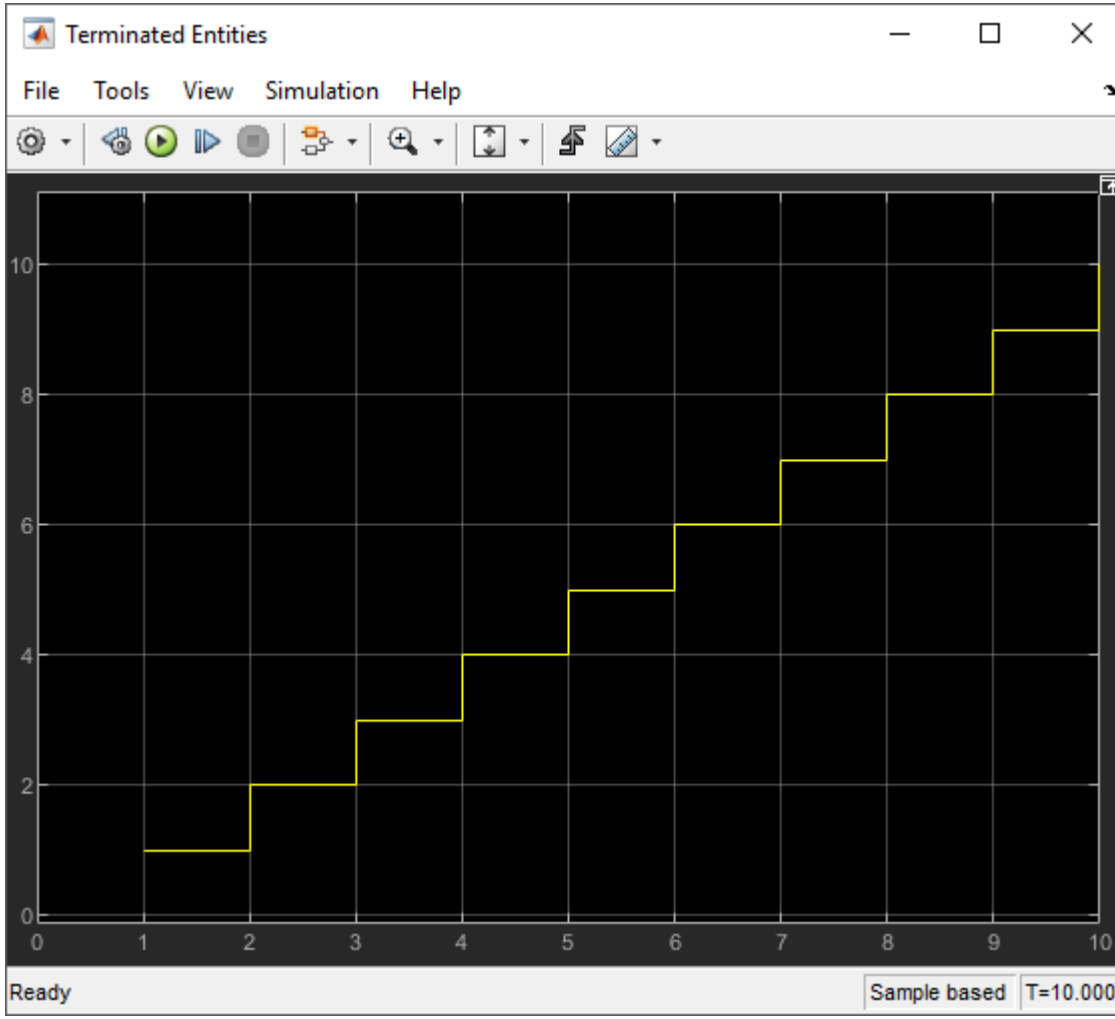
The top model represents the flow of entities that acquires a `Resource1` resource.

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.

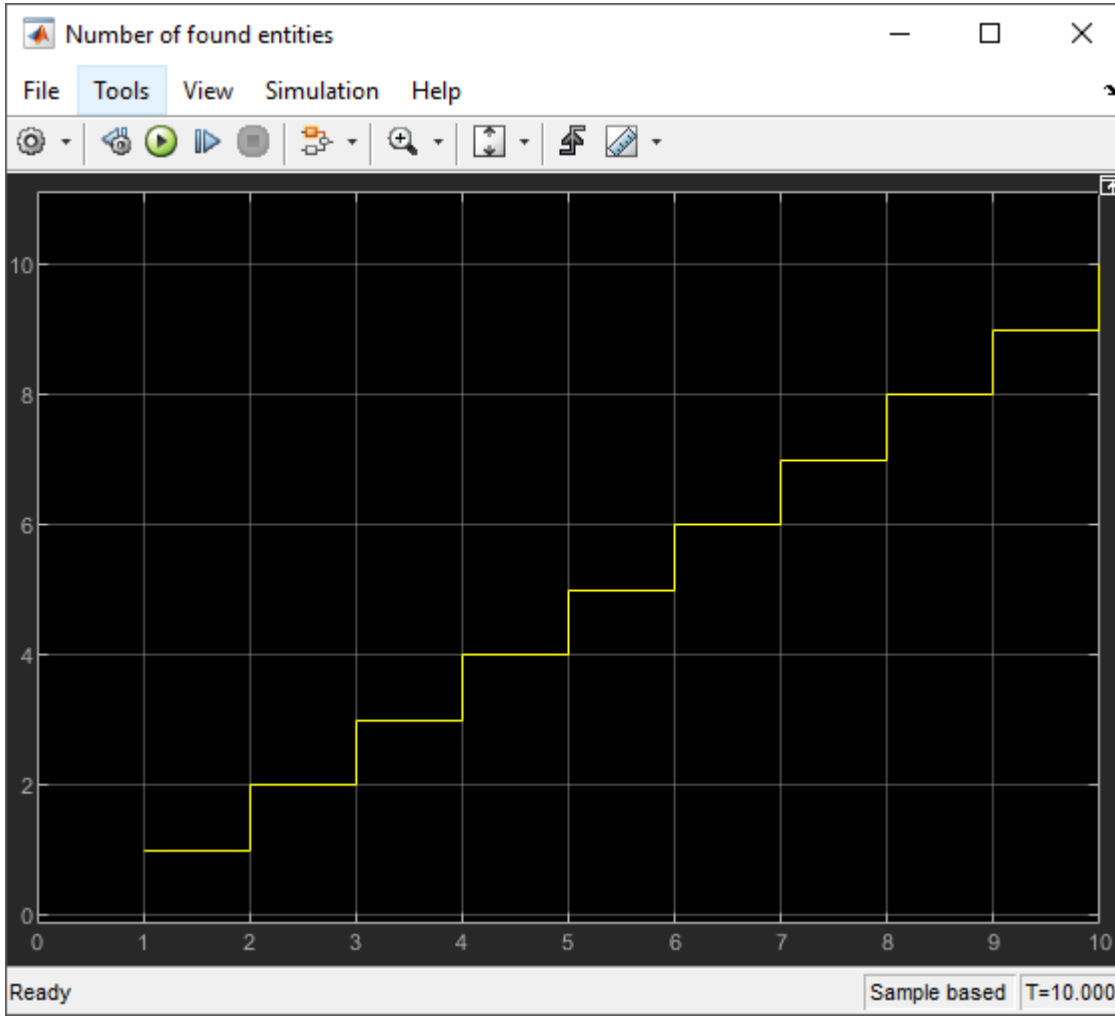
   By default, the block finds entities with the `Resource1` tag.
4. Add another Entity Generator block and label it Trigger Entity Generator. Connect it to the input port of the Entity Find block. In the block, change the **Entity type name** to `Trigger` and **Entity priority** to `100`.

   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.

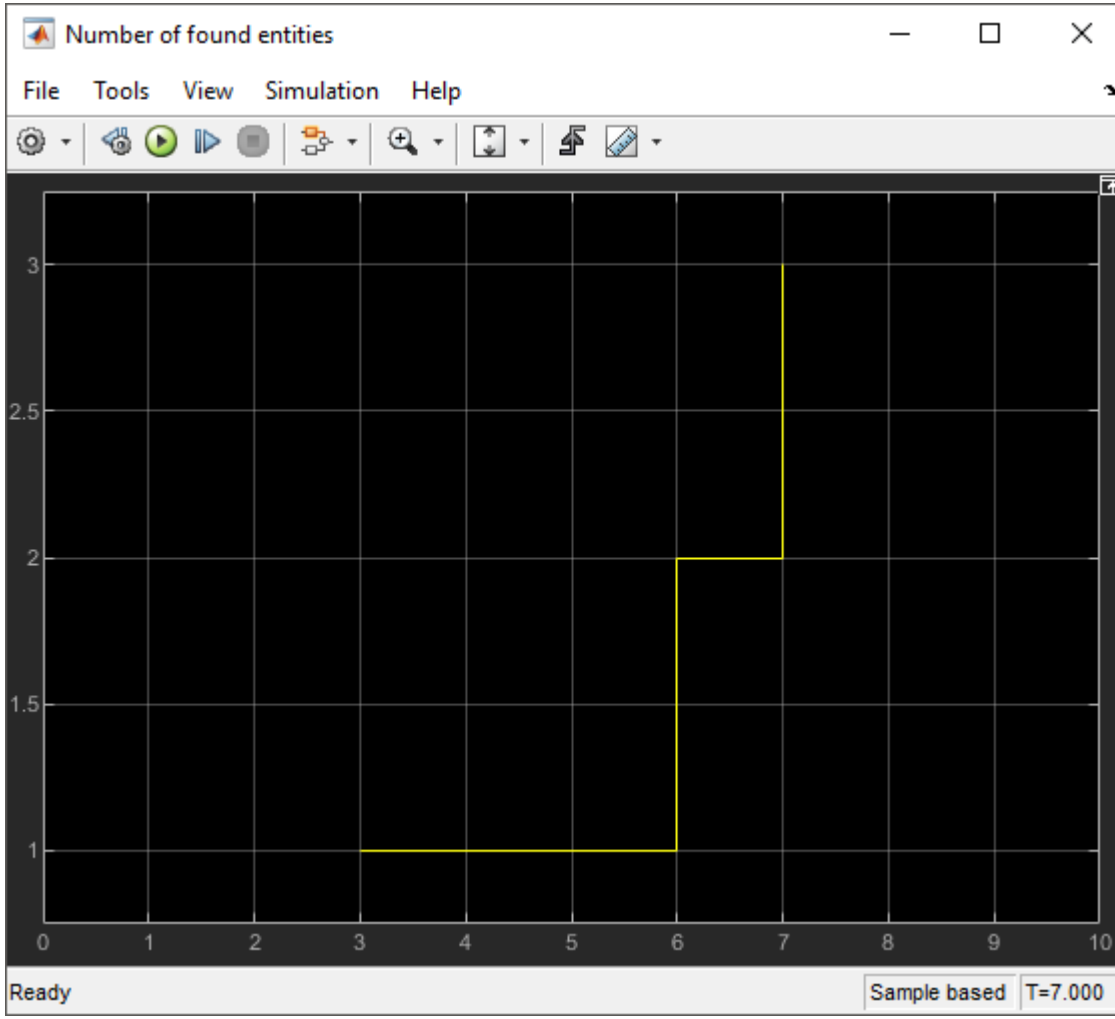
```
match = isequal(trigger.Attribute1, entity.Attribute1);
```

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`.

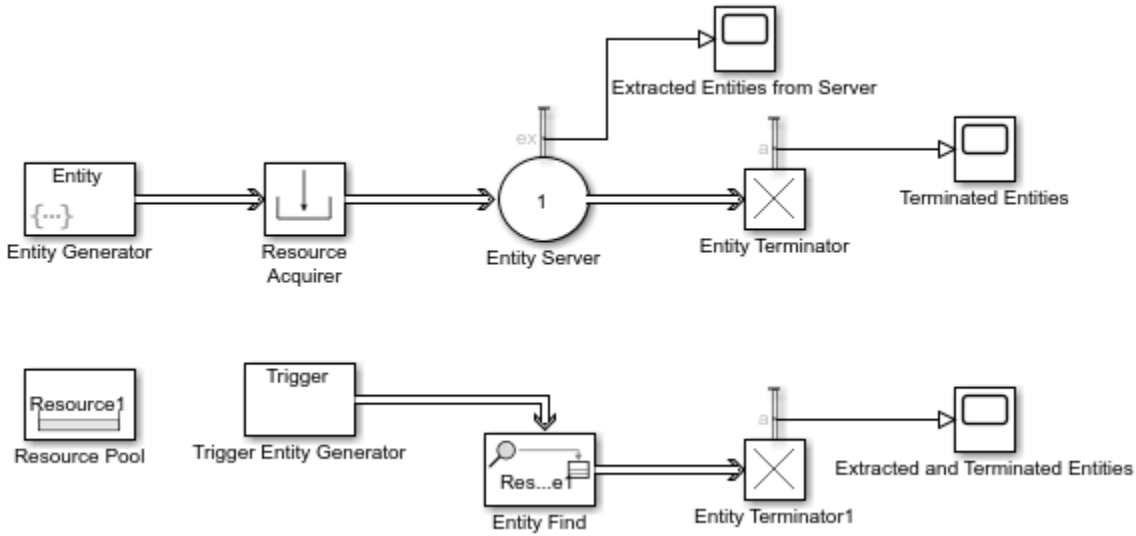
9 Simulate the model, observe that the number of found entities decreased to `3` because entities with the `Attribute1` value `2` are filtered out by the additional matching condition.

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



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.

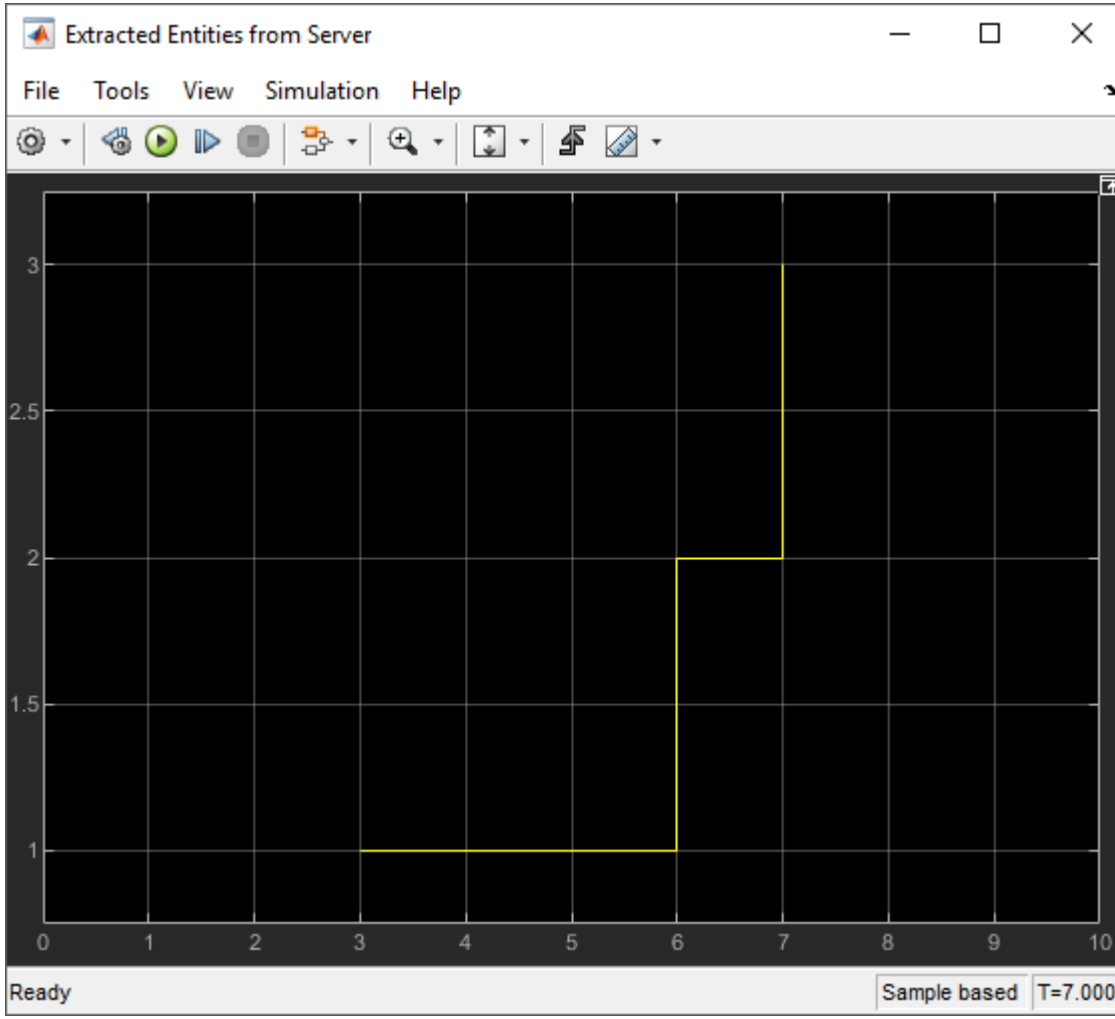


1. In the Entity Find Block Parameters dialog box, select the **Extract found entities** check box.

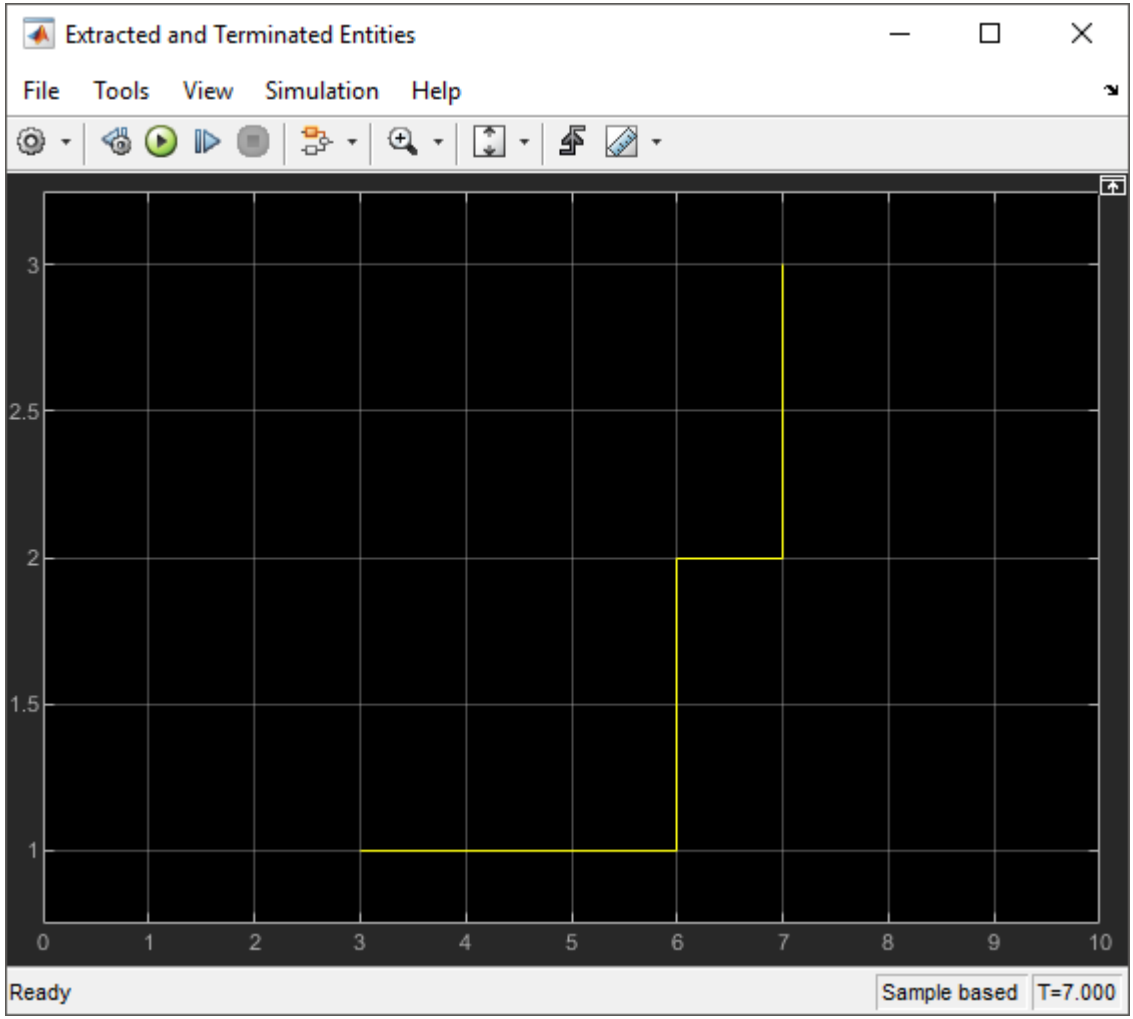
   Observe that a new output port appears at the Entity Find block for the extracted entities.
2. Connect the output of the Entity Find block to a new Entity Terminator1 block.
3. Output the **Number of entities extracted, ex** statistic from the Entity Server block and connect it to a scope.

   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.

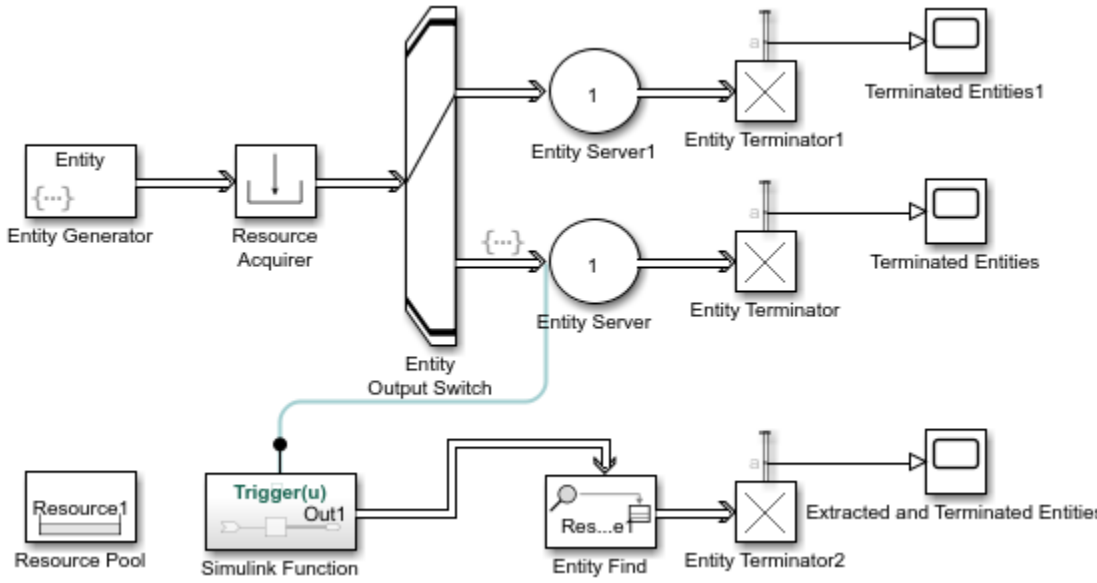
Change Found Entity Attributes

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 1-5.
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.

Trigger Entity Find Block with Event Actions

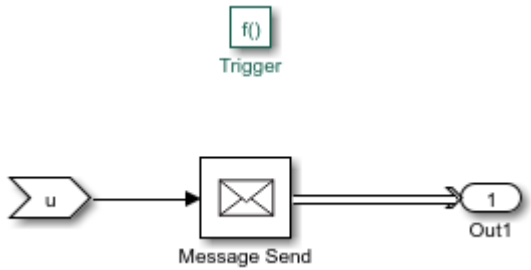
You can trigger the Entity Find block with event actions. In this example, the Entity Find block is triggered when an entity enters the Entity Server block. Modify the previous example by removing the Trigger Entity Generator and by adding the Entity Output Switch, Entity Server1, Entity Terminator2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



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.



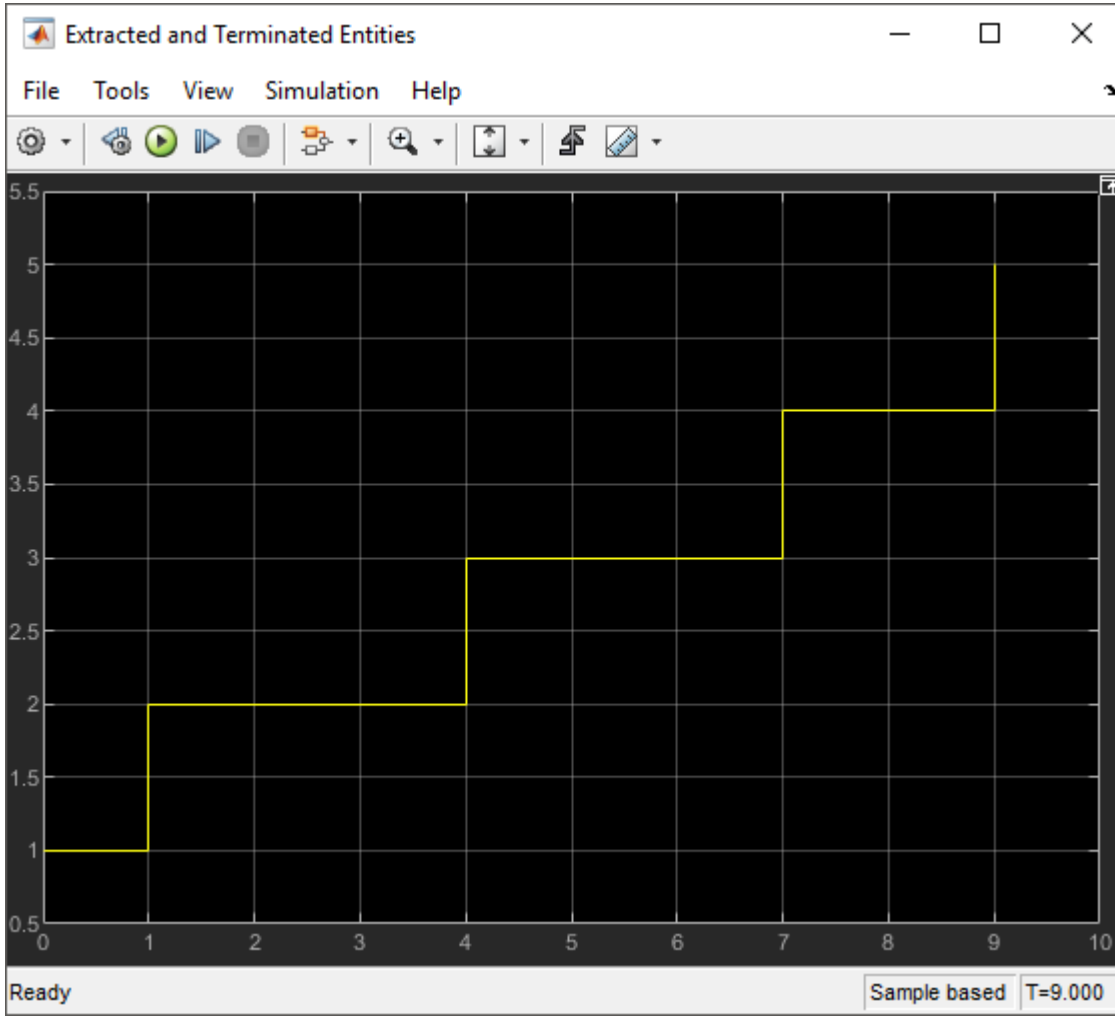
   The `Trigger(u)` function call generates a message to trigger the Entity Find block every time an entity enters the Entity Server1 block.
4. In the Entity Server block, in the **Entry action** field, enter this code.

   ```
   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.

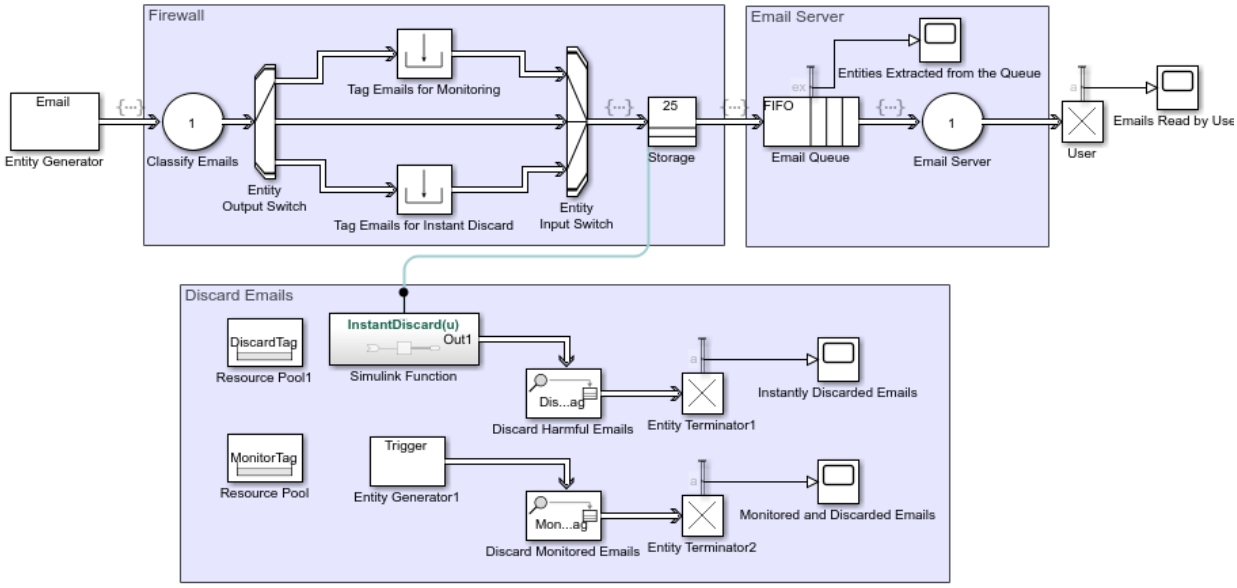


Build Firewall and Email Server

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.



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]);
   ```

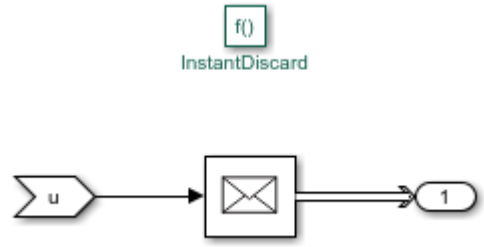
   The `Suspicious` attribute of an email changes in the entry. If the `Suspicious` attribute value is 2, the email is extracted and terminated. This represents the randomly observed suspicious activity in the system.
10. Add another Entity Server block. In the block, set the **Service time value** to 3, 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

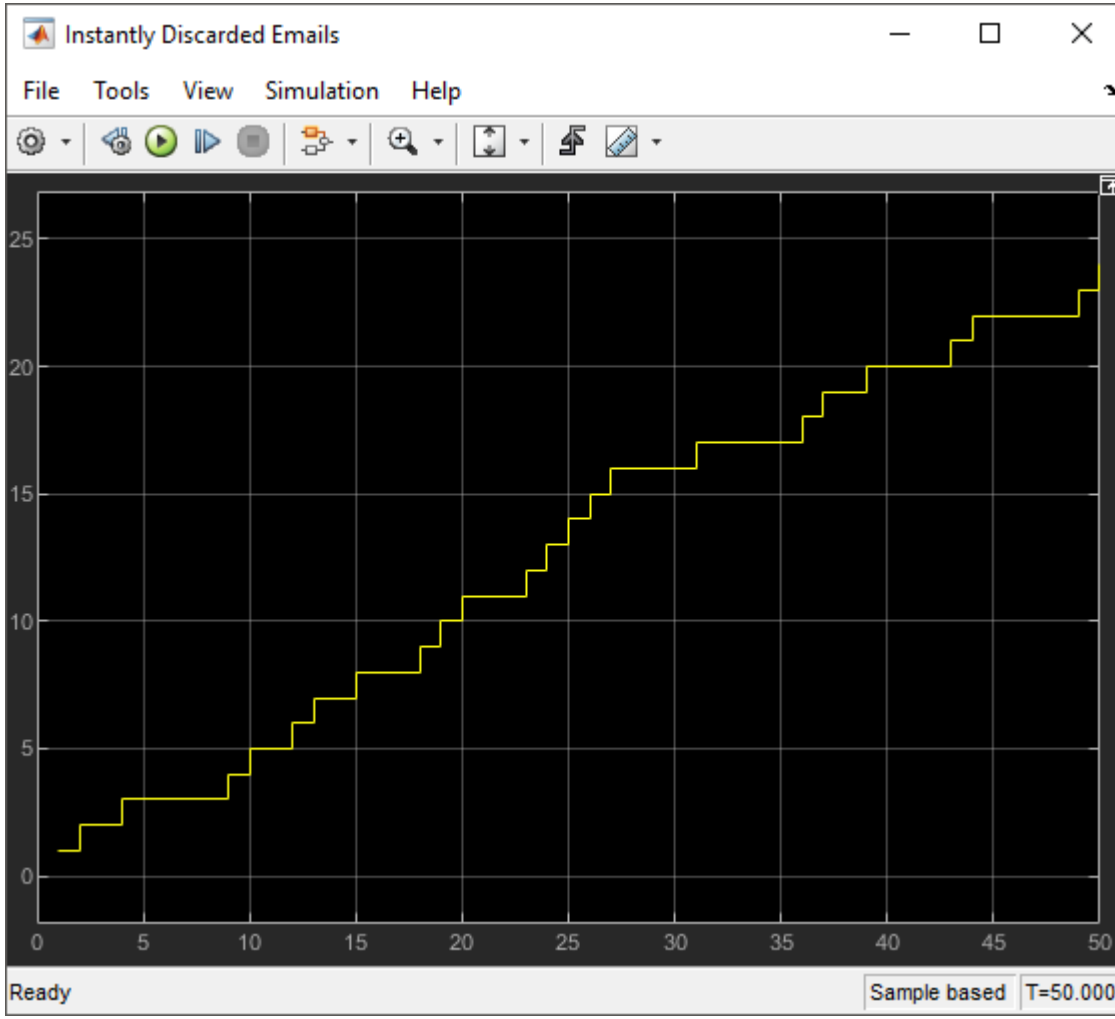
1. Add a Simulink Function block.
   a. Double-click the function signature on the Simulink Function block and enter `InstantDiscard(u)`.
   b. Double-click the Simulink Function block. Add a Message Send block and an Out1 block.

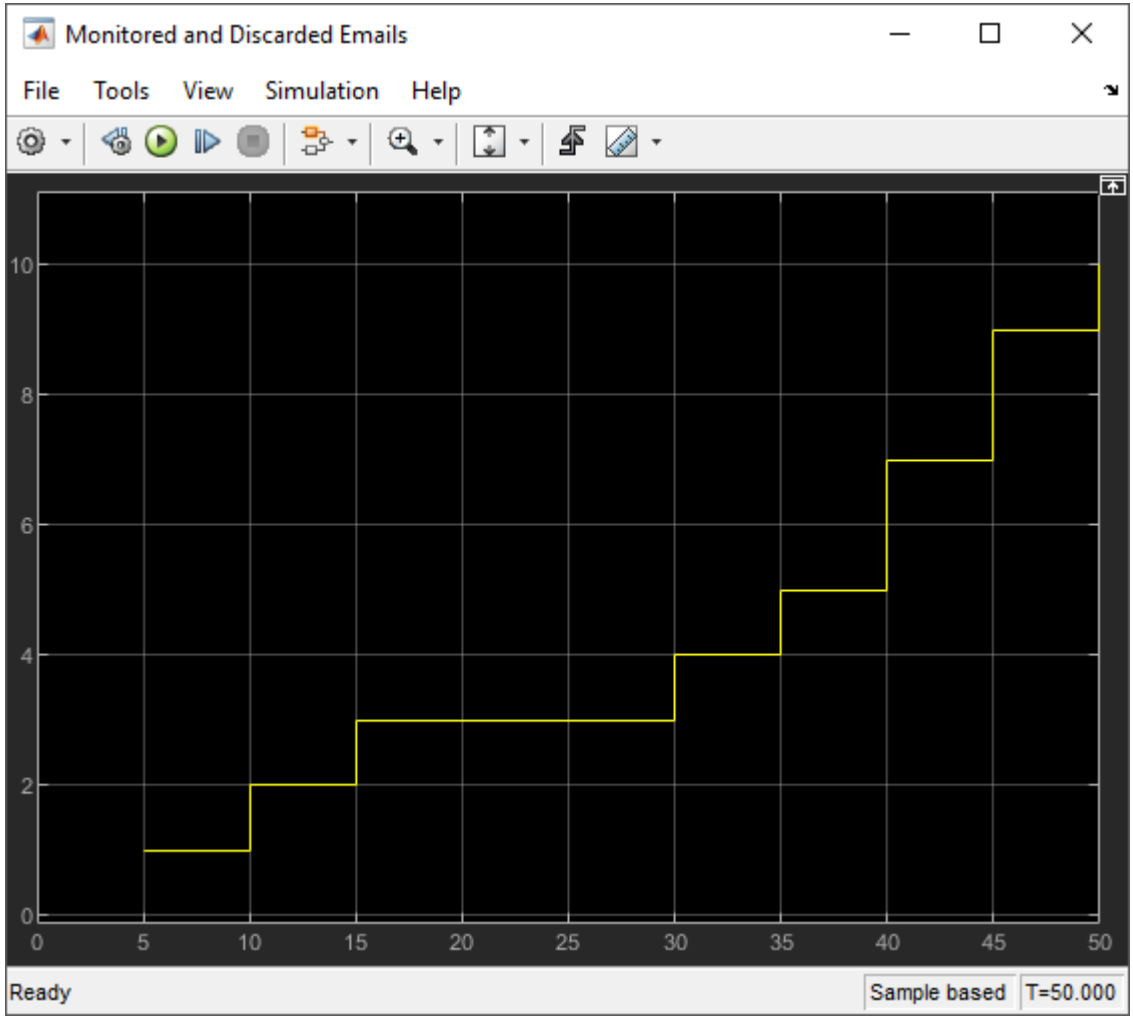


2. In the parent model, add an Entity Find block. In the block, set **Resource** to `DiscardTag` and select **Extract found entities** check box.

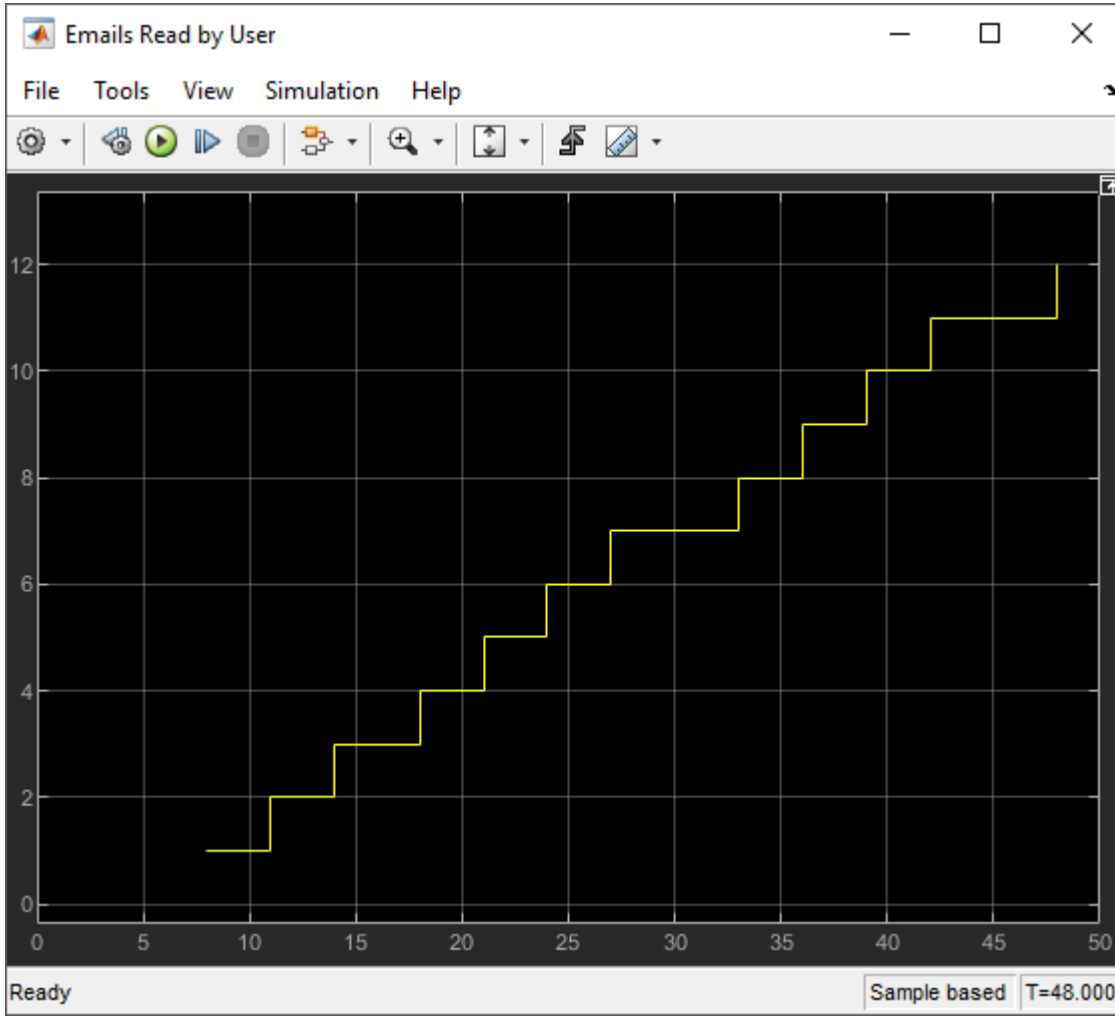
   Any email entry calls the `InstantDiscard()` function and triggers the Entity Find block to find and discard harmful emails.
3. Add another Entity Terminator block labeled Instantly Discarded Emails.
4. Add another Entity Find block. In the block, set the **Resource** to `MonitorTag` and select the **Extract 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.

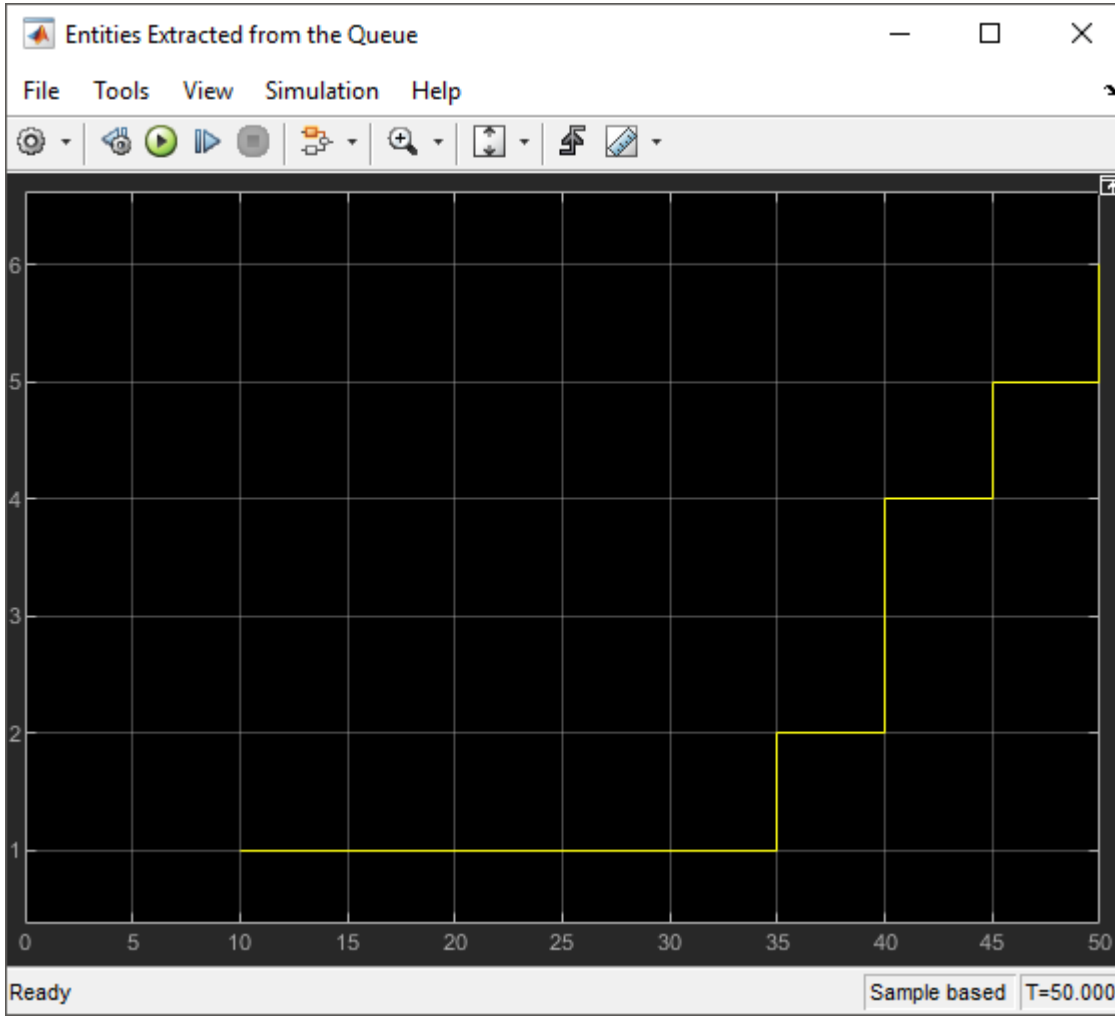

Instantly Discarded Emails
File
Tools
View
Simulation
Help
25
20
15
10
5
0
0
5
10
15
20
25
30
35
40
45
50
Ready
Sample based
T=50.000



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 5-15

More About

- “Model with Resources” on page 4-2
- “Set Resource Amount with Attributes” on page 4-4

5

Visualization, Statistics, and Animation

Interpret SimEvents Models Using Statistical Analysis

In this section...

Statistical reporting is an important part of SimEvents blocks. Choosing the right statistical measure is critical for evaluating the model performance. 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.

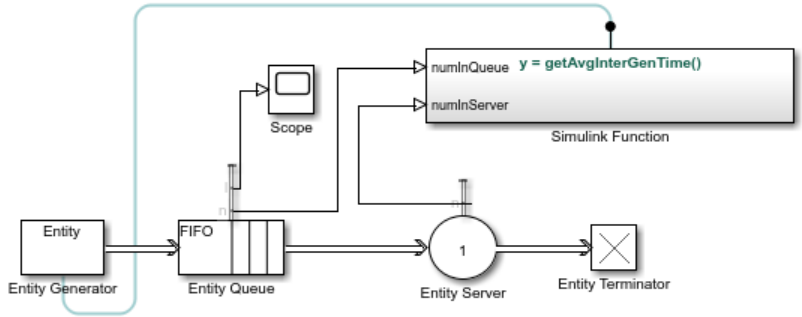
- Identify the appropriate size of the samples to compute more meaningful statistics.
- Decide if you want to investigate the transient behavior, the steady-state behavior, or both.
- Specify the number of simulations that ensures sufficient confidence in the results.

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 1-25.



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			✓	✓				✓	

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
Entity Batch Splitter			✓	✓				✓	
Entity Find	✓	✓		✓	✓				
Entity Generator				✓				✓	
Entity Queue	✓	✓		✓	✓	✓			
Entity Selector				✓	✓	✓			
Entity Server		✓		✓	✓	✓	✓	✓	✓
Entity Store	✓	✓		✓	✓	✓			
Entity Terminator			✓						
Multicast Receive Queue	✓	✓		✓	✓	✓			

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
Resource Acquirer		✓		✓	✓	✓			
Resource Pool									✓

The statistical parameters are updated on particular events during the simulation. 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					✓

Statistics Port	Updated on Event				
	Entry	Exit	Blocked	Preempted	Extracted
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 all time-weighted queue and averaging over total time t_f.

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

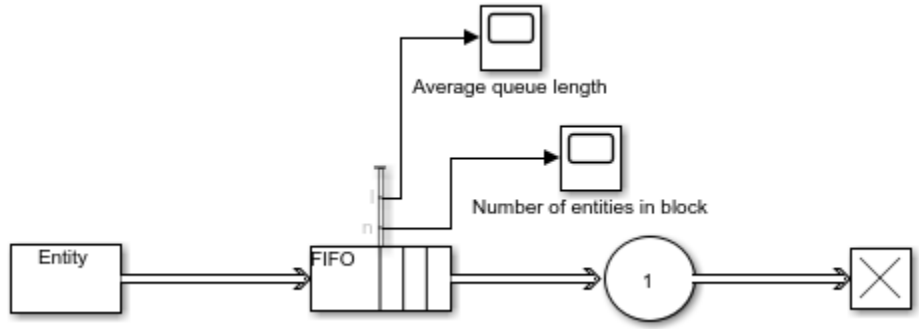
Where:

- t is the time between the entity arrival and / or departure events and f is the total number of such 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.

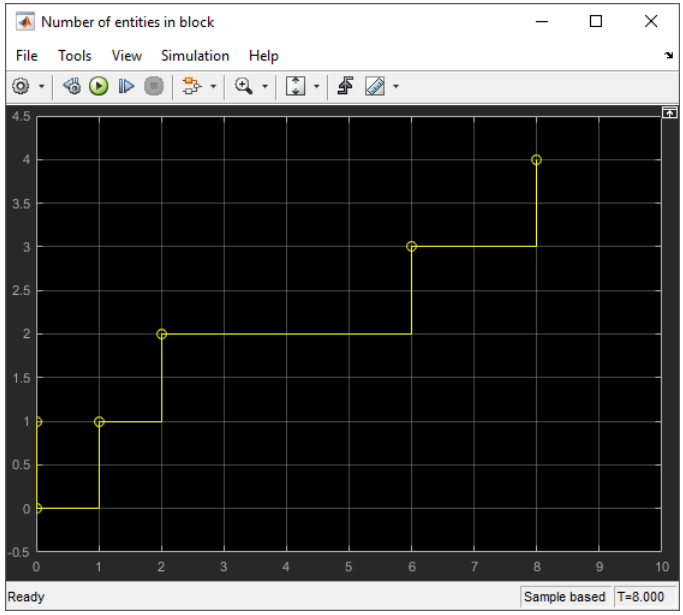
Average queue length example in the Entity Queue block

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.



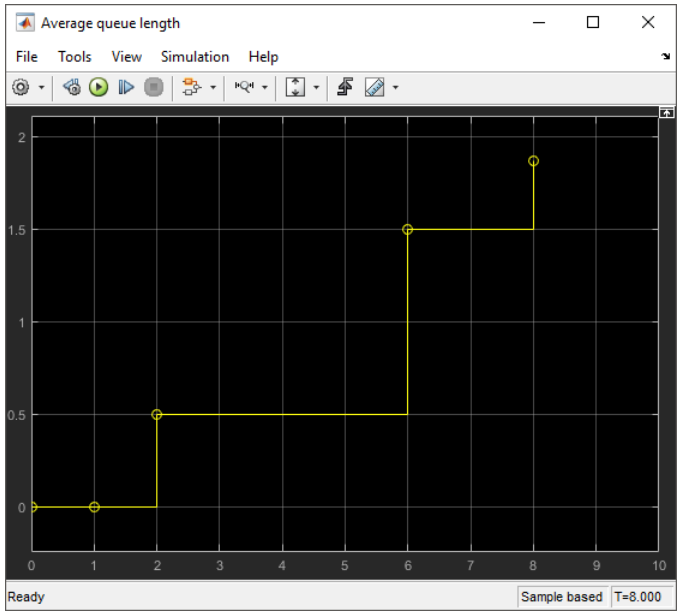
For the duration between `0` and `1`, the average queue length is `0` because the size of the queue is `0`. Between `1` and `2` the queue length is `1`. Average queue length at time $t_f = 2$ is calculated as follows.

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

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

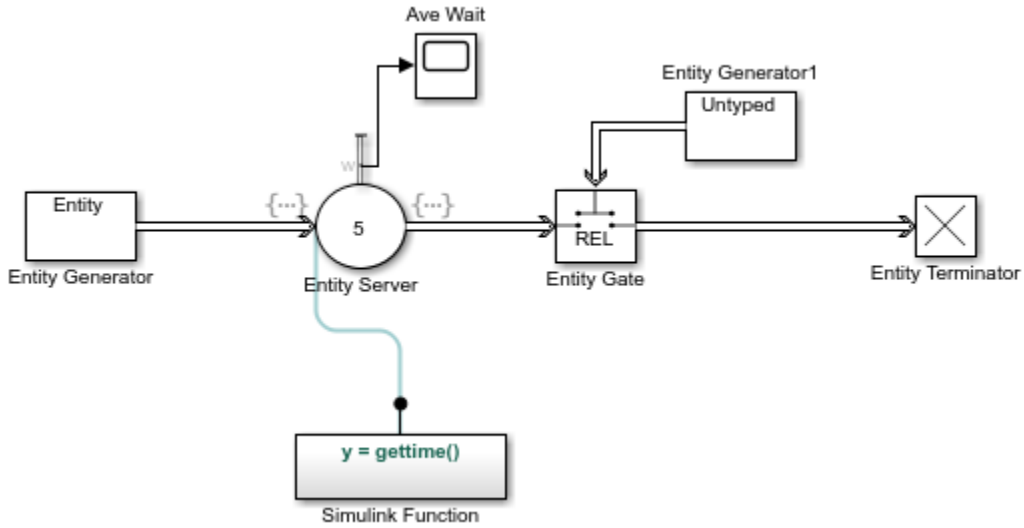
Average wait, w represents the sum of the wait times for entities departing the block divided by their total number, n.

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

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

Average wait of entities example in the Entity Server block

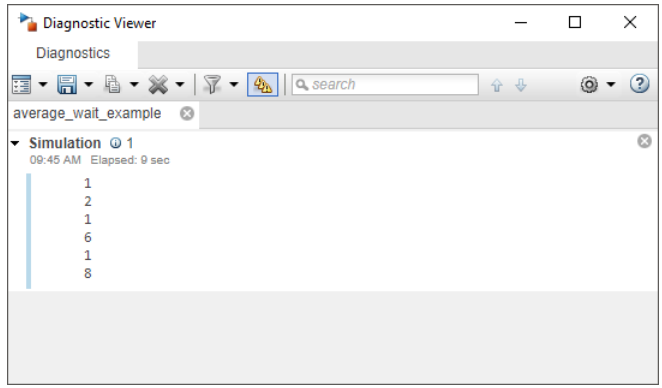
This example shows the average wait time for entities that are served in the Entity Server block.



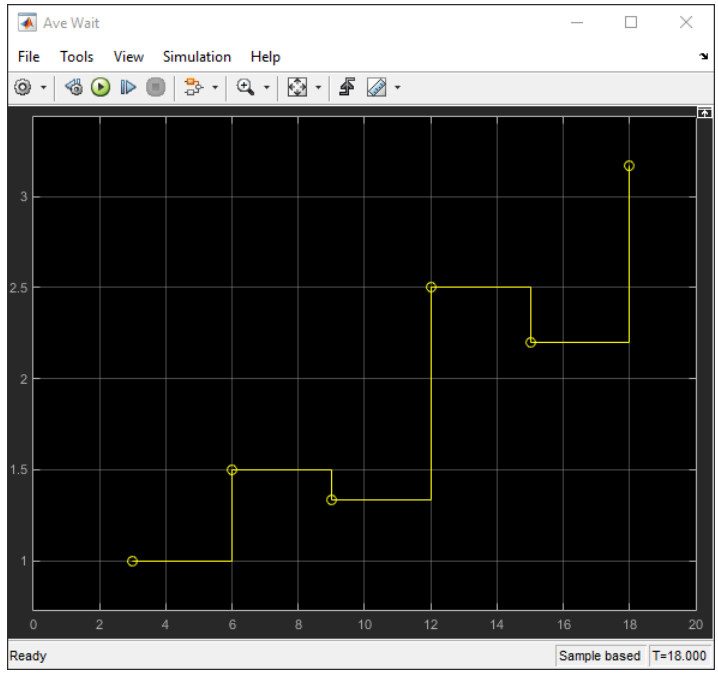
Calculate average wait in the example

The duration for the Entity Server block entry and exit of entity is computed by the `gettime()` function in the Simulink Function block.

The Diagnostic Viewer displays the duration between the Entity Server block 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, 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 which 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” on page 5-14
- “Explore Statistics and Visualize Simulation Results”

Visualization and Animation

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.

See Also

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

Related Examples

- “Visualize and Animate Simulations”

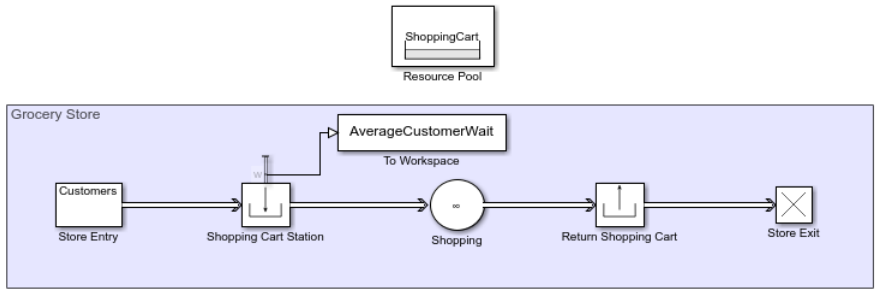
More About

- “Count Entities”

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.

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 Grocery Store 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 Multiple Simulations” (Simulink) and “Run Parallel Simulations” (Simulink).

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.

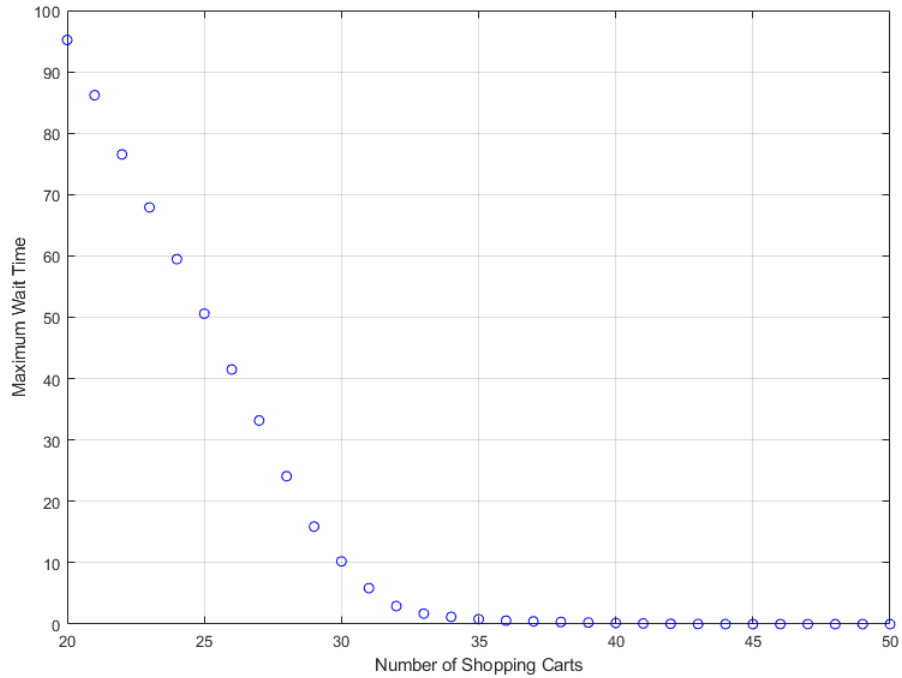
```
% Run NumSims number of simulations
NumCustomer = zeros(1,NumSims);
```

```
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

- “Explore Statistics and Visualize Simulation Results”

More About

- “Interpret SimEvents Models Using Statistical Analysis” on page 5-2

- “Count Entities”
- “Visualization and Animation” on page 5-14
- “Adjust Entity Generation Times Through Feedback” on page 1-25
- “Save SimEvents Simulation State with SimState” on page 6-8

Use the Sequence Viewer Block to Visualize Messages, Events, and Entities

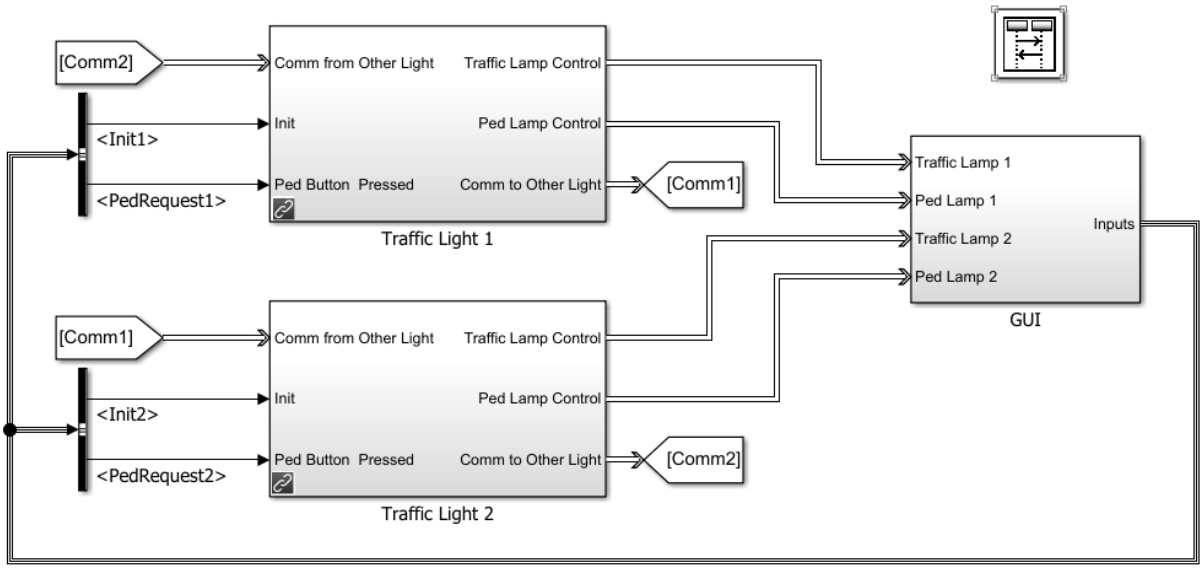
To see the interchange of messages and events between Stateflow charts and the movement of entities between SimEvents blocks, add a Sequence Viewer block to your Simulink model.

In the Sequence Viewer block, 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.

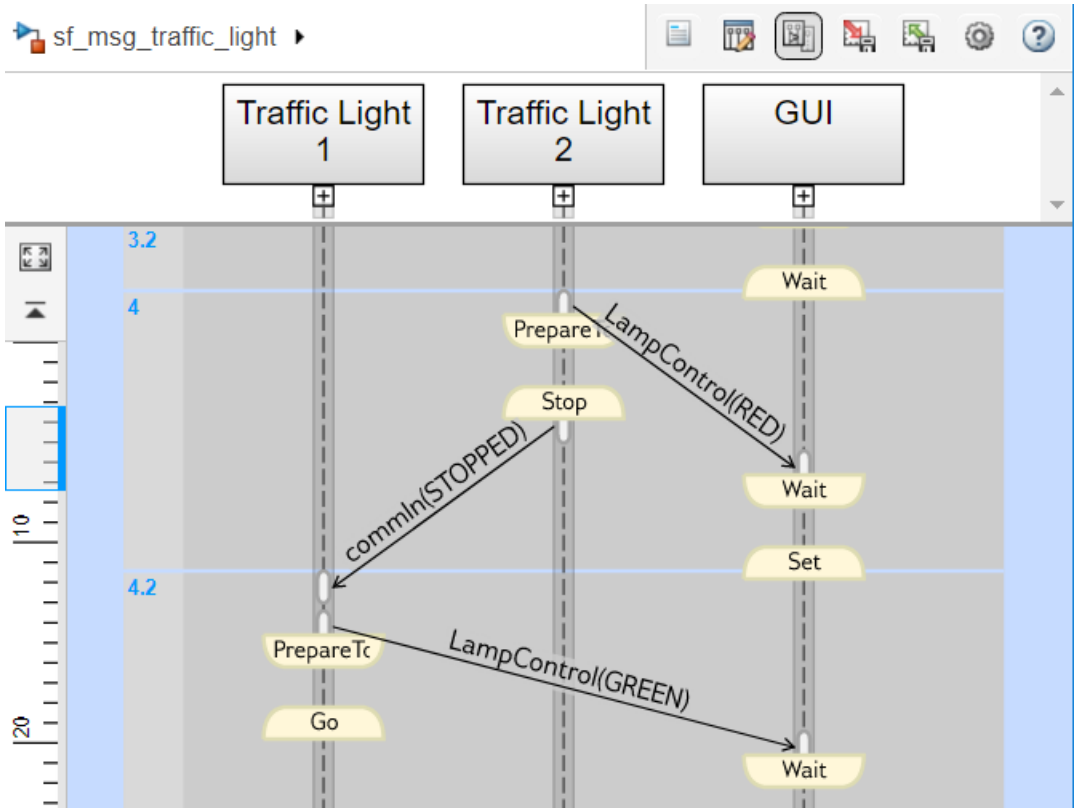
With the Sequence Viewer block, you can visualize the movement of entities between blocks when simulating SimEvents models. All SimEvents blocks that can store entities appear as lifelines in the Sequence Viewer window. Entities moving between these blocks appear as lines with arrows. You can view calls to Simulink Function blocks and to MATLABFunction 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`.



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 block settings.

- Restore Sequence Viewer block settings.
- Configure Sequence Viewer block parameters.
- Access the Sequence Viewer block documentation.

Property Inspector

In the Property Inspector, you can choose filters to show or hide:

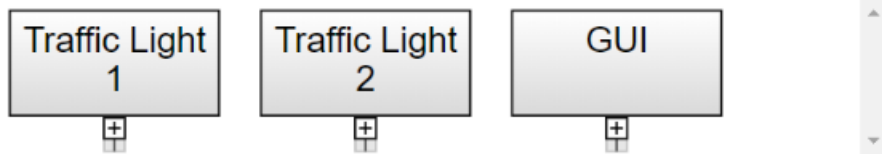
- Events
- Messages
- Function Calls
- State Changes and Transitions

Header Pane

The header pane below the Sequence Viewer toolbar shows lifeline headers containing the names of the corresponding blocks in a model.

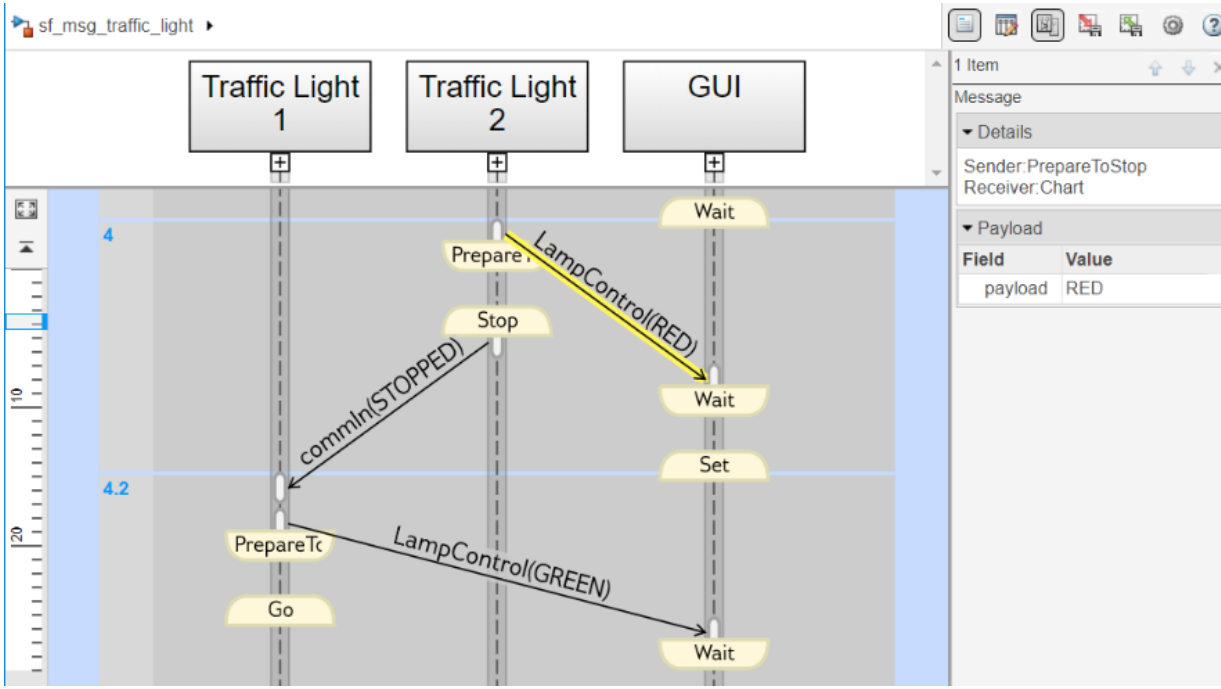
- Gray rectangular headers correspond to subsystems.
- White rectangular headers correspond to masked subsystems.
- Yellow headers with rounded corners correspond to Stateflow charts.

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.



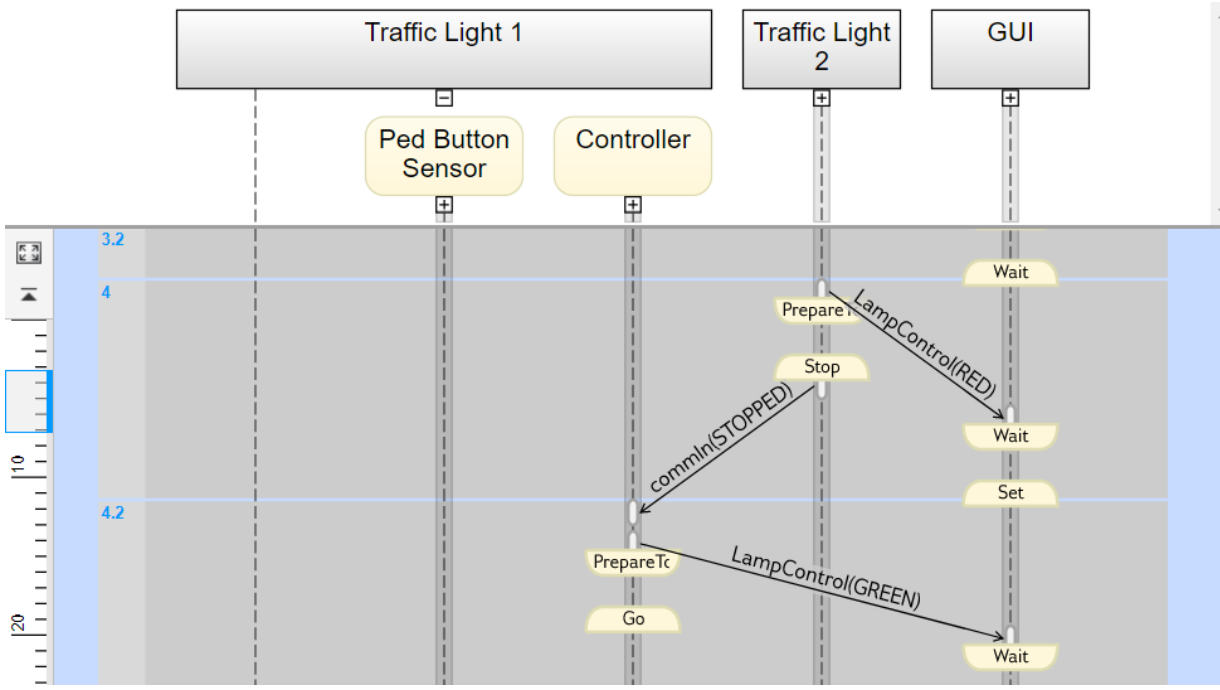
Navigate the Lifeline Hierarchy

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.

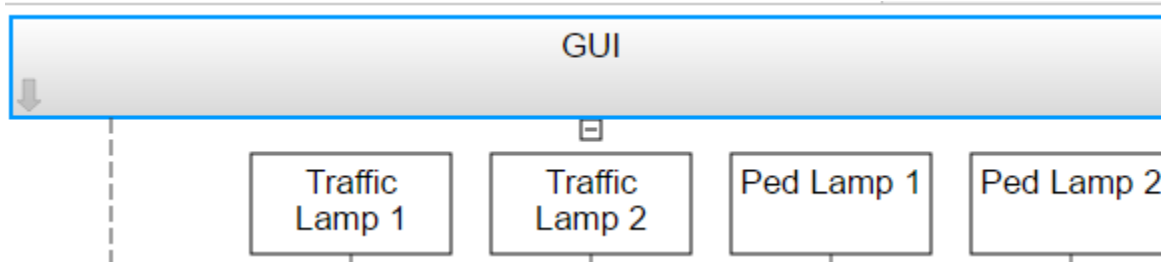
For example, expanding the lifeline for the Traffic Light 1 block reveals two new lifelines corresponding to the Stateflow charts Ped Button Sensor and Controller.



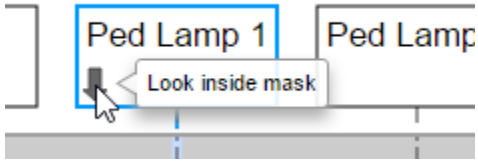
Expand a Masked Subsystem Lifeline

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.

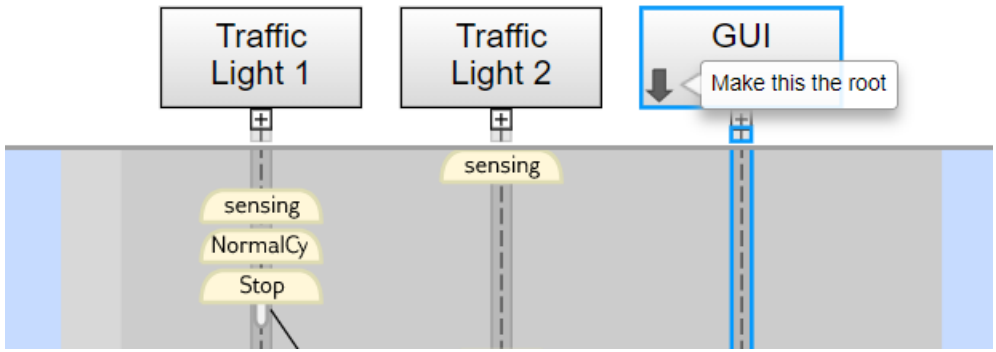


You can display the child lifelines in these masked subsystems by clicking the arrow in the parent lifeline header.

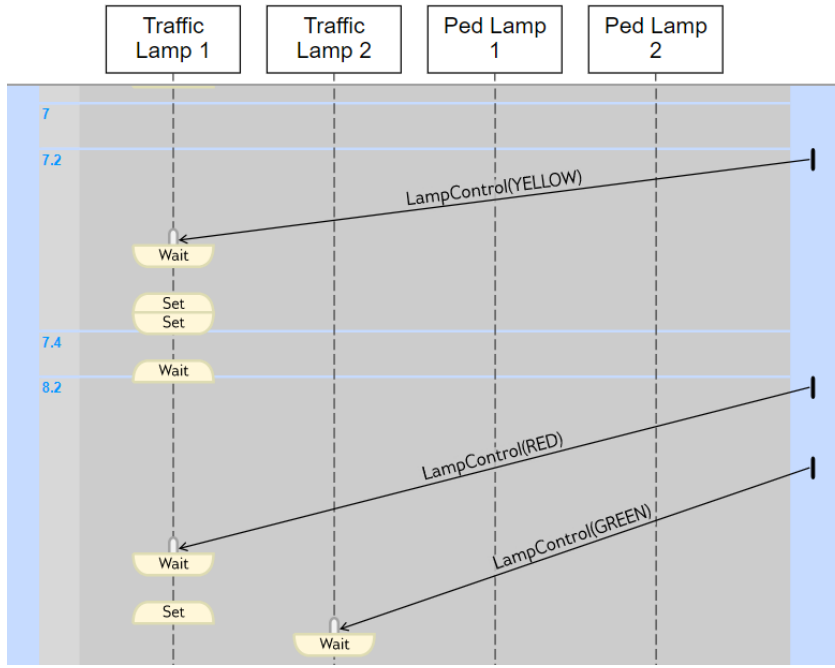


Change Root of Focus

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.



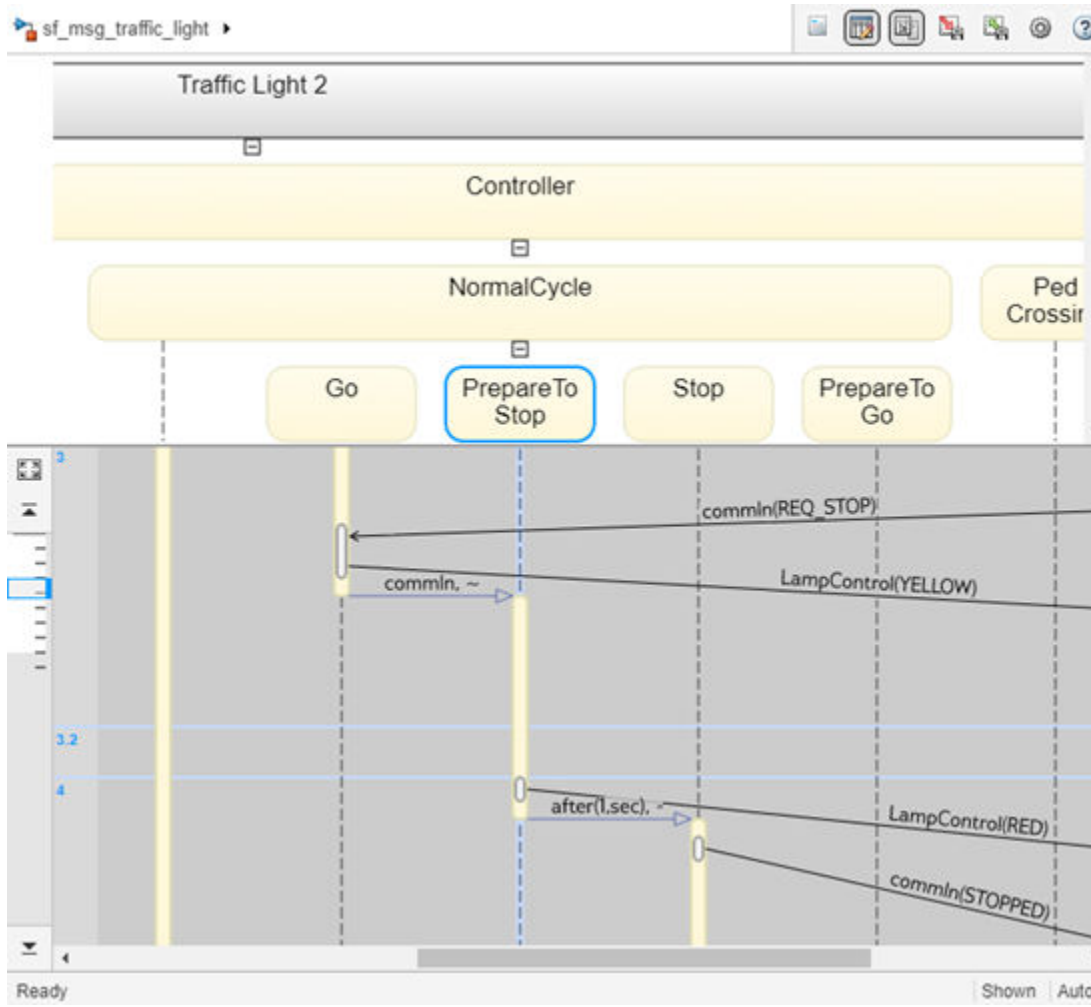
The Sequence Viewer window displays the current root lifeline path and shows its child lifelines. Any external events and messages are displayed as entering or exiting through vertical slots in the diagram gutter. When you point to a slot in the diagram gutter, a tooltip displays the name of the sending 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.

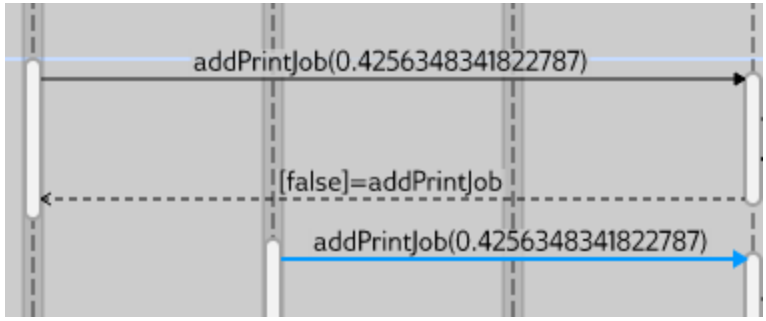
View Function Calls

The Sequence Viewer block 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
Calls to Stateflow graphical or Stateflow MATLAB functions	• Scoped — Select the **Export chart level functions** chart option. Use the *`chartName.functionName`* dot notation. • 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`*.

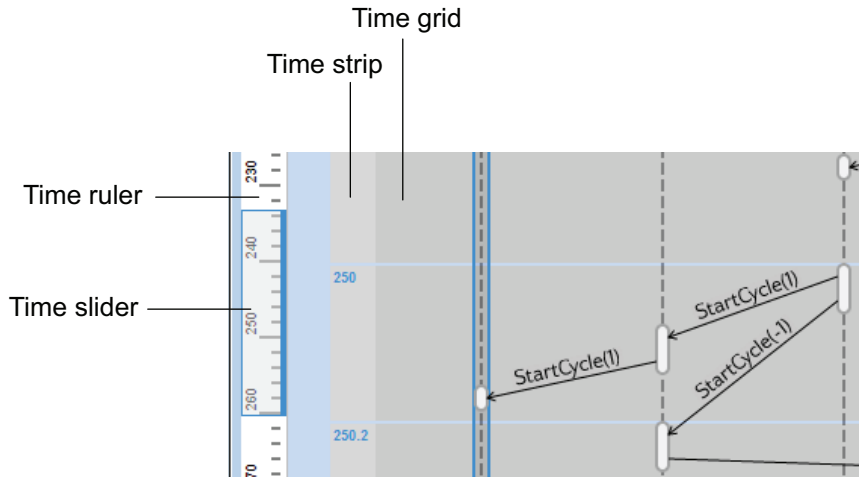
For example, in the model `slexPrinterExample`, a subsystem calls the Simulink Function block `addPrinterJob`. The function block replies with an output value of `false`.



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 Simulink 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 block 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

- “Communicate with Simulink Subsystems by Broadcasting Events” (Stateflow)
- “Communicate with Stateflow Charts by Sending Messages” (Stateflow)

6

Learning More About SimEvents Software

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 Replicator | Entity Server | Entity Terminator | MATLAB Discrete Event System | Multicast Receive Queue | Resource Acquirer

Entity Priorities

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

You specify entity priorities when you generate entities. You can later change entity priorities using an event action for the priority. For example, in the Entity Generator **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 event calendar includes event types such as:

- Entity generation
- Entity forwarding
- Entity destruction
- Timer
- Service completion

The event calendar sorts events based on times and associated entity priorities as outlined here:

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, it is undefined which event is served first. To get deterministic order, change one of the entity priorities.

For example, assume a forward event 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.

See Also

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

Related Examples

- “Sort by Priority” on page 2-6

Livelock Prevention

Large Finite Numbers of Simultaneous Events

Simultaneous events are events that occur at the same simulation clock time. If your simulation creates a large number of simultaneous events, this number might be an indication of an unwanted livelock situation. In this situation, a block returns to the same state infinitely often at the same time instant. SimEvents software prevents livelock with these limits:

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

See Also

More About

- “Information About Race Conditions and Random Times”

Storage and Nonstorage Blocks

In this section...

Storage Blocks

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

Nonstorage Blocks

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 Releaser
- Resource Pool

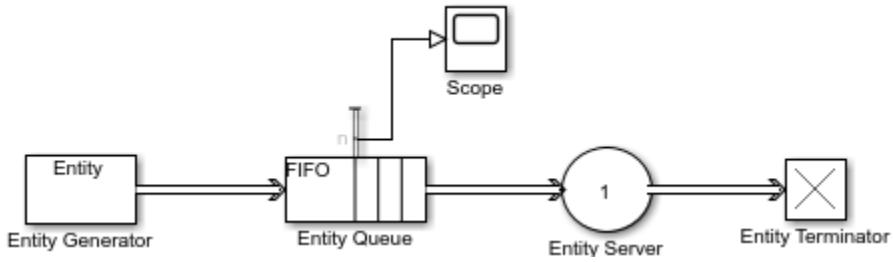
See Also

Composite Entity Creator | 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 | Resource Acquirer | Resource Pool | Resource Releaser

Save SimEvents Simulation State with SimState

This example shows how to save and restore the simulation state of a SimEvents model as `SimState` and use it as an initial state for future simulations. For more information about using `SimState`, see “Save and Restore Simulation State as SimState” (Simulink).

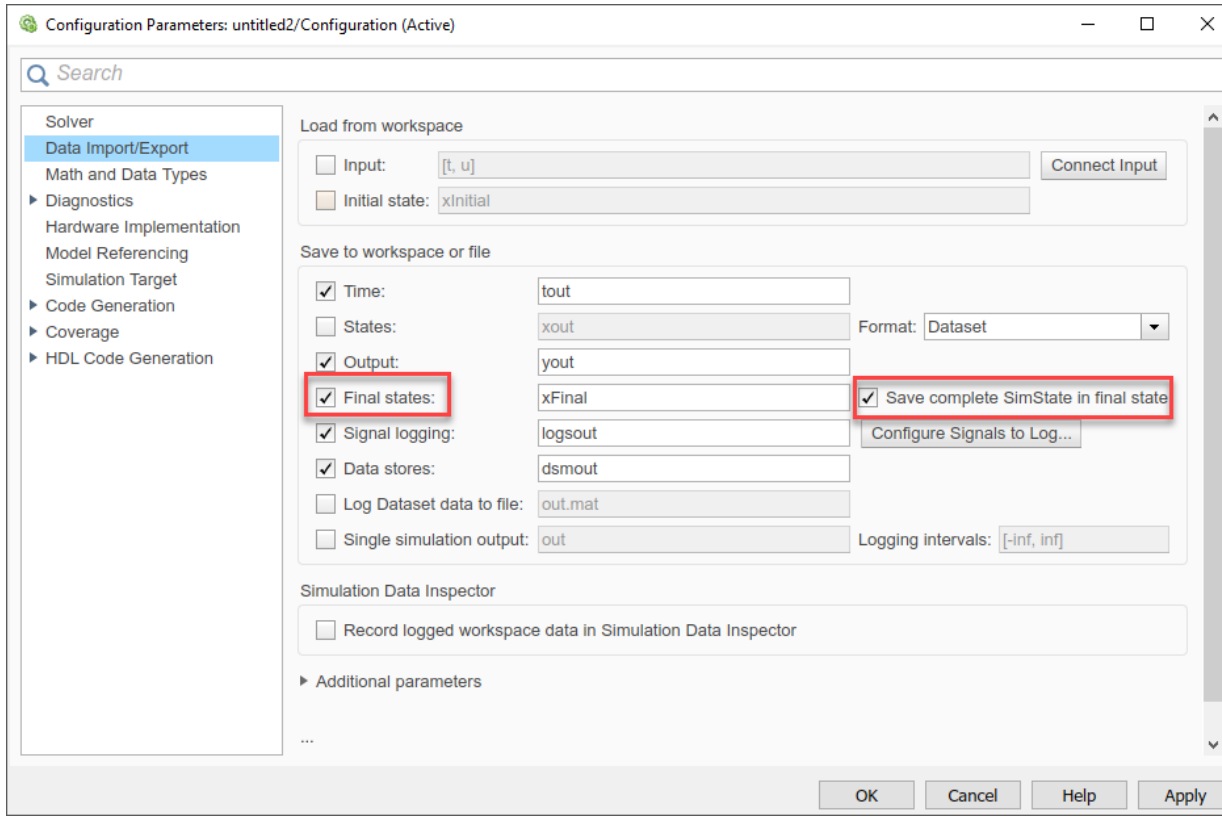
`SimState` is used to save the state of a simple queuing system with an Entity Generator block, an Entity Queue block, an Entity Server block, and an Entity Terminator block. The signal output port **n** displaying the number of entities departed the Entity Queue block is connected to a Scope block. 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.

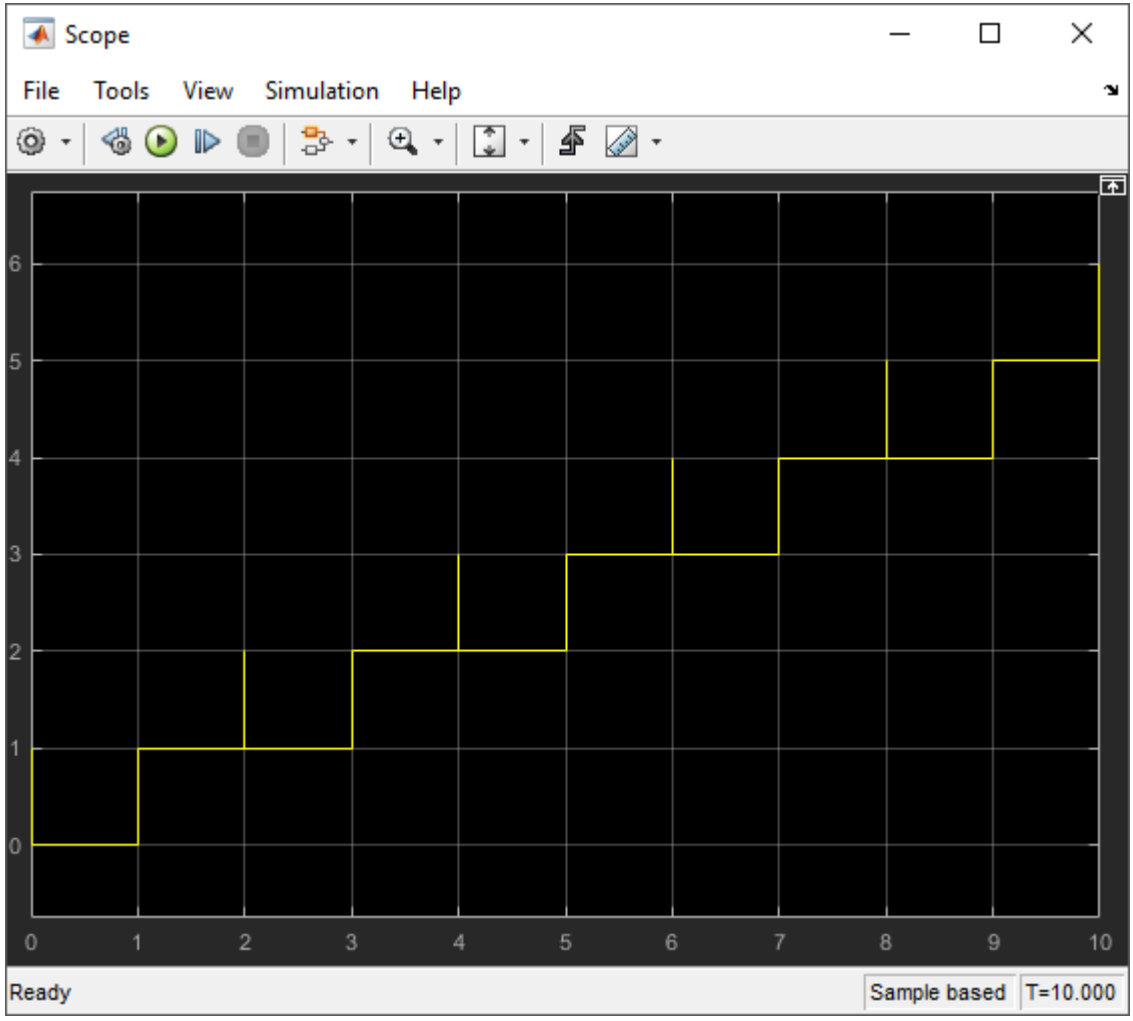
   The queue length increases throughout the simulation because service time is larger than the entity intergeneration time.

2. Select **Simulation > Model Configuration Parameters**. In the Configuration Parameters dialog box, in the **Data Import/Export** pane, select the **Final states** checkbox with the variable name `xFinal` and select the **Save complete SimState in final state** checkbox.



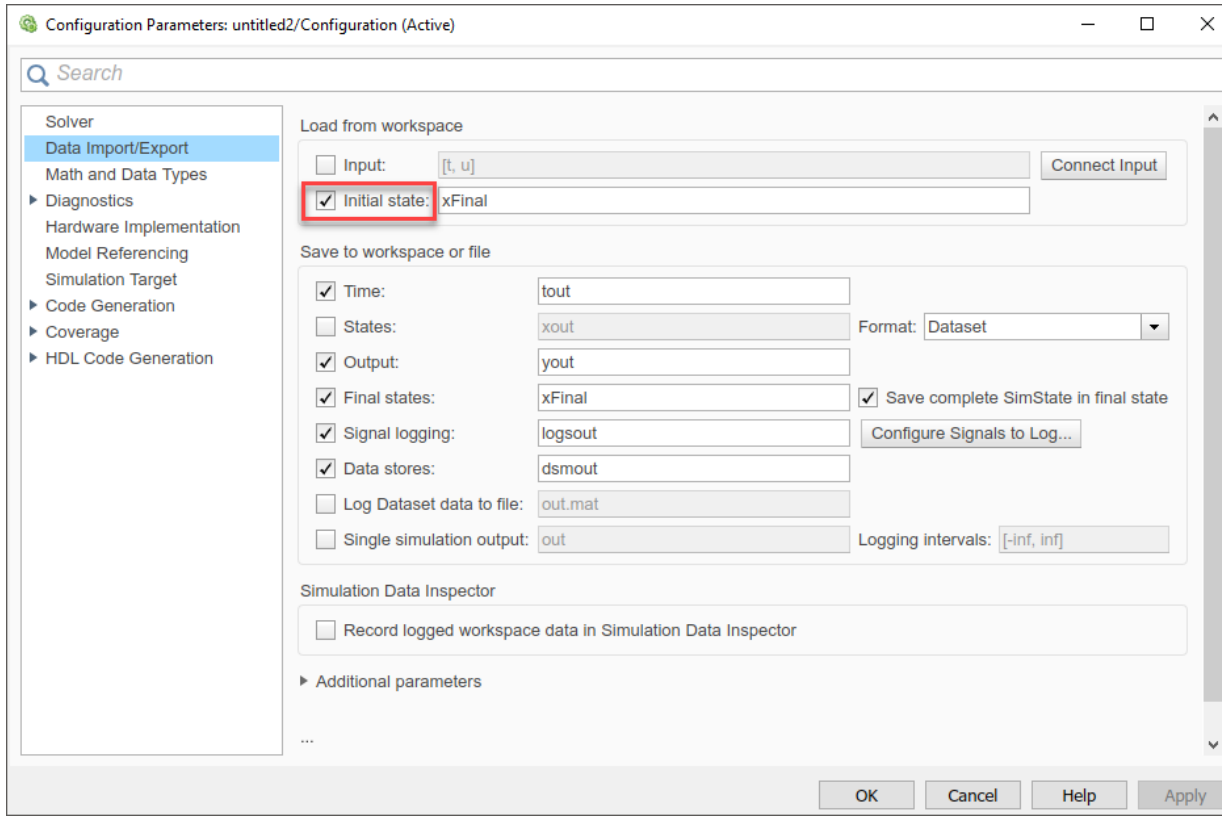
3 Simulate the model and open the Scope block. Observe that the final queue length is 6.

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** checkbox and specify the variable name as `xFinal`.

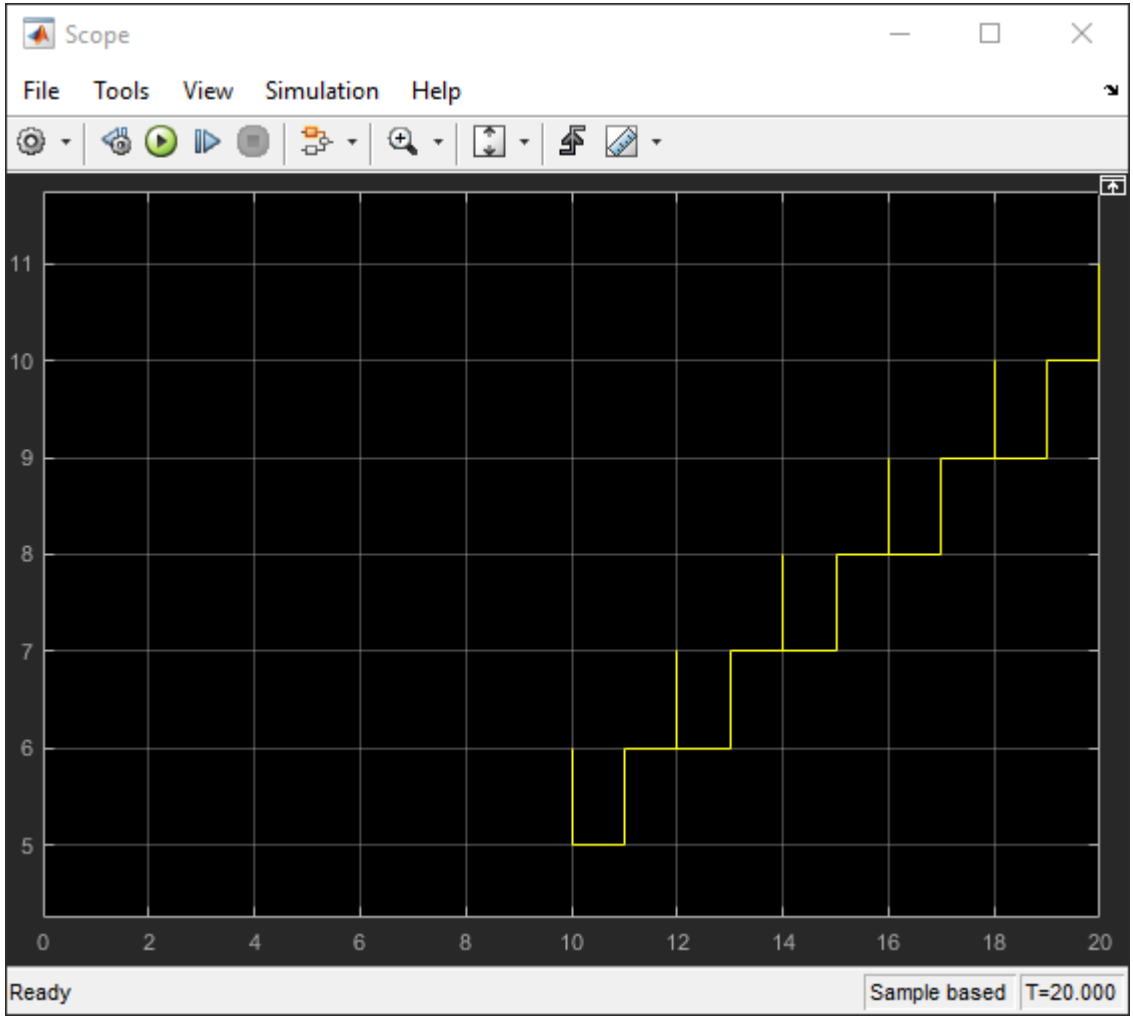
`xFinal` is used as an initial state for the next simulation.



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 7-6
- “Debug SimEvents Models” on page 12-3
- “Manage Entities Using Event Actions”

Use SimEvents with Simulink

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 Message 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 “Systems and Subsystems” (Simulink).

Discrete-Event Blocks in Virtual Subsystems

You can use discrete-event blocks without restriction in a virtual subsystem.

Discrete-Event Blocks in Nonvirtual Subsystems

For more information about atomic subsystems, see Subsystem, Atomic Subsystem, Nonvirtual Subsystem, CodeReuse Subsystem.

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.

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 7-3. 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 the selection of the **Analyze all choices during update diagram and generate preprocessor conditionals** check box for these blocks:

- Variant Subsystem
- Variant Sink
- Variant Source

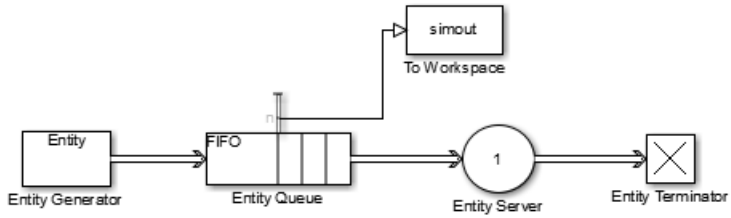
Save Simulation Data

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 **Simulation > Pause**.

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 Runtime Data from Simulation” (Simulink).

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 1-5

- “Generate Entities When Events Occur” on page 1-11

More About

- “Solvers for Discrete-Event Systems” on page 7-6

Solvers for Discrete-Event Systems

In this section...

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.

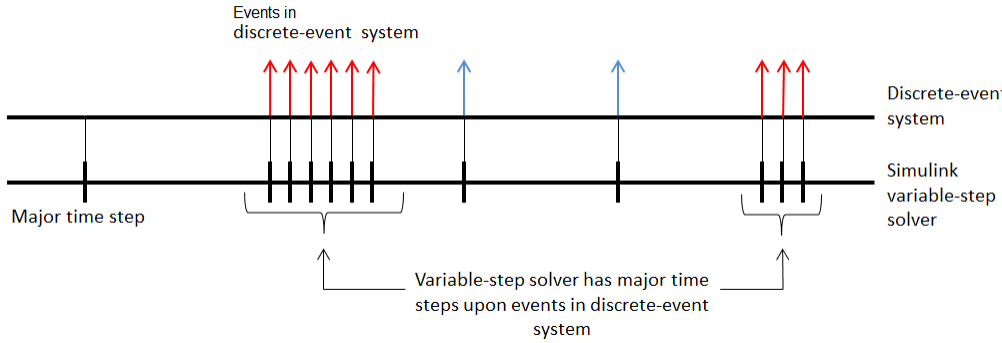
When choosing a solver type for your model, use the following guidelines:

- 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.

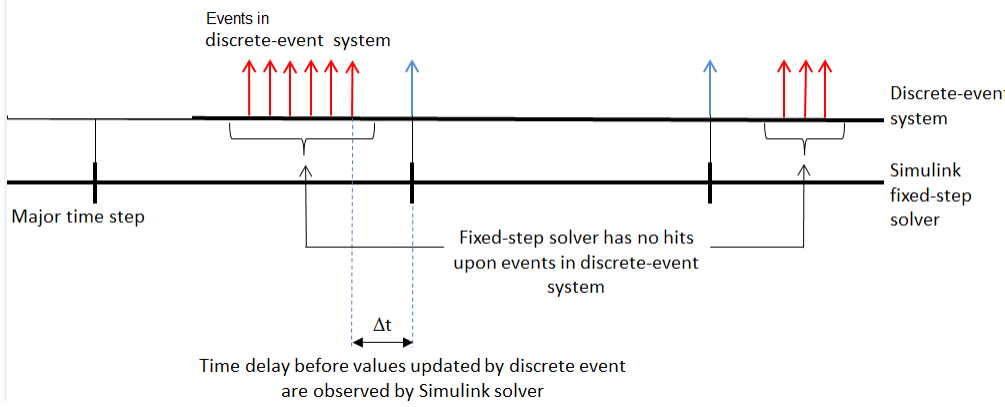


Fixed-Step Solvers for Discrete-Event Systems

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.



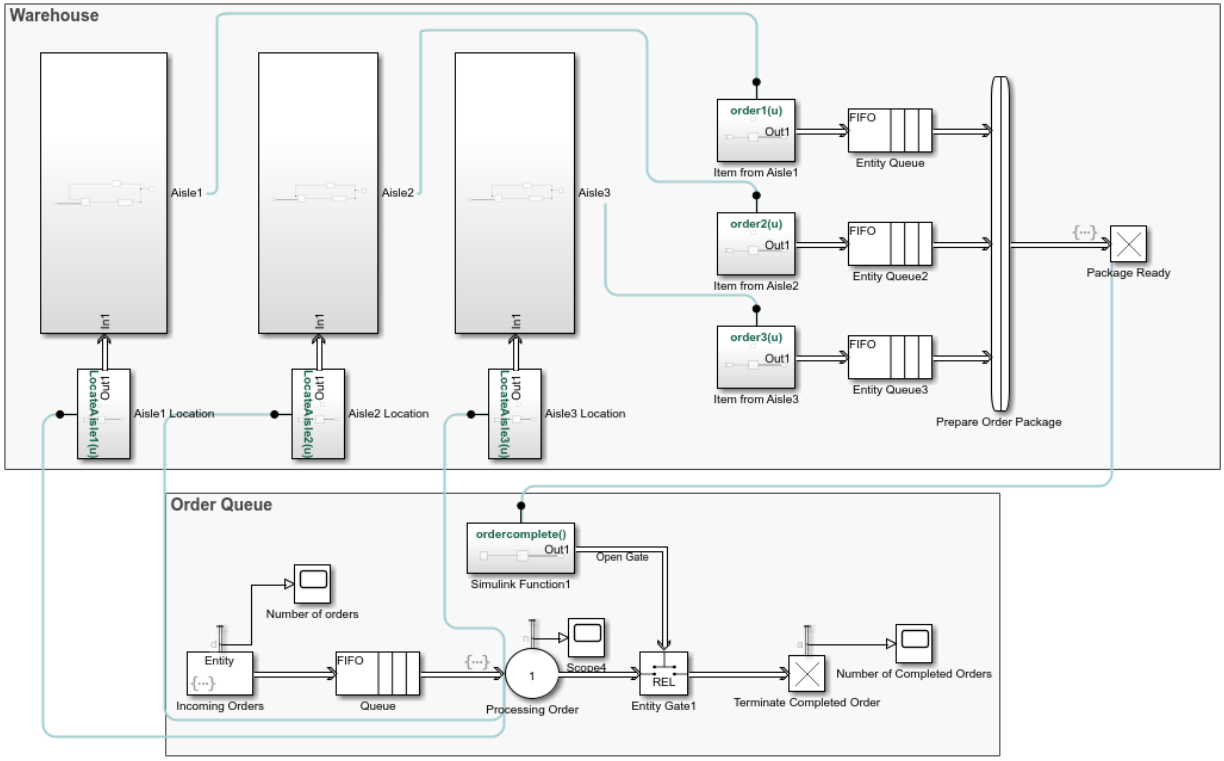
See Also

More About

- “Solvers” (Simulink)
- “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.



The model has two major components.

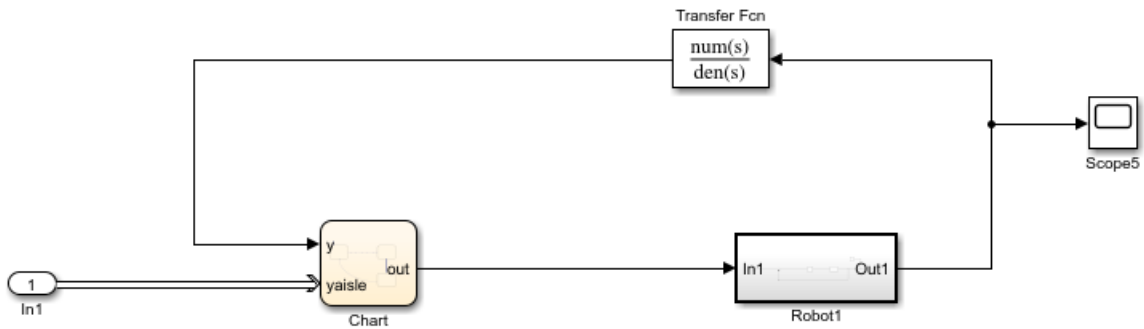
- The Order Queue component represents an online order queue with the blocks from the SimEvents library.

- 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 example, 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.

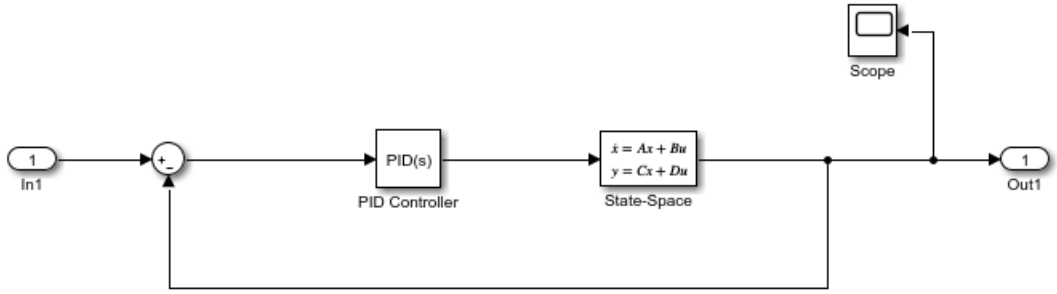
Warehouse Component

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.



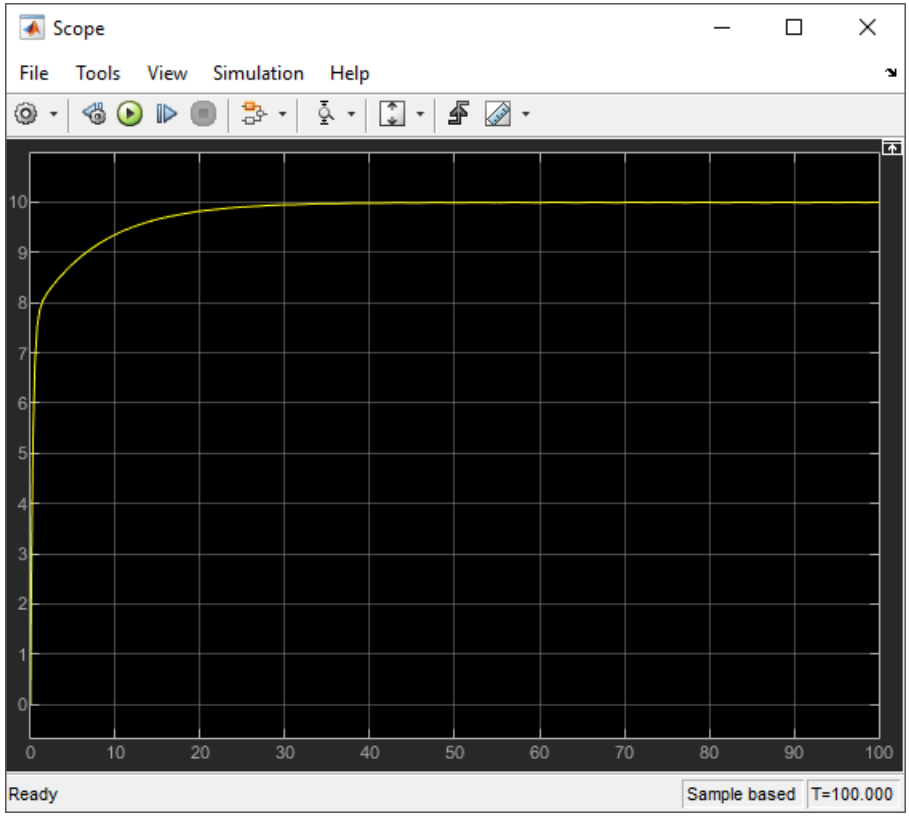
Robot1 Subsystem

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.

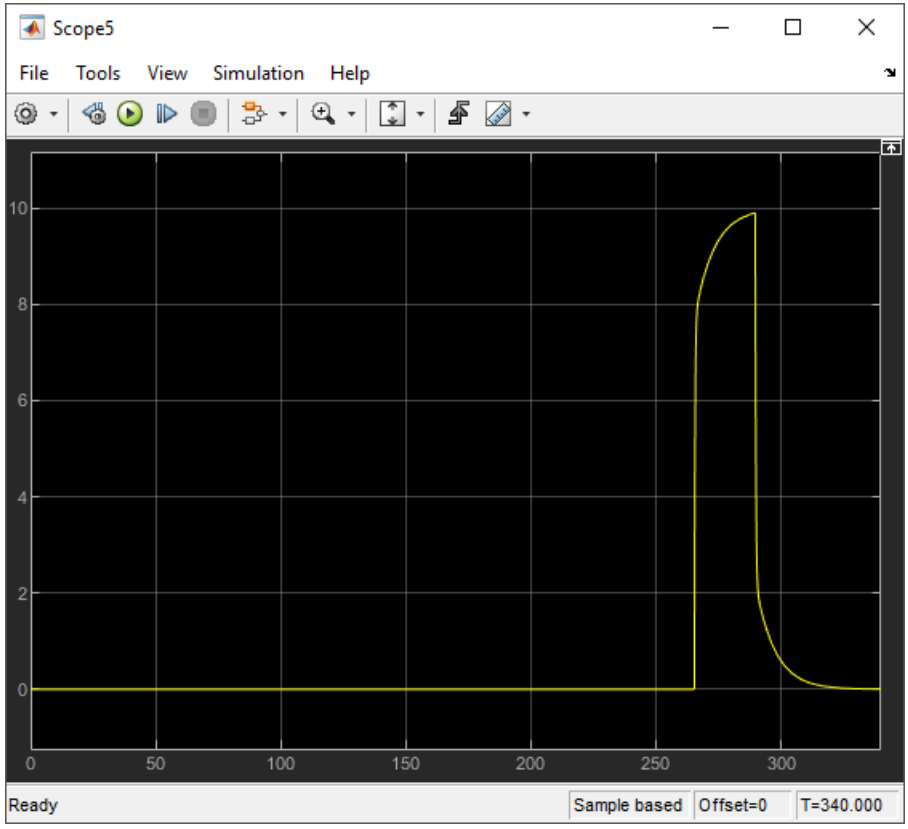
For instance, if the signal from the In1 block is a constant with value `10`, starting from the initial state `0`, the output of the system converges to `10`.



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.

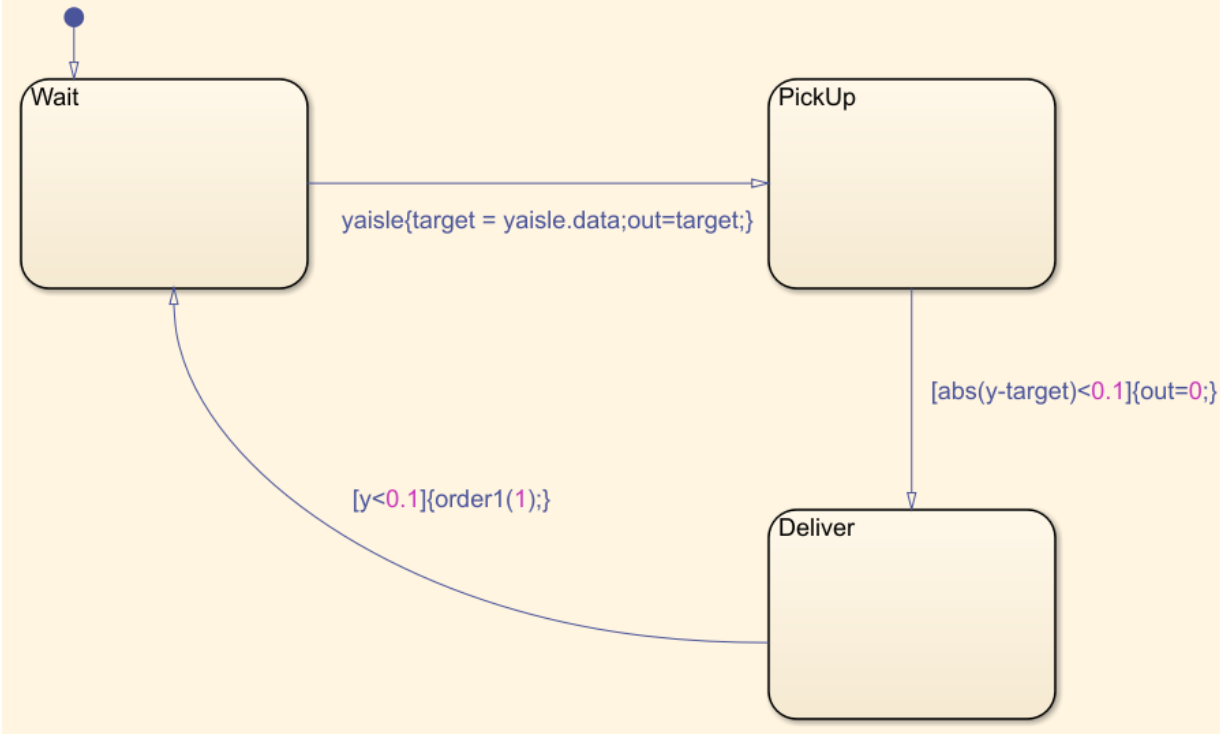
Scheduler

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`.

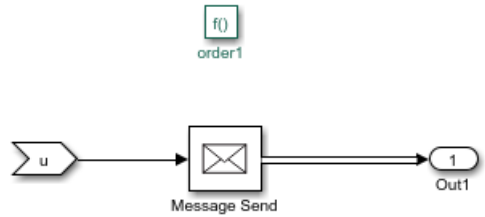
- When a robot is `0.1` units away from *yaisle*, it picks up the item. Then, the robot transitions to a `Deliver` state. The `out` signal becomes `0` and the robot returns back to $y = 0$ for delivery. At the simulation time `290`, Robot1 subsystem is `0.1` unit away from $y = 10$ and transitions to 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);
```

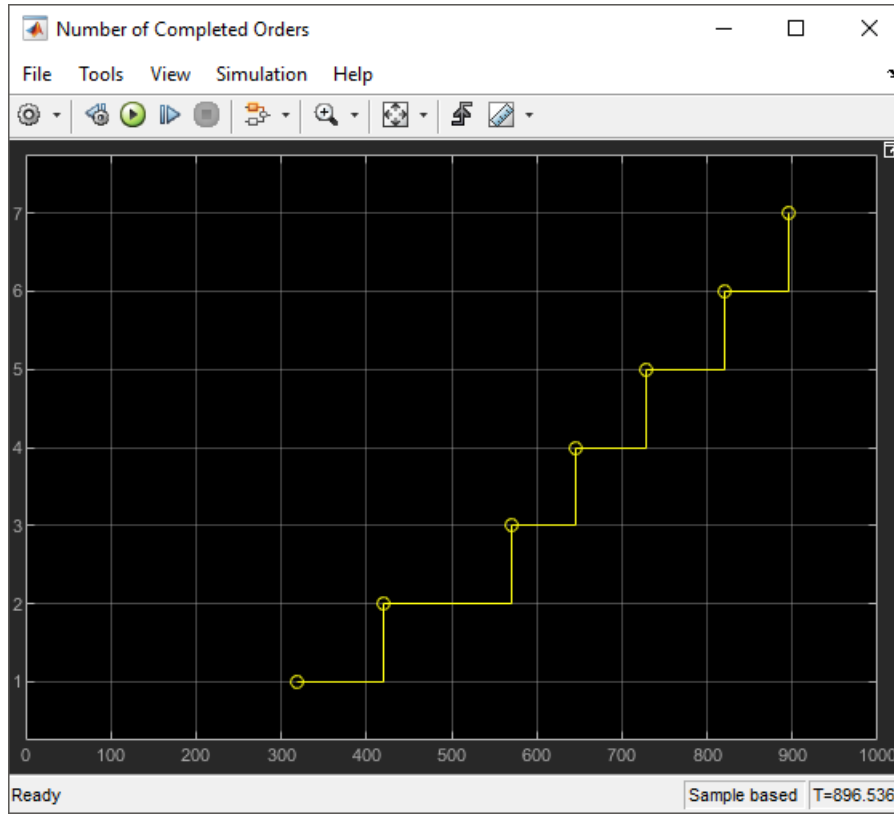
Calling the `LocateIsle()` function communicates the *yaisle* coordinate of an item to the corresponding robot.

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 7-6

8

Build Discrete-Event Systems Using Charts

Discrete-Event Systems Created with Stateflow Charts

Why Use the Discrete Event Chart

A Stateflow discrete-event chart can receive, process, and send SimEvents entities. Using Stateflow discrete-event charts to create SimEvents systems lets you take advantage of:

- Graphical state transition and MATLAB action language used in Stateflow software
- Precise timing for temporal events arrival
- Triggering on message
- Dynamic event scheduling

Note With SimEvents and its required 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.

See Also

Discrete Event Chart

Related Examples

- “Specify Chart Properties” (Stateflow)

More About

- “How Discrete-Event Charts Differ from Stateflow Charts” on page 8-3
- “Event Triggering in Discrete-Event Charts” on page 8-5

How Discrete-Event Charts Differ from Stateflow Charts

In this section...

Discrete Event Chart Properties

Discrete event chart properties allow you to specify how your chart interfaces with the Simulink model.

Set Properties for a Chart

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.

   - **Name**
   - **Machine**
   - **Saturate on integer overflow**
   - **Create data for monitoring**
   - **Lock Editor**

Notes:

- SimEvents software supports only MATLAB action language
- SimEvents always supports variable-size arrays

Define Message (Entity) Input and Output

A discrete-event chart uses SimEvents entities the same way that Stateflow software uses messages. As with Stateflow charts, you can add message (entity) input and output using the Stateflow Editor or Model Explorer. Based on the desired scope, select one of the following options:

Scope	Menu Option
Input	**Message (Entity) Input from Simulink**
Output	**Message (Entity) Output from Simulink**

Define Local Messages

As with Stateflow charts, 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, select **Chart > Add Other Elements > Local Message (Entity)...**.

Specify Message Properties

Discrete-event charts have this additional property 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.

See Also

Discrete Event Chart

More About

- “Discrete-Event Systems Created with Stateflow Charts” on page 8-2
- “Event Triggering in Discrete-Event Charts” on page 8-5

Event Triggering in Discrete-Event Charts

In this section...

Event Triggering

SimEvents discrete-event system charts support these events in the chart:

- Message
- Temporal
- Local
- Implicit (`enter`, `exit`, `on`, `change`)

SimEvents discrete-event system charts do not support these events in the chart:

- Conditions without 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.

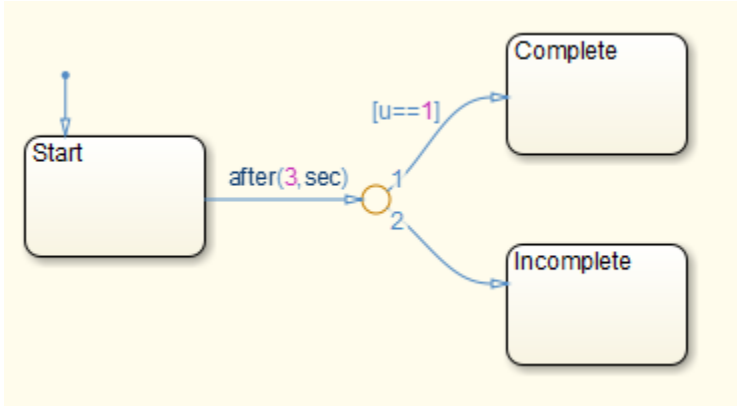
Message Triggering

When a message 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.
- 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 honors the specified time delay value exactly. 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.

Operator	Description
`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, the software supports both `after` and `before`.

See Also

Discrete Event Chart

More About

- “Discrete-Event Systems Created with Stateflow Charts” on page 8-2
- “How Discrete-Event Charts Differ from Stateflow Charts” on page 8-3

Build Discrete-Event Systems Using System Objects

Create Custom Blocks Using MATLAB Discrete-Event System Block

In this section...

Why Use the MATLAB Discrete-Event System Block

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 System objects to create this custom block for SimEvents models. You can author such discrete-event systems via a set of MATLAB methods.

You can create a custom discrete-event system from scratch that:

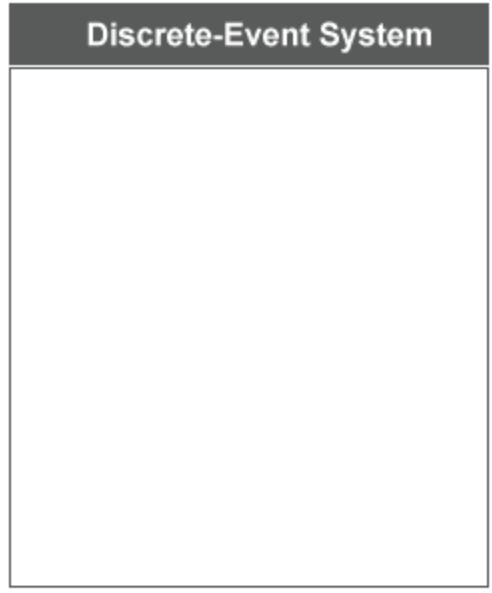
- Contains multiple entity storage elements, with each storage element containing multiple SimEvents entities, and configure it to sort entities in a particular order.
- Has an entity or a storage element that can schedule and execute multiple types of events. These events can model activities such as entity creation, consumption, search, transmission (send/receive), 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.
- Utilize MATLAB toolboxes for computation and scaling of complex systems.

The MATLAB Discrete-Event System block is similar to the MATLAB System block with the following exceptions:

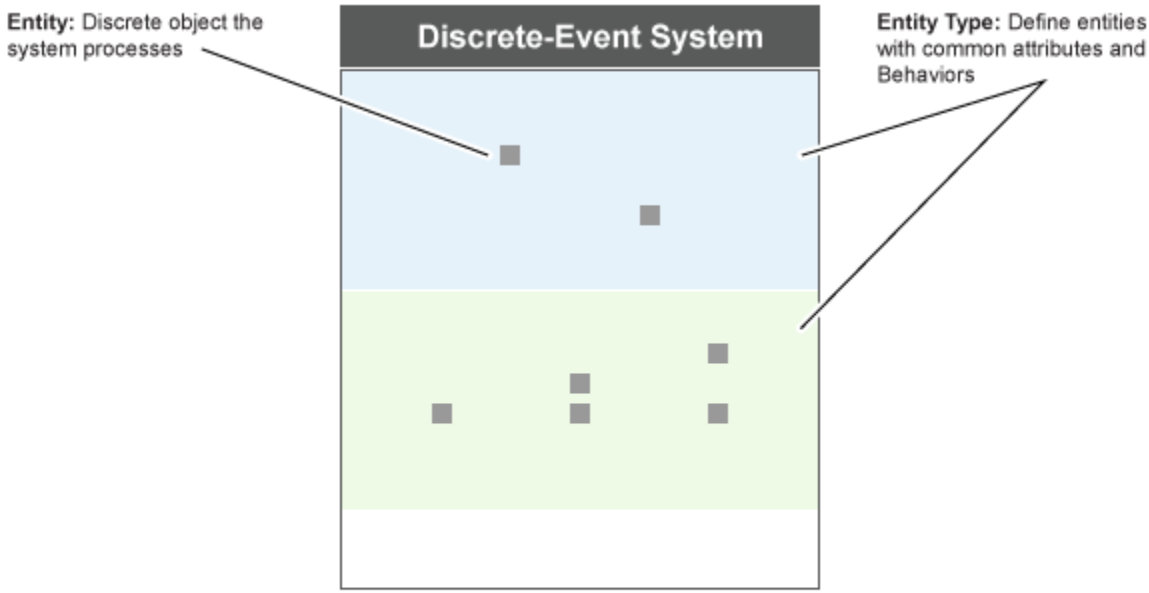
- The resulting System object is an instantiation of the `matlab.DiscreteEventSystem` class rather than the `matlab.System` class.
- The `matlab.DiscreteEventSystem` has its own set of System object methods particular to discrete-event systems. For a complete list, see `matlab.DiscreteEventSystem`. Use these methods to define static properties or define the behavior of objects.

- The `matlab.DiscreteEventSystem` also inherits a subset of the MATLAB System methods. For a complete list of this subset, see `matlab.DiscreteEventSystem`.

Discrete System Framework



Starting with a blank canvas, MATLAB discrete System objects provide a framework to illustrate the behavior of your discrete-event system.

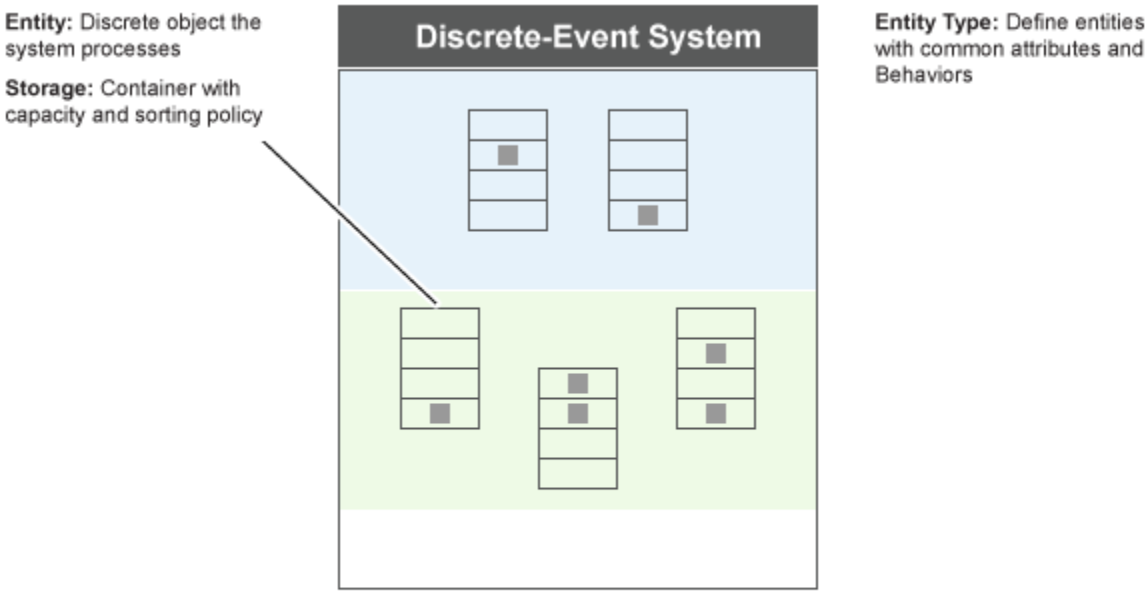


In a discrete-event system, an entity is a discrete object that the system processes. An entity has a type. An entity type defines a class of entities that share a common set of data specifications and run-time methods. Examples of data specifications include dimensions, data type, and complexity.

Consider these guidelines when defining custom entity types using the `getEntityTypesImpl` method:

- You can specify multiple entity types in one discrete-event system. 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.

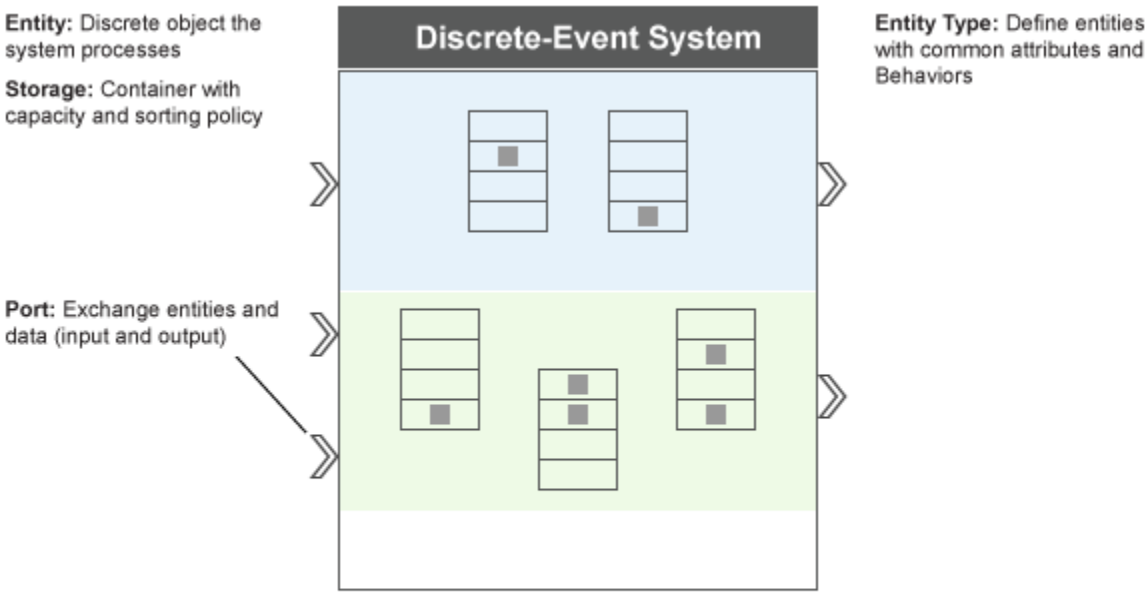
- When forwarding an entity, the source and destination data specifications must be same in these instances:
  - From an input port to a storage element
  - Between storage elements
  - From a storage element to an output port



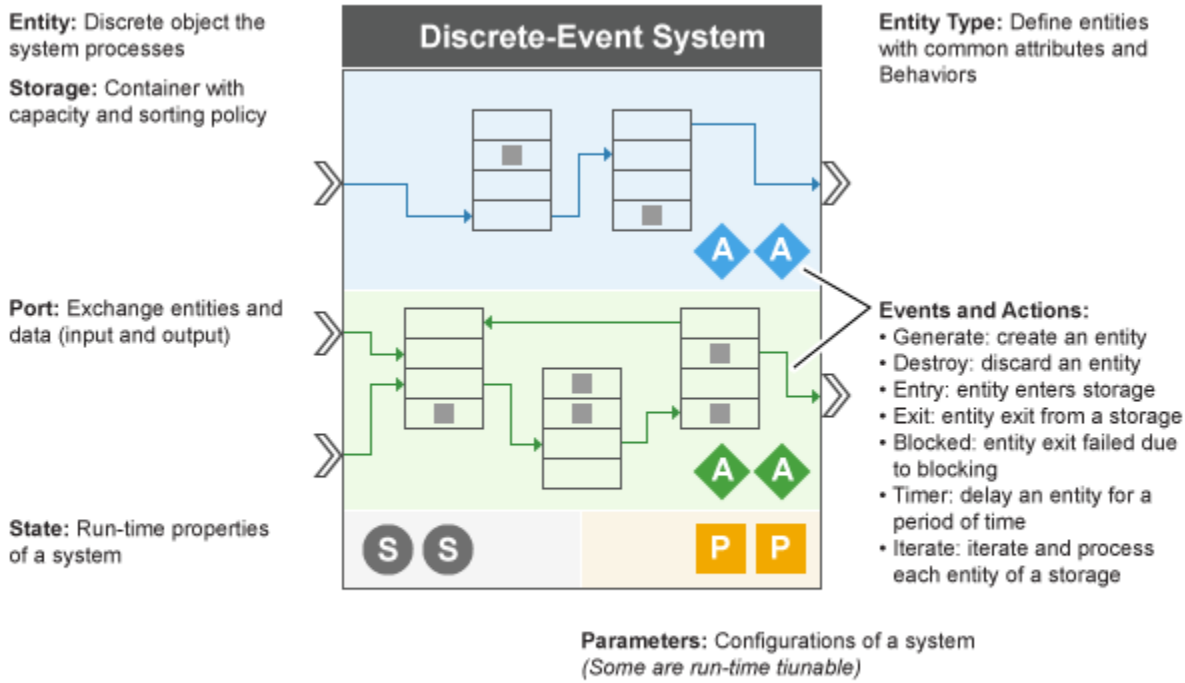
During simulation, an entity always occupies a unit of storage space. Such storage spaces are provided by entity storage elements. A MATLAB discrete-event system can contain multiple entity storage elements. Use the `getEntityStorageImpl` method to specify storage elements. A storage space is a container 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`.



Ports enable a discrete-event system to exchange entities and data with other blocks or systems. A MATLAB discrete-event system supports 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.



You can schedule events for a discrete-event system to execute. Events are associated with user-defined actions. An event action defines how the system behaves by changing state or entity values, and executing the next events of the system.

A MATLAB discrete-event system can have these types of events:

- Storage events — Schedule these events on a storage element. The actor is a storage element.
  - Generate a new entity inside a storage element.
  - Iterate each entity of a storage element.
- Entity events — Schedule these events on an entity. Actor is an entity.
  - Delay an entity.

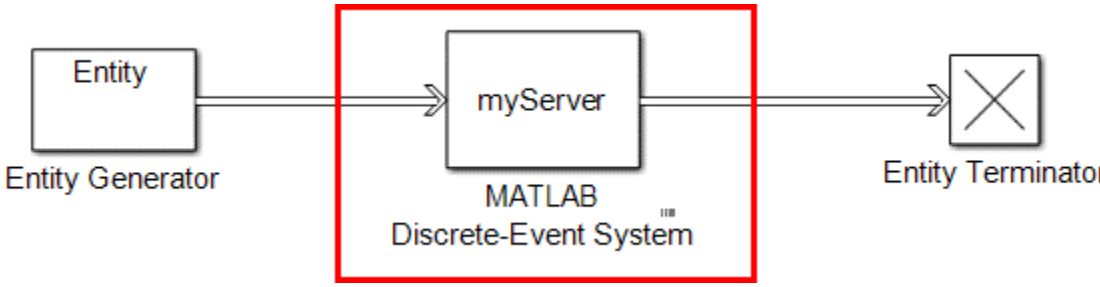
- Forward an entity from its current storage to another storage or output port.
- Destroy the existing entity of a storage element.

A MATLAB discrete-event system provides methods and functions to:

- Schedule events
- Define event actions in response to events
- Initialize events
- Cancel events

Create a Custom Entity Server Block

In this example, you create a custom entity server block. The server block has five servers. The servers serve entities, at entry of the block, for one second. The server outputs each entity through the output port.

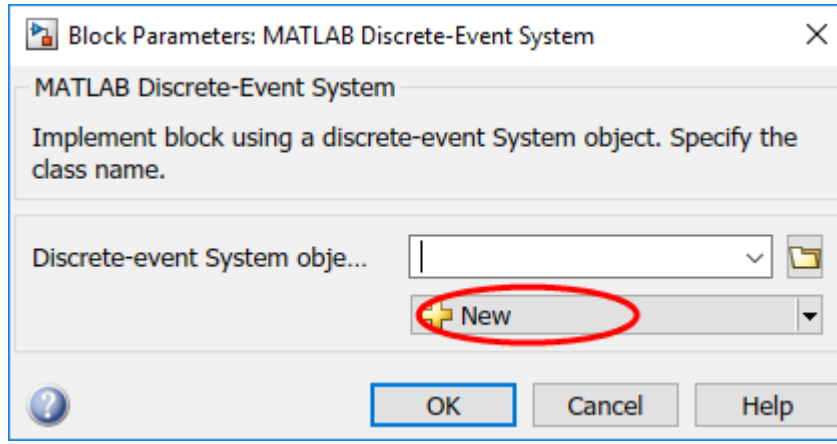


Create the Model

Implement a block and assign a System object to it.

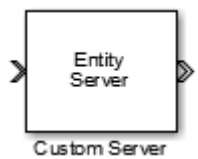
1 Open a new model and add the MATLAB Discrete-Event System block from the SimEvents library.

2 In the block dialog box, from the **New** list, select `Basic` if you want to create a System object from a template. Modify the template as needed and save the System object.

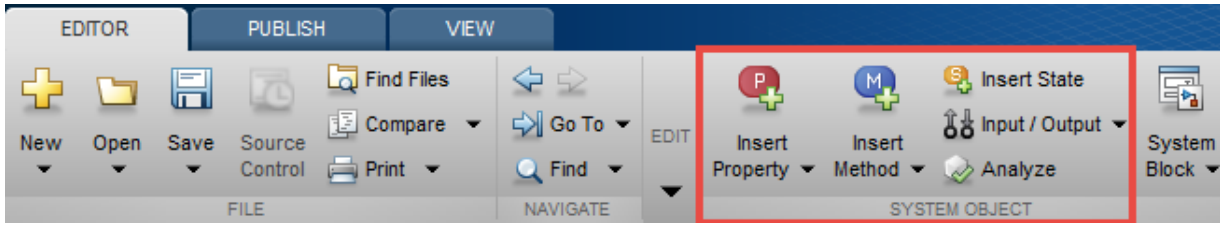


3 If the System object exists, enter its name in the **Discrete-event System object name**. Click the list arrow. If valid System objects exist in the current folder, the names appear in the list.

The MATLAB Discrete-Event System block icon and port labels update to the icons and labels of the corresponding System object. For example, suppose that you select a System object named `desCustomServer` in your current folder. The block updates as shown.



You can define Discrete-Event System objects from the MATLAB Editor using code insertion options.



A template is provided. By selecting **Insert Property** or **Insert Method**, the MATLAB Editor adds predefined properties, methods, states, inputs, or outputs to your System object.

```
classdef Untitled3 < matlab.DiscreteEventSystem
    % Untitled3 Add summary here
    %
    % This template includes the minimum set of functions required
    % to define a Discrete Event System object.

    % Public, tunable properties
    properties

    end

    properties(DiscreteState)

    end

    % Pre-computed constants
    properties(Access = private)

    end

    % Discrete-event algorithms
    methods
        function [entity,events] = entry(obj,storage,entity,source)
            % Specify event actions when entity enters storage
            events = [];
        end
    end

    methods(Access = protected)
        function setupImpl(obj)
            % Perform one-time calculations, such as computing constants
```

```
        end

        function resetImpl(obj)
            % Initialize / reset discrete-state properties
        end
    end
end
```

Use these tools to create and modify System objects faster, and to increase accuracy by reducing typing errors.

Write Code for Custom Entity Server

To create your custom entity server, you modify the template code as follows:

1. Edit the Base MATLAB Object. To simulate your customer entity server, your MATLAB System Object inherits `matlab.DiscreteEventSystem` .
2. Add parameters. The parameters capture the properties of your server such as the number of servers and how long they service an entity.
   - Tunable parameters are parameters that can be tuned during run time. For your entity server, the service time is tunable.
   - Nontunable parameters are parameters that cannot be tuned during run time. For your entity server, the number of servers or capacity is nontunable.
3. Add methods
   - The storage method defines how your custom entity server stores and sorts the entities. MATLAB System Object provides the function `getEntityStorageImpl(obj)`, which allows you to specify storage specifications and define the inputs and outputs for your entity server.
   - MATLAB System Object provides the function:`entry(obj, storage, entity, from)` and `obj.eventForward('output', 1, obj.ServiceTime)`. With these functions you define service at entry, implement service time, and output the entity through an output port.

Your final MATLAB System Object code looks like:

```
classdef myServer < matlab.DiscreteEventSystem
    % Custom entity server with capacity and service time
    % as parameters

    properties (Nontunable)
```

```
            % Number of servers
            Capacity = 5;
        end

        properties
            % Service time
            ServiceTime = 1.0;
        end

        methods (Access=protected)

            function entityTypes = getEntityTypesImpl(obj)
                % Specify entity type
                entityTypes = obj.entityType('myType');
            end

            function [inputTypes,outputTypes] = getEntityPortsImpl(obj)
                % Specify entity type at input and output ports
                inputTypes = {'myType'};
                outputTypes = {'myType'};
            end

            function [storageSpecs, I, O] = getEntityStorageImpl(obj)
                % Specify storage with capacity from a parameter.
                % Connect the
                % storage to both input and output port.
                storageSpecs = obj.queueFIFO('myType', obj.Capacity);
                I = 1;
                O = 1;
            end

        end
        methods
            function [entity, events] = entry(obj, storage, entity, from)
                % Forward incoming entity to output port after service
                % completes
                events = obj.eventForward('output', 1, obj.ServiceTime);
            end
        end
    end
```

Many different MATLAB System Object functions allow you to capture the properties and behaviors of your unique discrete-event system. The model in this example 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 system, and interface it with the graphical programming features of SimEvents.

For examples of MATLAB Discrete-Event System and System objects, type SimEvents Examples in the SimEvents Help browser.

In addition, in the SimEvents library, double-click the Design Patterns block. The **MATLAB Discrete-Event System** category contains these discrete-event system design patterns:

Example	Application
Custom Generator	Implement a more complicated entity generator.
Selection Queue	Select a specific entity to output from a queue.

See Also

`matlab.DiscreteEventSystem` | `matlab.System`

Related Examples

- “Use a MATLAB Discrete-Event System Block” on page 9-14

More About

- “Integrate System Objects Using MATLAB System Block” (Simulink)
- “Implement a Discrete-Event System Object” on page 9-16
- “Custom Entity Types, Ports, and Storage” on page 9-24
- “Work with Events” on page 9-27

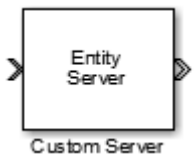
Use a MATLAB Discrete-Event System Block

Implement a block and assign a System object to it. You can then explore the block to see the effect.

1. Create a 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` if you want to create a System object from a template. Modify the template according to your needs and save the System object.
3. If the System object exists, enter its name in the **Discrete-event System object name**. Click the list arrow. If valid System objects exist in the current folder, the names appear in the list.

   The MATLAB Discrete-Event System block icon and port labels update to the icons and labels of the corresponding System object. For example, suppose that you selected a System object named desCustomServer in your current folder. The block updates as shown in the figure:



See Also

`matlab.DiscreteEventSystem` | `matlab.System`

More About

- “Integrate System Objects Using MATLAB System Block” (Simulink)
- “Create Custom Blocks Using MATLAB Discrete-Event System Block” on page 9-2

- “Implement a Discrete-Event System Object” on page 9-16
- “Custom Entity Types, Ports, and Storage” on page 9-24
- “Work with Events” on page 9-27

Implement a Discrete-Event System Object

In this section...

The `matlab.DiscreteEventSystem` provides methods that let you work with these elements of a discrete-event system:

- Static properties of the object entity types, ports, and storage
  - `getEntityPortsImpl`
  - `getEntityStorageImpl`
  - `getEntityTypesImpl`
- Event initialization
  - `setupEvents`
- Runtime behavior of the object
  - `blocked`
  - `destroy`
  - `entry`
  - `exit`
  - `generate`
  - `iterate`
  - `timer`

While implementing these methods, define entity type, entity storage, create, schedule, and cancel events. Use these functions:

- Define entity storage
  - `queueFIFO`
  - `queueLIFO`
  - `queuePriority`
- Create and schedule events

  - `eventGenerate`
  - `eventIterate`
  - `eventTimer`
  - `eventForward`
  - `eventDestroy`
- Cancel events
  - `cancelGenerate`
  - `cancelIterate`
  - `cancelTimer`
  - `cancelForward`
  - `cancelDestroy`
- Define entity type
  - `entityType`

Extract or Modify Entities

If an entity that is a part of a MATLAB Discrete-Event System block is requested for extraction, the `exit` method of the block will be triggered. When `exit` method is called, its **destination** argument is set to `extract`. See `modified` for entity modification.

Additional Notes

When referencing entity attributes or system properties in discrete-event System objects, 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

See Also

`matlab.DiscreteEventSystem` | `matlab.DiscreteEventSystem.blockedImpl` | `matlab.DiscreteEventSystem.cancelDestroy` | `matlab.DiscreteEventSystem.cancelForward` | `matlab.DiscreteEventSystem.cancelGenerate` | `matlab.DiscreteEventSystem.cancelIterate` | `matlab.DiscreteEventSystem.cancelTimer` | `matlab.DiscreteEventSystem.destroy` | `matlab.DiscreteEventSystem.entityType` | `matlab.DiscreteEventSystem.entry` | `matlab.DiscreteEventSystem.eventDestroy` | `matlab.DiscreteEventSystem.eventForward` | `matlab.DiscreteEventSystem.eventGenerate` | `matlab.DiscreteEventSystem.eventIterate` | `matlab.DiscreteEventSystem.eventTimer` | `matlab.DiscreteEventSystem.exit` | `matlab.DiscreteEventSystem.generate` | `matlab.DiscreteEventSystem.getEntityPortsImpl` | `matlab.DiscreteEventSystem.getEntityStorageImpl` | `matlab.DiscreteEventSystem.getEntityTypesImpl` | `matlab.DiscreteEventSystem.iterate` | `matlab.DiscreteEventSystem.queueFIFO` | `matlab.DiscreteEventSystem.queueLIFO` | `matlab.DiscreteEventSystem.queuePriority` | `matlab.DiscreteEventSystem.queueSysPriority` | `matlab.DiscreteEventSystem.setupEvents` | `matlab.DiscreteEventSystem.timer` | `matlab.System`

Related Examples

- “Use a MATLAB Discrete-Event System Block” on page 9-14

More About

- “Integrate System Objects Using MATLAB System Block” (Simulink)
- “Create Custom Blocks Using MATLAB Discrete-Event System Block” on page 9-2
- “Custom Entity Types, Ports, and Storage” on page 9-24
- “Work with Events” on page 9-27

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 objects

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 seExampleSchedulerClass file in the Develop Custom Scheduler of a Multicore Control System 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`

Old Method Name	New Method Name
`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 `matlab.DiscreteEventSystem.initEventArray` to initialize arrays.

Before	After
`function events = setupEventsImpl(obj)`	`function events = setupEvents(obj)` `  events = obj.initEventArray;`

Append Elements to Array of Structures

Append elements to array of structures. For example:

Before	After
`events(id) = obj.eventGenerate(1, numevents(id), 0, obj.Priorities(id)); %#ok<*AGROW>`	`events = [events obj.eventGenerate(1, int2str(id), 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, numevents(id), 0, obj.Priorities(id)); %#ok<*AGROW>`	`events = [events obj.eventGenerate(1, int2str(id), 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');`
	`id = 1;`
	`id = str2double(tag);`

- Do not pass System objects 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” (MATLAB)

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” (MATLAB) for more information.

Limitations of Code Generation with Discrete-Event System Block

Limitations include:

- No “Global Variables” (MATLAB)
- “System Objects in MATLAB Code Generation” (Simulink)
- “MATLAB System Block Limitations” (Simulink)

See Also

`matlab.DiscreteEventSystem` | `matlab.DiscreteEventSystem.blockedImpl` | `matlab.DiscreteEventSystem.cancelDestroy` | `matlab.DiscreteEventSystem.cancelForward` | `matlab.DiscreteEventSystem.cancelGenerate` | `matlab.DiscreteEventSystem.cancelIterate` | `matlab.DiscreteEventSystem.cancelTimer` | `matlab.DiscreteEventSystem.destroy` | `matlab.DiscreteEventSystem.entityType` | `matlab.DiscreteEventSystem.entry` | `matlab.DiscreteEventSystem.eventDestroy` | `matlab.DiscreteEventSystem.eventForward` | `matlab.DiscreteEventSystem.eventGenerate` | `matlab.DiscreteEventSystem.eventIterate` | `matlab.DiscreteEventSystem.eventTimer` |

`matlab.DiscreteEventSystem.exit` | `matlab.DiscreteEventSystem.generate` | `matlab.DiscreteEventSystem.getEntityPortsImpl` | `matlab.DiscreteEventSystem.getEntityStorageImpl` | `matlab.DiscreteEventSystem.getEntityTypesImpl` | `matlab.DiscreteEventSystem.iterate` | `matlab.DiscreteEventSystem.queueFIFO` | `matlab.DiscreteEventSystem.queueLIFO` | `matlab.DiscreteEventSystem.queuePriority` | `matlab.DiscreteEventSystem.queueSysPriority` | `matlab.DiscreteEventSystem.setupEvents` | `matlab.DiscreteEventSystem.timer` | `matlab.System`

More About

- “Integrate System Objects Using MATLAB System Block” (Simulink)
- “Create Custom Blocks Using MATLAB Discrete-Event System Block” on page 9-2
- “Implement a Discrete-Event System Object” on page 9-16

Custom Entity Types, Ports, and Storage

In this section...

Entity Types

In a discrete-event system, an 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 in one discrete-event system. Each type must have a unique name.
- An entity storage element, input port, and output port must specify the entity type it works 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 data specifications of source and destination must be same in these instances:
  - From input port to storage
  - Between storage elements
  - From a storage element to output port

For a discrete-event system with multiple entity types, each entity type shares a common set of event action methods. When naming these methods, use this convention:

`entitytypeAction``Impl`

For example, if your discrete-event system has two entity types, `car` and `truck`, use method names such as:

```
carEntryImpl
trucEntryImpl
```

For discrete-event systems with one entity type, you can still use this convention, or use the convention *`action`*`Impl`, such as

```
entryImpl
```

Custom Entity Ports

A MATLAB discrete-event system supports 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.

Custom Entity Storage

A MATLAB discrete-event system can contain multiple entity storage elements. Use the `getEntityStorageImpl` method to specify storage elements. An entity storage is a random-access 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`.

See Also

`matlab.DiscreteEventSystem` |
`matlab.DiscreteEventSystem.getEntityPortsImpl` |
`matlab.DiscreteEventSystem.getEntityStorageImpl` |
`matlab.DiscreteEventSystem.getEntityTypesImpl` |
`matlab.DiscreteEventSystem.queueFIFO` |
`matlab.DiscreteEventSystem.queueLIFO` |
`matlab.DiscreteEventSystem.queuePriority` |
`matlab.DiscreteEventSystem.queueSysPriority` | `matlab.System`

Related Examples

- “Use a MATLAB Discrete-Event System Block” on page 9-14

More About

- “Integrate System Objects Using MATLAB System Block” (Simulink)
- “Create Custom Blocks Using MATLAB Discrete-Event System Block” on page 9-2
- “Implement a Discrete-Event System Object” on page 9-16
- “Work with Events” on page 9-27

Work with Events

In this section...

Event Types

A MATLAB discrete-event system can have the following types of events:

- Storage events — Schedule these events on a storage element. The actor is a storage element.
  - Generate

    Create a new entity inside a storage element.
  - Iterate

    Iterate and process each entity of a storage element.
- Entity events — Schedule these events on an entity. Actor is an entity.
  - Timer

    Delay an entity.
  - Forward

    Move an entity from its current storage to another storage or output port.
  - Destroy

    Destroy the existing entity of a storage element.

You can:

- Schedule events

- Define event actions in response to events
- Initialize events
- Cancel events

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.

Action	Triggering Event	Description
`generate`	Generate	Called after a new entity is created inside a storage element.
`iterate`	Iterate	Upon execution of an iterate event, the discrete-event system calls this method for each entity, starting from the front of the storage, to the back. You can stop the iteration before reaching the last entity. If the entity order must change, the order changes after the entire iteration completes.
`timer`	Timer	Called when a timer of an entity expires (completes).
`entry`	Forward	When an entity is forwarded from storage *A* to storage *B*, the discrete-event system first calls `exit` of *A*, then `entry` of *B*.
`exit`	Forward	When an entity is forwarded from storage *A* to storage *B*, the discrete-event system first calls `exit` of *A*, then `entry` of *B*.
`blocked`	Forward	Upon execution of a forward event, if entity cannot leave due to blocking, the discrete-event system calls the `blocked` action method.
`destroy`	Destroy	The discrete-event system calls this method before an existing entity is destroyed and removed from storage.

Initialization Events

Use the `setupEvents` method to schedule initial events of a discrete-event system. You can schedule only storage events using this method. This method does not have a specific entity type name.

Cancellation of Previously Scheduled Events

Use the `cancel*` methods to cancel previously scheduled events of a discrete-event system.

Additional Notes

- 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.

  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.

See Also

`matlab.DiscreteEventSystem` | `matlab.DiscreteEventSystem.blocked` | `matlab.DiscreteEventSystem.cancelDestroy` | `matlab.DiscreteEventSystem.cancelForward` | `matlab.DiscreteEventSystem.cancelGenerate` | `matlab.DiscreteEventSystem.cancelIterate` | `matlab.DiscreteEventSystem.cancelTimer` | `matlab.DiscreteEventSystem.destroy` | `matlab.DiscreteEventSystem.entry` |

`matlab.DiscreteEventSystem.eventDestroy` |
`matlab.DiscreteEventSystem.eventForward` |
`matlab.DiscreteEventSystem.eventGenerate` |
`matlab.DiscreteEventSystem.eventIterate` |
`matlab.DiscreteEventSystem.eventTimer` |
`matlab.DiscreteEventSystem.exitImpl` |
`matlab.DiscreteEventSystem.generate` |
`matlab.DiscreteEventSystem.iterate` |
`matlab.DiscreteEventSystem.setupEvents` |
`matlab.DiscreteEventSystem.timer` | `matlab.System`

Related Examples

- “Use a MATLAB Discrete-Event System Block” on page 9-14

More About

- “Integrate System Objects Using MATLAB System Block” (Simulink)
- “Create Custom Blocks Using MATLAB Discrete-Event System Block” on page 9-2
- “Implement a Discrete-Event System Object” on page 9-16
- “Custom Entity Types, Ports, and Storage” on page 9-24

Custom Visualization

Interface for Custom Visualization

In this section...

SimulationObserver Class

To create an observer, create a class that derives from the `simevents.SimulationObserver` object. You can use observers to implement animators to visualize model simulation, or debuggers.

- 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”.

See Also

`simevents.SimulationObserver`

Related Examples

- “Create an Application” on page 10-4
- “Use the Observer to Monitor the Model” on page 10-7
- “Stop Simulation and Disconnect the Model” on page 10-8
- “Custom Visualization Examples” on page 10-9

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`

Action	Method
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 seExampleRestaurantAnimator < simevents.SimulationObserver
   ```

   `seExampleRestaurantAnimator` 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 = seExampleRestaurantAnimator
               % Constructor
               modelname = 'seExampleCustomVisualization';
               this@simevents.SimulationObserver(modelname);
               this.mModel = modelname;
           end
   ```

The *`matlabroot`*`\toolbox\simevents\examples` folder contains this application example, `seExampleRestaurantAnimator.m`. This example uses an observer object to implement an animator for the `seExampleCustomVisualization` model.

For more information, see **Using Custom Visualization for Entities** in the SimEvents **Examples** tab.

See Also

`simevents.SimulationObserver`

Related Examples

- “Use the Observer to Monitor the Model” on page 10-7

More About

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=seExampleRestaurantAnimator;
   ```

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.

See Also

`simevents.SimulationObserver`

Related Examples

- “Create an Application” on page 10-4
- “Stop Simulation and Disconnect the Model” on page 10-8
- “Custom Visualization Examples” on page 10-9

More About

- “Interface for Custom Visualization” on page 10-2

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

- “Create an Application” on page 10-4
- “Use the Observer to Monitor the Model” on page 10-7
- “Custom Visualization Examples” on page 10-9

More About

- “Interface for Custom Visualization” on page 10-2

Custom Visualization Examples

In this section...

The `Using Custom Visualization for Entities` example visualizes a restaurant layout with customer entities entering, dining, and leaving. It uses `seExampleCustomVisualization` to model a restaurant. To observe the visualization, start the model and the animator.

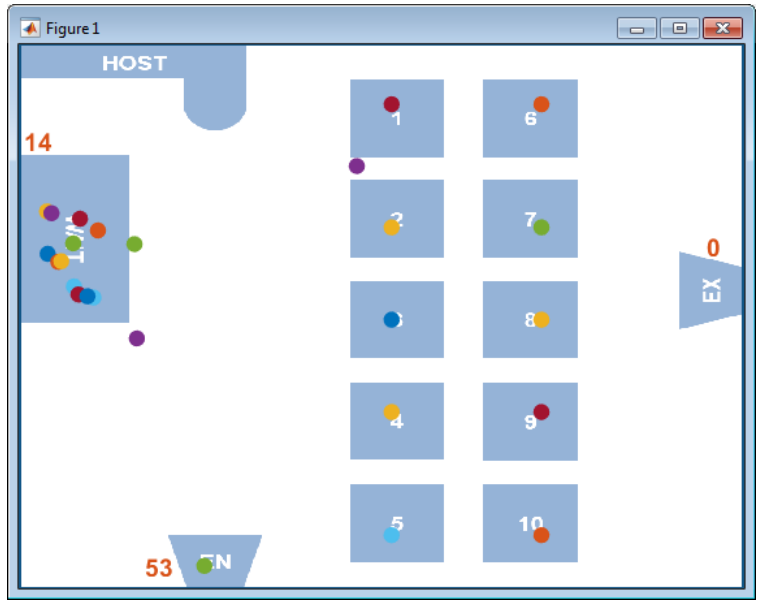
Structure of Example Model

The `seExampleCustomVisualization` model has these major components:

- The Entity Generator block (Patron Enter) generates entities representing customer entities. Each customer has a *TimeToDine* amount of time to dine.
- These customer entities enter a waiting area, where a Resource Acquirer block acquires a table for the customer.
- The Resource Pool block contains 10 table resources.
- When a table entity is available for a waiting customer entity, the Entity Server block serves the customer for a *TimeToDine* amount of time.
- When a customer entity is done dining, the Resource Releaser block releases the table resource back to the resource pool.
- The customer entity leaves the restaurant through the Entity Terminator block (Patron Leave).

Visualize Entities

The `seExampleRestaurantAnimator` application animates the diners entering, dining, and leaving the restaurant. The animator application draws a different colored dot for each customer. As customers move through the restaurant, the application animates the motion of the dots.



See Also

`simevents.SimulationObserver`

Related Examples

- “Create an Application” on page 10-4
- “Use the Observer to Monitor the Model” on page 10-7
- “Stop Simulation and Disconnect the Model” on page 10-8

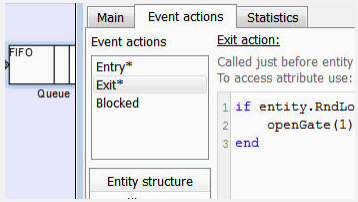
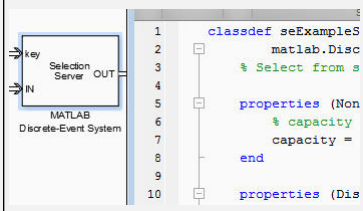
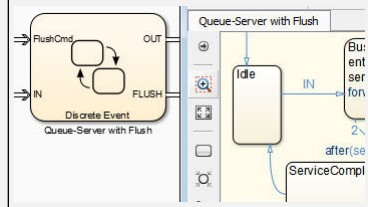
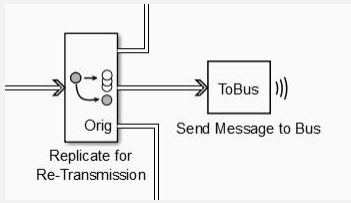
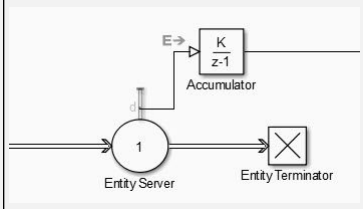
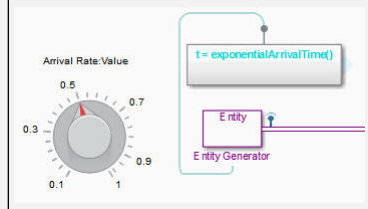
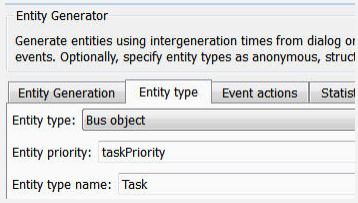
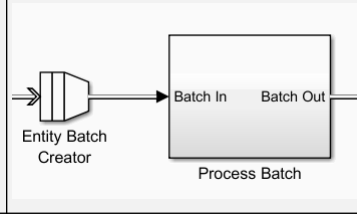
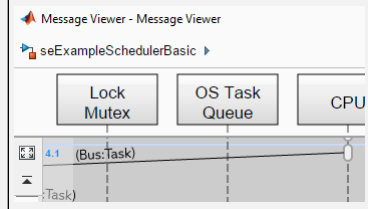
More About

- “Interface for Custom Visualization” on page 10-2
- “Access Property Values” (MATLAB)

Migrating SimEvents Models

Migration Considerations

To take advantage of SimEvents features, migrate legacy SimEvents models (pre-R2016a). Benefits include:

Event actions 	MATLAB Discrete-Event System block 	Discrete-Event Chart block 
Entity multicast 	Domain transitions 	Simulink integration 
Unified entity type 	Entity Batch Creator and Splitter blocks 	Sequence Viewer 

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.

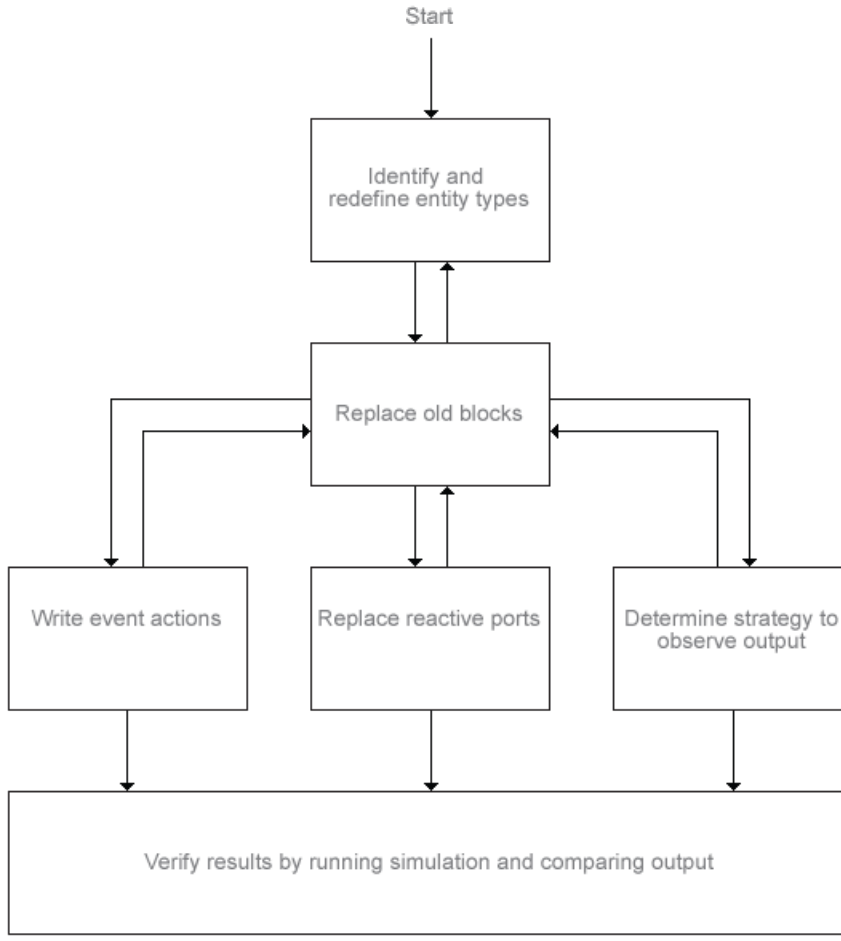
See Also

More About

- “Migration Workflow” on page 11-4
- “Identify and Redefine Entity Types” on page 11-7
- “Replace Old Blocks” on page 11-9
- “Connect Signal Ports” on page 11-13
- “Write Event Actions for Legacy Models” on page 11-19
- “Observe Output” on page 11-32
- “Reactive Ports” on page 11-34

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” (Simulink)).

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 11-7)
3. When possible, replace old blocks with new blocks (“Replace Old Blocks” on page 11-9) 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 11-19)
   b. Replace Get Attribute blocks with event actions in other blocks (“Connect Signal Ports” on page 11-13)
   c. Replace Attribute Function blocks with event actions in other blocks (“Replace Attribute Function Blocks with Event Actions” on page 11-26)
   d. Replace random number generators with event actions in other blocks (“Generate Random Numbers with Event Actions” on page 11-21)
5. Replace reactive ports (see “If Connected to Reactive Ports” on page 11-16).
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 11-32).
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 11-7
- “Replace Old Blocks” on page 11-9
- “Connect Signal Ports” on page 11-13
- “Write Event Actions for Legacy Models” on page 11-19
- “Observe Output” on page 11-32

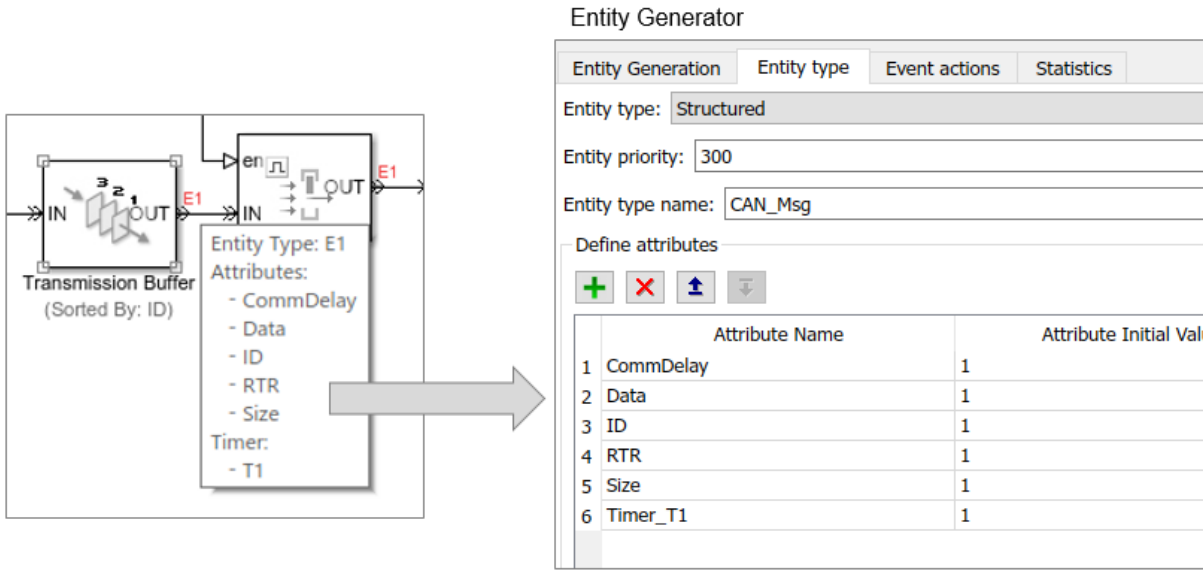
Identify and Redefine Entity Types

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 **Display** menu, select **Signals & Ports** > **Port 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 11-9
- “Connect Signal Ports” on page 11-13
- “Write Event Actions for Legacy Models” on page 11-19
- “Observe Output” on page 11-32
- “Reactive Ports” on page 11-34

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 11-26.
Attribute Scope	Wait until “If Using Get Attribute Blocks to Observe Output” on page 11-13.
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 11-32.
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 11-19.
Entity Departure Function-Call Generator	Wait until “Write Event Actions for Legacy Models” on page 11-19.
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 11-19.
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).

Old Block	Action for New SimEvents Model
Event-Based Entity Generator	Replace with Entity Generator.
Event-Based Random Number	Wait until “Generate Random Numbers with Event Actions” on page 11-21.
Event-Based Sequence	Wait until “Write Event Actions for Legacy Models” on page 11-19.
FIFO Queue	Replace with Entity Queue.
Get Attribute	Wait until “Connect Signal Ports” on page 11-13.
Infinite Server	Replace with Entity Server.
Initial Value	Delete (block no longer needed).
Input Switch	Replace with Entity Input Switch.
Instantaneous Entity Counting Scope	Wait until “If Using Get Attribute Blocks to Observe Output” on page 11-13.
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 1-49.
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 11-19.

Old Block	Action for New SimEvents Model
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 11-16.
Signal-Based Function-Call Generator	Wait until “If Connected to Reactive Ports” on page 11-16.
Single Server	Replace with Entity Server.
Start Timer	For an example, see “Measure Point-to-Point Delays” on page 1-49.
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 11-14.
X-Y Signal Scope	Simulink XY Graph.

When done, return to “Migration Workflow” on page 11-4.

See Also

More About

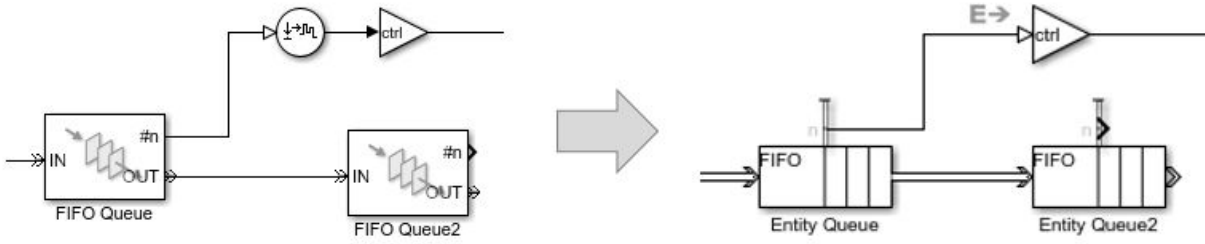
- “Migration Considerations” on page 11-2
- “Migration Workflow” on page 11-4
- “Identify and Redefine Entity Types” on page 11-7
- “Connect Signal Ports” on page 11-13
- “Write Event Actions for Legacy Models” on page 11-19
- “Observe Output” on page 11-32

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

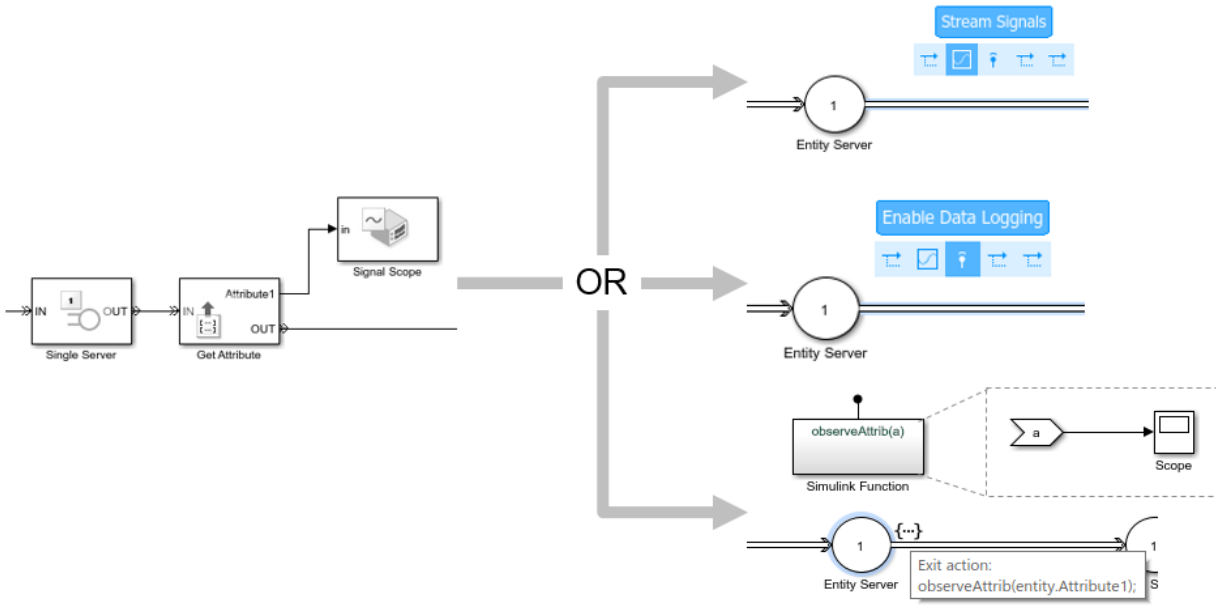
SimEvents models no longer require gateway blocks. Remove all gateway blocks, as shown in the figure:



Return to “Connect Signal Ports” on page 11-13.

If Using Get Attribute Blocks to Observe Output

If you use Get Attribute blocks to observe output, see “Observe Output” on page 11-32. 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.



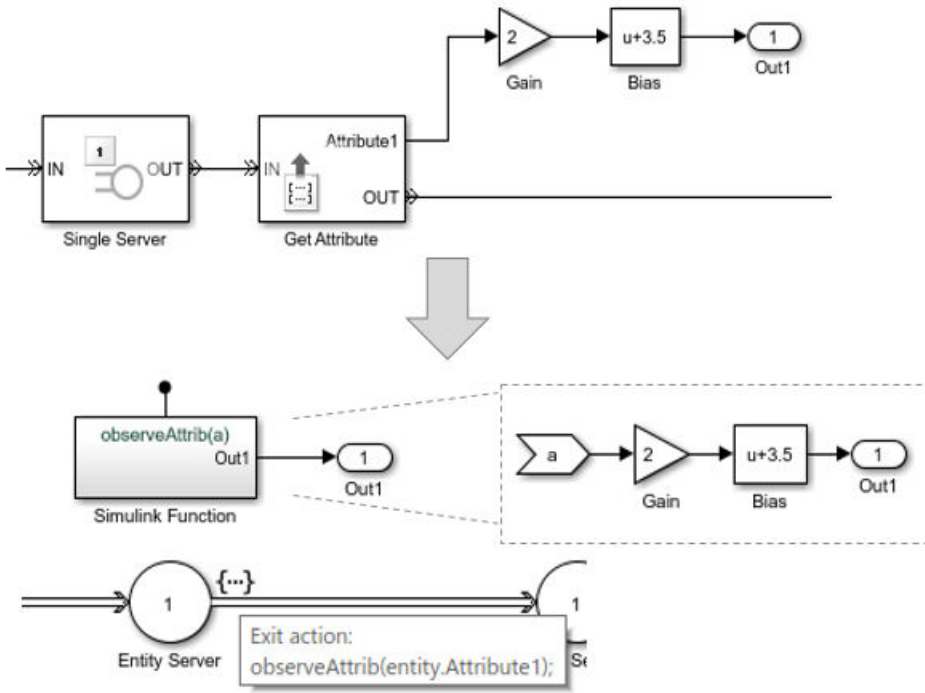
Return to “Connect Signal Ports” on page 11-13.

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.

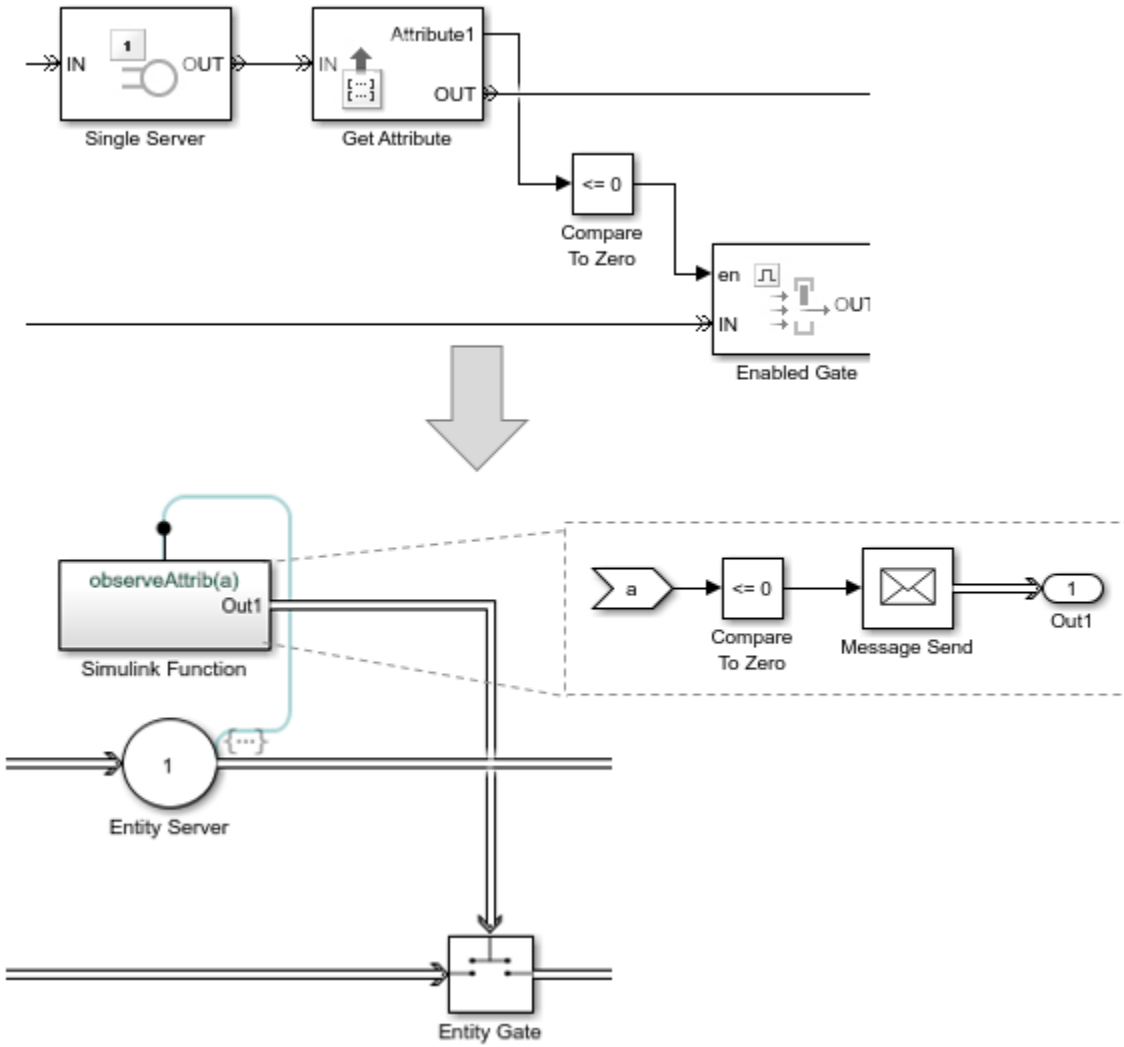
Statistics Port	Updated on Event			
	Entry	Exit	Blocked	Preempted
Number of entities departed, d		✓		
Number of entities in block, n	✓			
Number of entities arrived, a	✓			
Pending entity present in block, pe		✓	✓	✓

Statistics Port	Updated on Event			
	Entry	Exit	Blocked	Preempted
Number of pending entities, np		✓	✓	✓
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 11-13.

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 11-34.

Return to “Connect Signal Ports” on page 11-13.

See Also

More About

- “Migration Considerations” on page 11-2
- “Migration Workflow” on page 11-4
- “Identify and Redefine Entity Types” on page 11-7
- “Replace Old Blocks” on page 11-9
- “Write Event Actions for Legacy Models” on page 11-19
- “Observe Output” on page 11-32
- “Reactive Ports” on page 11-34

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

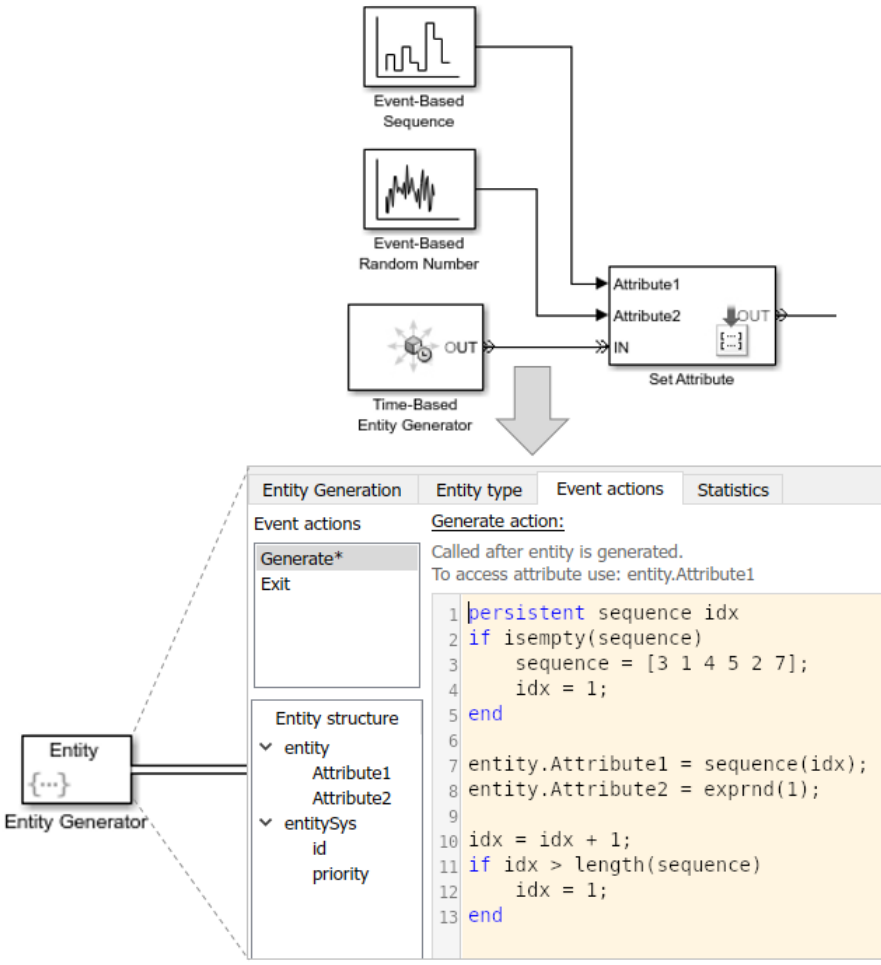
Use these guidelines to replace Set Attribute blocks:

- 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.

Generate Random Numbers with Event Actions

You can generate random numbers using:

Random Number Distribution

Replace Event-Based Random Number block random number distribution modes with equivalent MATLAB code in event actions.

To reproduce these distributions in a SimEvents model, use code like those in the **Usage** column of this table in event actions or intergeneration time actions in the Entity Generator block.

Distribution	Parameters	Usage	Requires Statistics and Machine Learning Toolbox Product
Exponential	Mean (m)	`-m * log(1-rand)`	No
Uniform	Minimum (m) 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) Number of trials (N)	`binornd(N,P)`	Yes

Distribution	Parameters	Usage	Requires Statistics and Machine Learning Toolbox Product

Triangular	Minimum (m)
	Maximum (M)
	Mode (mode)
		`persistent pd`
		`if isempty(pd)`
		`  pd = makedist('Triangular',...`
		`  'a',m,'b',mode,'c',M)`
		`end`
		`random(pd)`
			Yes

Gamma	Threshold (T) Scale (a) Shape (b)	`gamrnd(b,a)`	Yes
Gaussian (normal)	Mean (m) 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) Mu (mu) Sigma (S)	`T + lognrnd(mu,S)`	Yes

Log-logistic	Threshold (T)
	Scale (a)
		`persistent pd`
		`if isempty(pd)`
		`  pd = makedist('Loglogistic',...`
		`  'mu',m,'sigma',S);`
		`end`
		`random(pd)`
			Yes

Distribution	Parameters	Usage	Requires Statistics and Machine Learning Toolbox Product
Beta	Minimum (m) Maximum (M) Shape parameter a (a) Shape parameter b (b)	`betarnd(a,b)`	Yes

Discrete uniform	Minimum (m)
	
	Maximum (M)
	
	Number of values (N)
		`persistent V P`
		`if isempty(V)`
		`  step = (M-m)/N;`
		`  V = m : step : M;`
		`  P = 0 : 1/N : N;`
		`end`
		`r = rand;`
		`idx = find(r < P, 1);`
		`V(idx)`
			No

Weibull	Threshold (T) Scale (a) Shape (b)	`T + wblrnd(a,b)`	Yes

Arbitrary continuous	Value vector (V)
	
	Cumulative probability function vector (P)
		`r = rand;`
		`if r == 0`
		`  val = V(1);`
		`else`
		`  idx = find(r < P,1);`
		`  val = V(idx-1) + ...`
		`  (V(idx)-V(idx-1))*(r-P(idx-1));`
		`end`
			No

Distribution	Parameters	Usage	Requires Statistics and Machine Learning Toolbox Product

Arbitrary discrete	Value vector (V)
	
	Probability vector (P)
		`r = rand;`
		`idx = find(r < cumsum(P),1);`
		`V(idx)`
			No

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.

Seeds for Random Number Generation

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

```
persistent init
if isempty(init)
        rng(12234);
        init=true;
end
```

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:

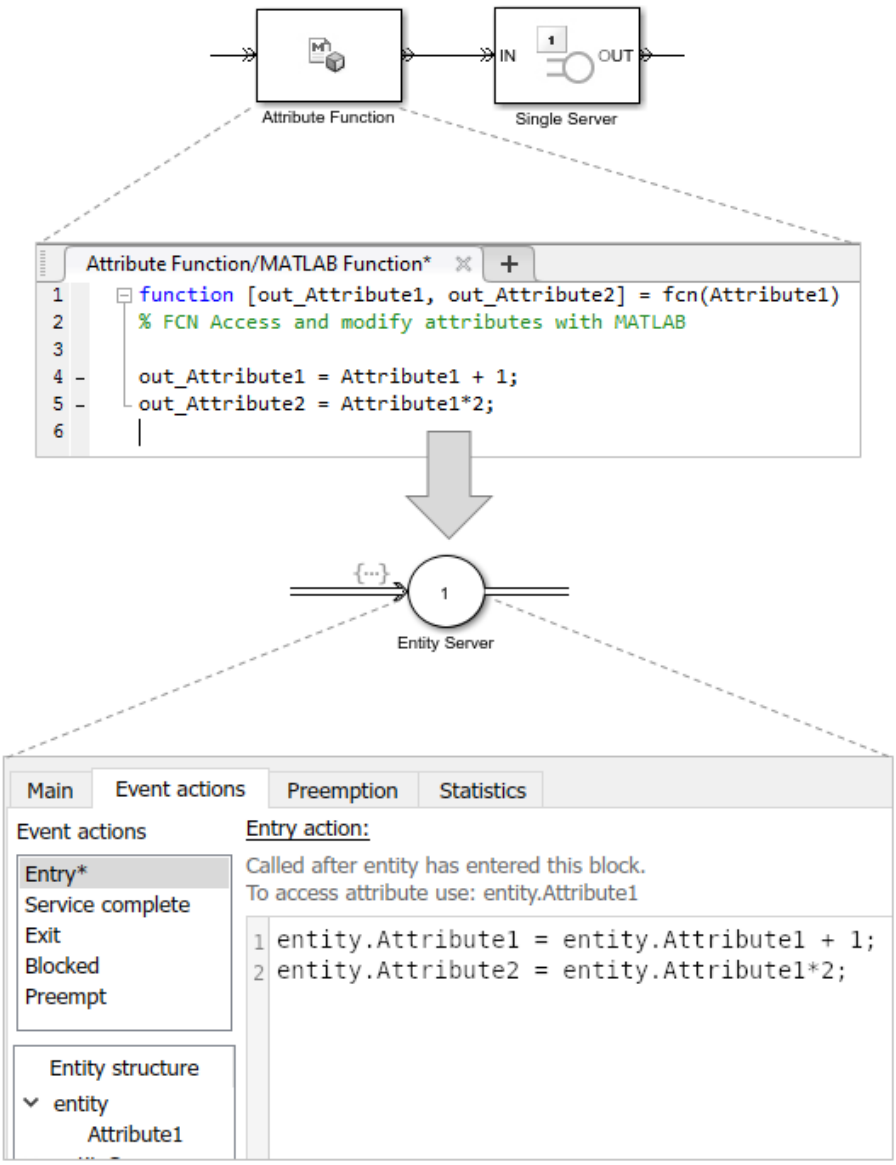
```
persistent sequence idx
if isempty(sequence)
    sequence = [3 1 4 5 2 7];
    idx = 1;
end

entity.Attribute1 = sequence(idx);

idx = idx + 1;
if idx > length(sequence)
    idx = 1;
end
```

Replace Attribute Function Blocks with Event Actions

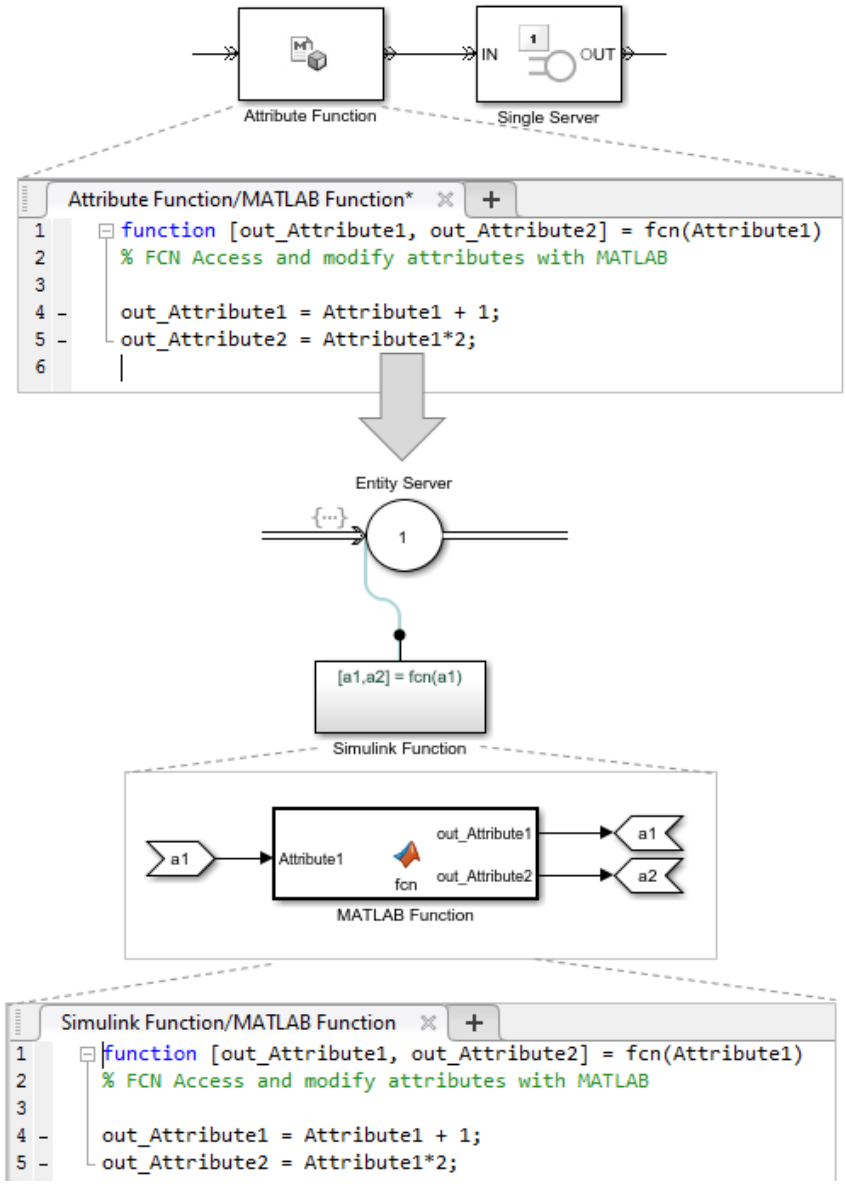
Replace Attribute Function blocks, which manipulate attributes using MATLAB code, with event actions:

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:


Attribute Function
IN
1
OUT
Single Server
Attribute Function/MATLAB Function*
1 function [out_Attribute1, out_Attribute2] = fcn(Attribute1)
2 % FCN Access and modify attributes with MATLAB
3
4 - out_Attribute1 = Attribute1 + 1;
5 - out_Attribute2 = Attribute1*2;
6
{...}
1
Entity Server
Main
Event actions
Preemption
Statistics
Event actions
Entry*
Service complete
Exit
Blocked
Preempt
Entity structure
entity
Attribute1
Entry action:
Called after entity has entered this block.
To access attribute use: entity.Attribute1
1 entity.Attribute1 = entity.Attribute1 + 1;
2 entity.Attribute2 = entity.Attribute1*2;

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.



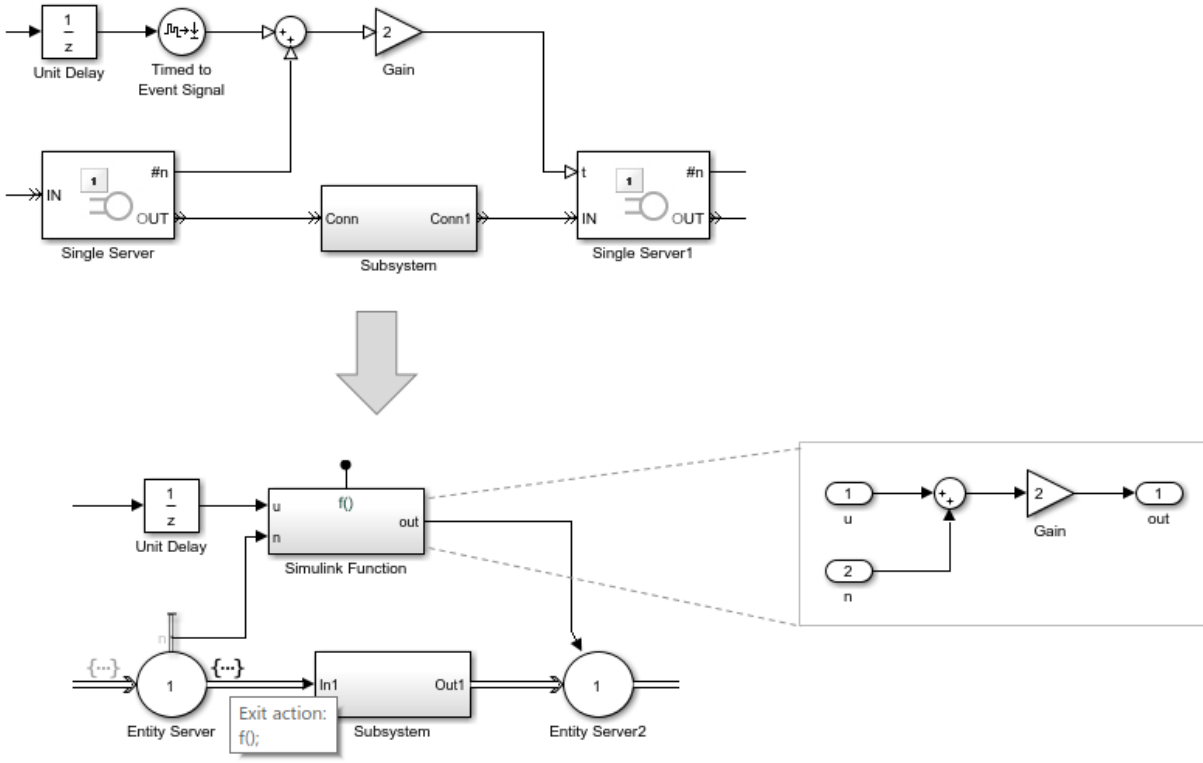
Return to “Migration Workflow” on page 11-4.

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 11-7
- “Replace Old Blocks” on page 11-9
- “Connect Signal Ports” on page 11-13
- “Observe Output” on page 11-32
- “Reactive Ports” on page 11-34

Observe Output

Use these methods to observe output from your SimEvents model:

Items to Observe	Visualization Tool
Statistics	• Simulation Data Inspector • Simulink To Workspace block • Simulink Scope block • Simulink Display block • Simulink To File block • Simulink Dashboard blocks
Entities passing through model	
Attributes	
Count simultaneous entities and messages	Simulation Data Inspector
Count simultaneous events	Simulation Data Inspector — Each event is now a message reactive port
Entities moving through blocks in the model	Sequence Viewer
Entity animation	**Display > Message Animation**
Step through Simulation	Simulink Simulation Stepper
Custom animation	SimEvents custom visualization API.

Return to “Migration Workflow” on page 11-4.

See Also

More About

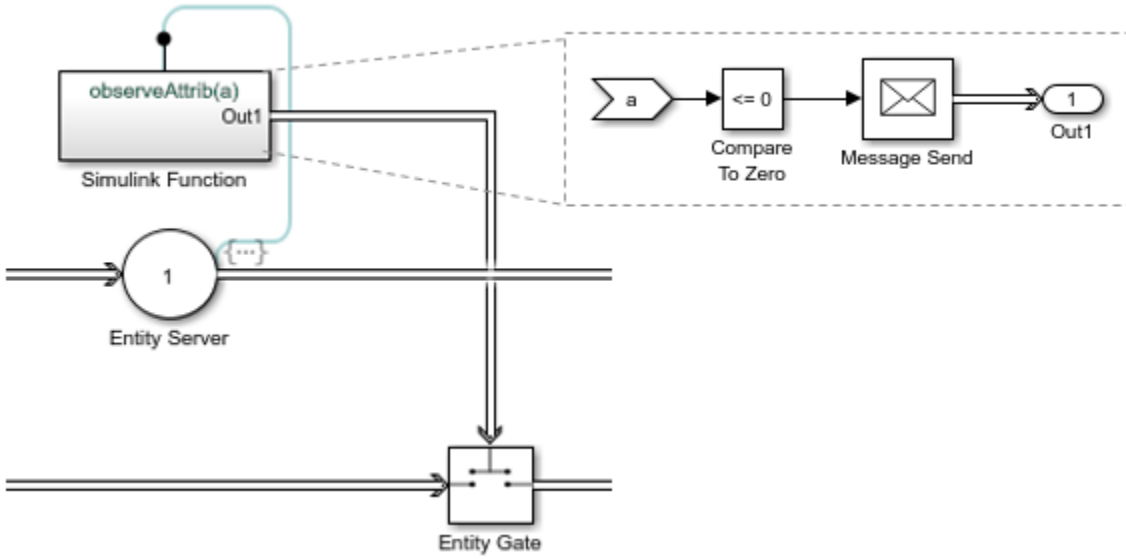
- “Migration Considerations” on page 11-2
- “Migration Workflow” on page 11-4
- “Identify and Redefine Entity Types” on page 11-7
- “Replace Old Blocks” on page 11-9
- “Connect Signal Ports” on page 11-13

- “Write Event Actions for Legacy Models” on page 11-19
- “Reactive Ports” on page 11-34

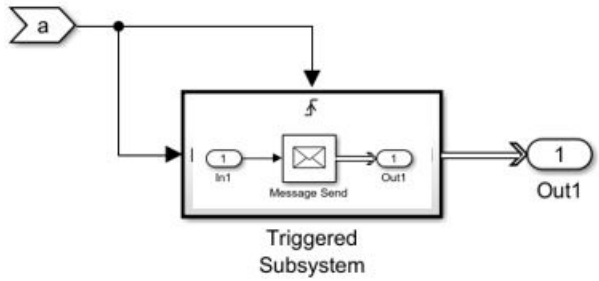
Reactive Ports

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.

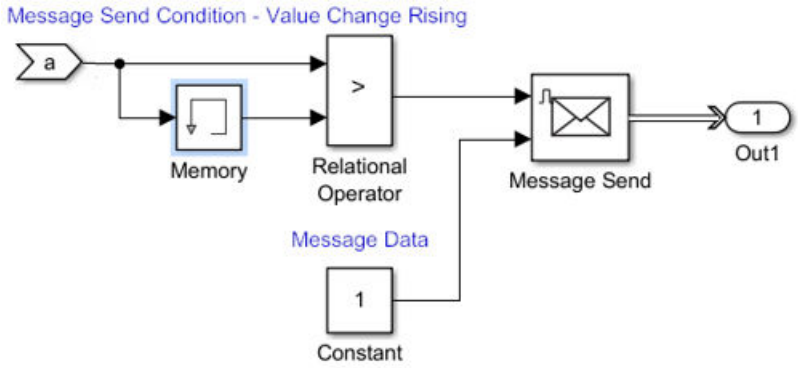
Here is an example of sending a message when data is less than or equal to 0.



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. In receive mode, send a message to advance one entity for each message that arrives on the control port.
Entity Input Switch 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 11-7
- “Replace Old Blocks” on page 11-9
- “Connect Signal Ports” on page 11-13
- “Write Event Actions for Legacy Models” on page 11-19
- “Observe Output” on page 11-32

Troubleshoot SimEvents Models

Which Debugging Tool to Use

To decide which observation tool to use:

Items to Observe	Visualization Tool
Statistics	• SimEvents Debugger • SimEvents Entity Inspector • Simulation Data Inspector • Simulink To Workspace block • Simulink Scope block • Simulink Display block • Simulink To File block • Simulink dashboard blocks
Entities passing through model	
Entity animation	**Display > Message Animation**
Step through simulation	• SimEvents Debugger • Simulink Simulation Stepper
Custom animation	Use SimEvents custom visualization API.

See Also

SimEvents Debugger

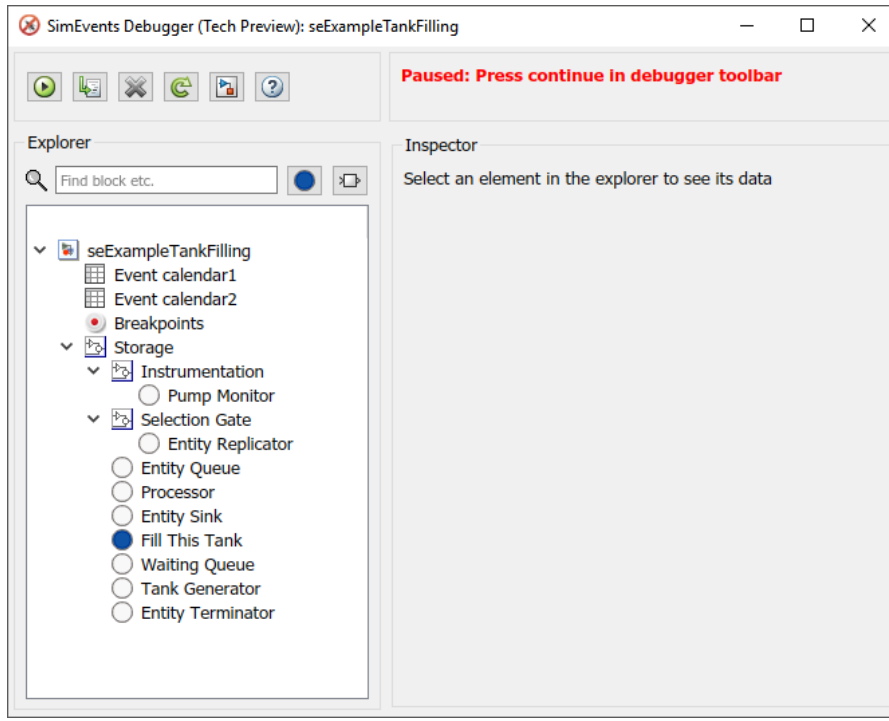
Related Examples

- “Debug SimEvents Models” on page 12-3

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 Editor, 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 “Interface for Custom Visualization” on page 10-2.

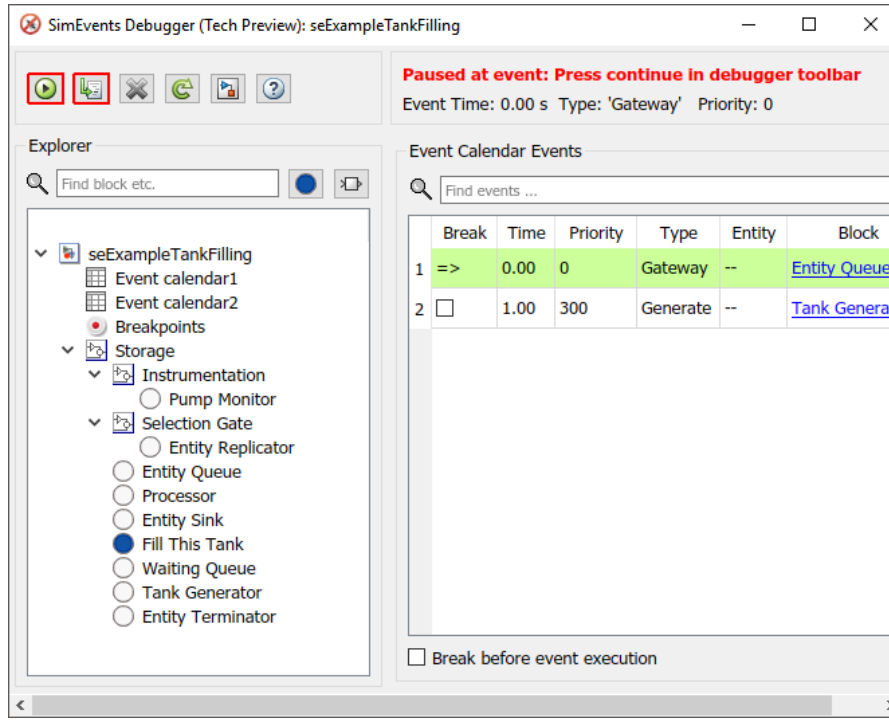
Start the Debugger

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  .

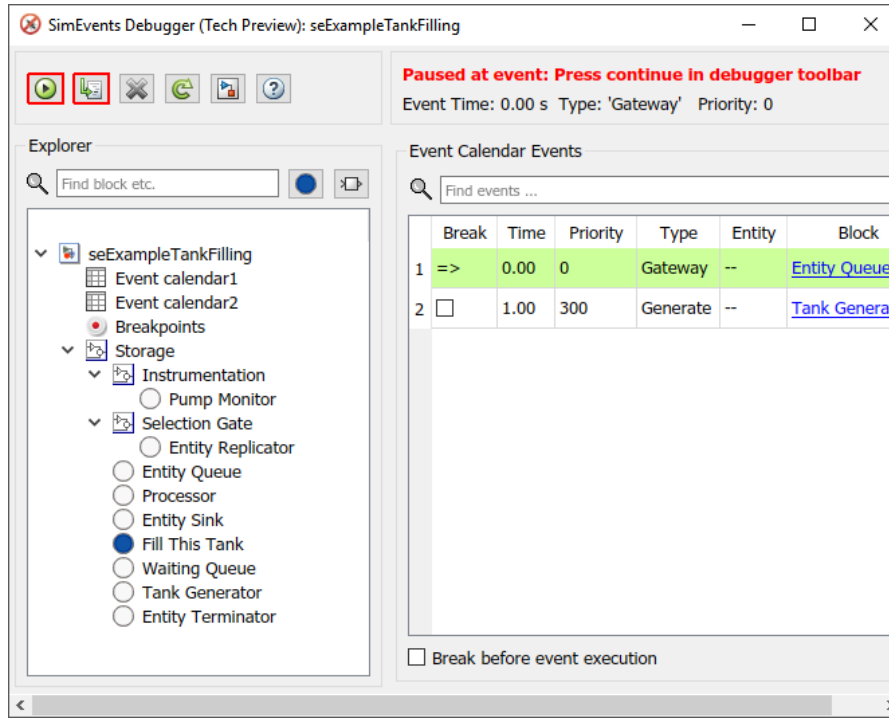
   Note You can also click **Continue** () to have the debugger continue the simulation. However, doing so without setting breakpoints causes the simulation to complete and the debugger to close.

   ---
5. The debugger pauses at the next event and displays it in the event calendar. The current event is highlighted in green.

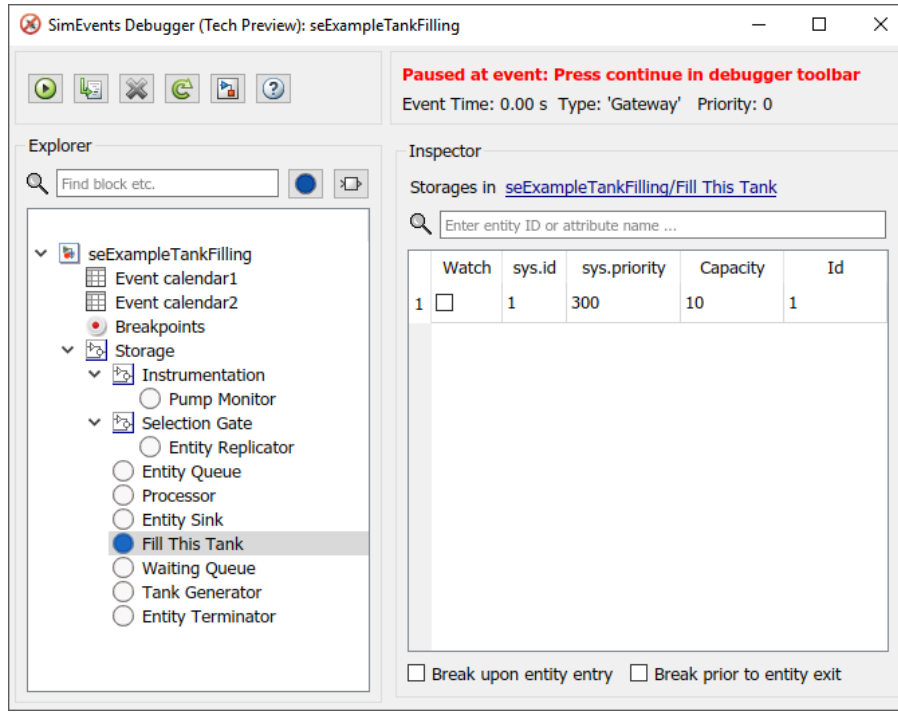


Step Through Model

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.

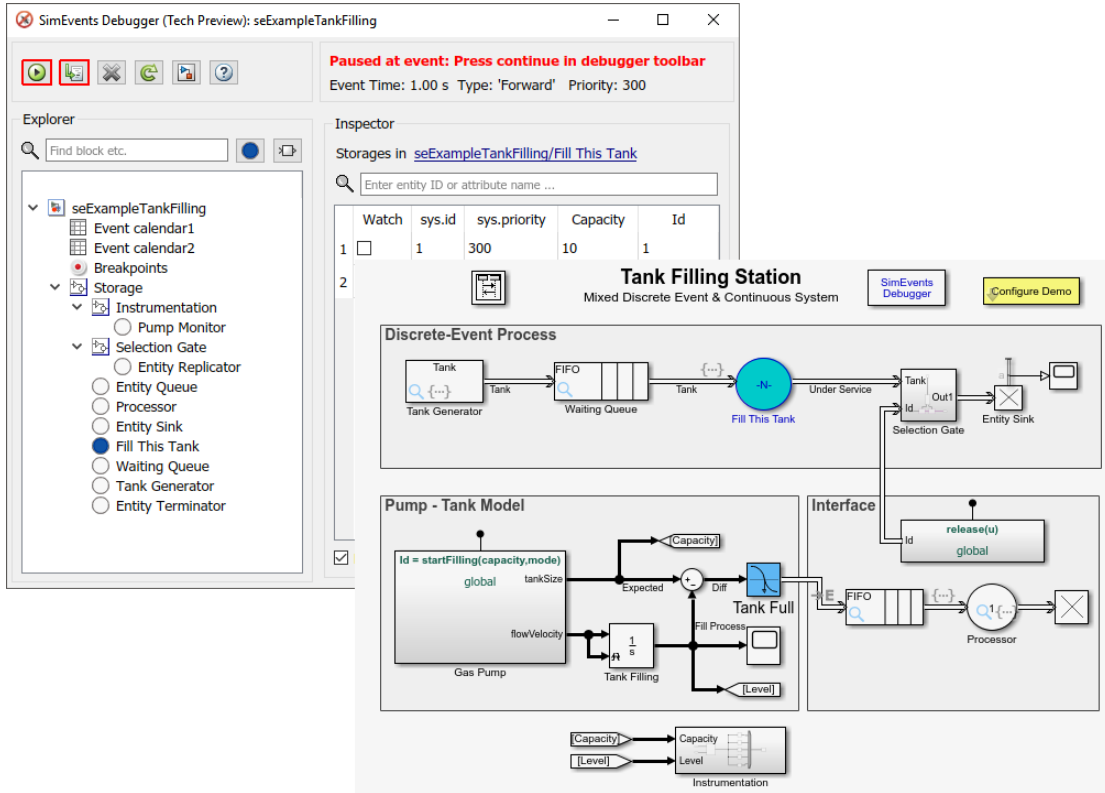
4 To set breakpoints for when this entity enters and leaves the block, at the bottom of the **Inspector** pane, select the two check boxes **Break upon entity entry** and **Break prior to entity exit**.

   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.

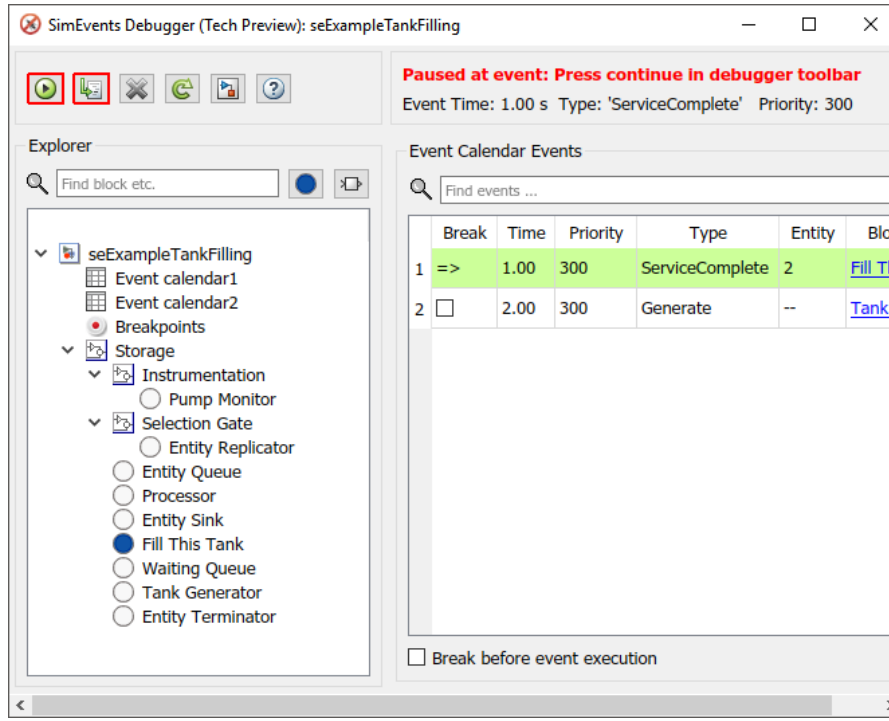
5 To progress to the next event, click  .

6 Click **Continue**. Simulation continues until the next `PostEntry` or `PreExit` event.



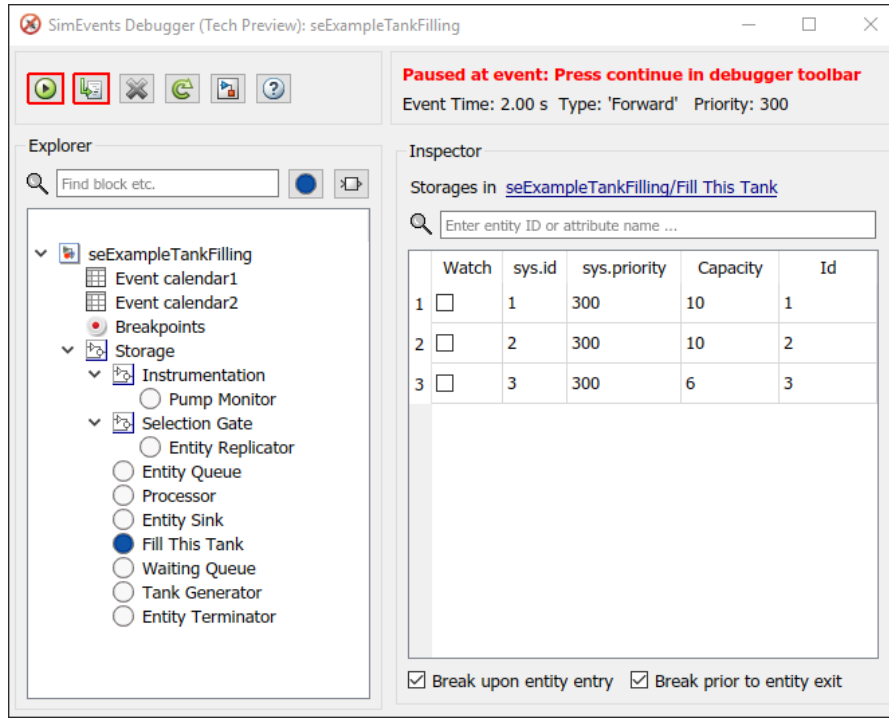
The block associated with the breakpoint is highlighted.

7 Step to the next event.



The next breakpoint at which the debugger stops is highlighted in the event calendar.

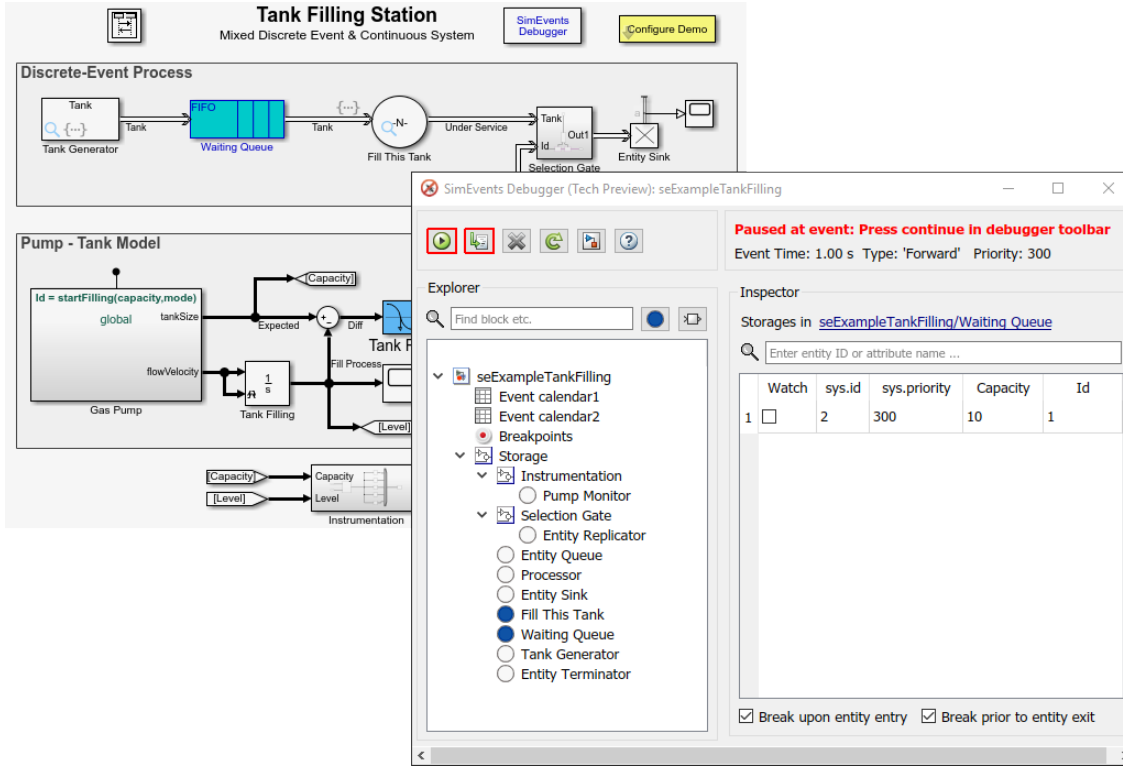
8 Continue the simulation.



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

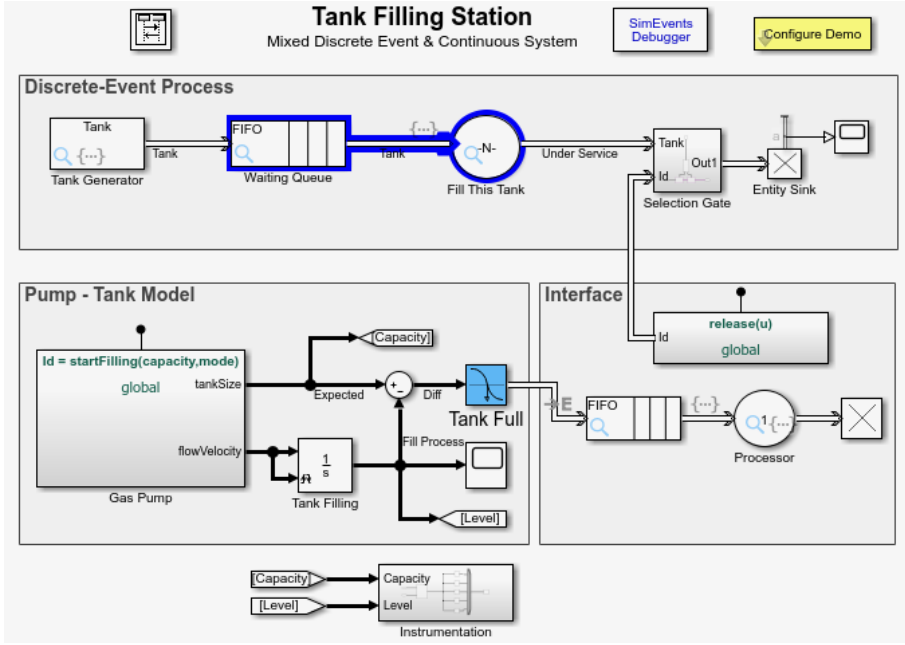
SimEvents Debugger

More About

- “Which Debugging Tool to Use” on page 12-2
- “Create Custom Visualization”
- “Interface for Custom Visualization” on page 10-2

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 the Simulink editor, select **Display > SimEvents Animation Menu**, then select one of the animation speeds.



See Also

More About

- “Which Debugging Tool to Use” on page 12-2
- “Visualize and Animate Simulations”