

**SimEvents®**

Reference



**MATLAB® & SIMULINK®**

R2023a



## How to Contact MathWorks



Latest news: [www.mathworks.com](http://www.mathworks.com)  
Sales and services: [www.mathworks.com/sales\\_and\\_services](http://www.mathworks.com/sales_and_services)  
User community: [www.mathworks.com/matlabcentral](http://www.mathworks.com/matlabcentral)  
Technical support: [www.mathworks.com/support/contact\\_us](http://www.mathworks.com/support/contact_us)



Phone: 508-647-7000



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

*SimEvents® Reference*

© COPYRIGHT 2005–2023 by The MathWorks, Inc.

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

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

### Trademarks

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

### Patents

MathWorks products are protected by one or more U.S. patents. Please see [www.mathworks.com/patents](http://www.mathworks.com/patents) for more information.

## Revision History

March 2007	Online only	Revised for Version 2.0 (Release 2007a). Previously part of <i>SimEvents® User's Guide</i> .
September 2007	Online only	Revised for Version 2.1 (Release 2007b)
March 2008	Online only	Revised for Version 2.2 (Release 2008a)
October 2008	Online only	Revised for Version 2.3 (Release 2008b)
March 2009	Online only	Revised for Version 2.4 (Release 2009a)
September 2009	Online only	Revised for Version 3.0 (Release 2009b)
March 2010	Online only	Revised for Version 3.1 (Release 2010a)
September 2010	Online only	Revised for Version 3.1.1 (Release 2010b)
April 2011	Online only	Revised for Version 3.1.2 (Release 2011a)
September 2011	Online only	Revised for Version 4.0 (Release 2011b)
March 2012	Online only	Revised for Version 4.1 (Release 2012a)
September 2012	Online only	Revised for Version 4.2 (Release 2012b)
March 2013	Online only	Revised for Version 4.3 (Release 2013a)
September 2013	Online only	Revised for Version 4.3.1 (Release 2013b)
March 2014	Online only	Revised for Version 4.3.2 (Release 2014a)
October 2014	Online only	Revised for Version 4.3.3 (Release 2014b)
March 2015	Online only	Revised for Version 4.4 (Release 2015a)
September 2015	Online only	Revised for Version 4.4.1 (Release 2015b)
March 2016	Online only	Revised for Version 5.0 (Release 2016a)
September 2016	Online only	Revised for Version 5.1 (Release 2016b)
March 2017	Online only	Revised for Version 5.2 (Release 2017a)
September 2017	Online only	Revised for Version 5.3 (Release 2017b)
March 2018	Online only	Revised for Version 5.4 (Release 2018a)
September 2018	Online only	Revised for Version 5.5 (Release 2018b)
March 2019	Online only	Revised for Version 5.6 (Release 2019a)
September 2019	Online only	Revised for Version 5.7 (Release 2019b)
March 2020	Online only	Revised for Version 5.8 (Release 2020a)
September 2020	Online only	Revised for Version 5.9 (Release 2020b)
March 2021	Online only	Revised for Version 5.10 (Release 2021a)
September 2021	Online only	Revised for Version 5.11 (Release 2021b)
March 2022	Online only	Revised for Version 5.12 (Release 2022a)
September 2022	Online only	Revised for Version 5.13 (Release 2022b)
March 2023	Online only	Revised for Version 5.14 (Release 2023a)



## 1 **Functions**

## 2 **Blocks**

## 3 **Configuration Parameters**

<b>SimEvents Pane</b> .....	<b>3-2</b>
SimEvents Pane Overview .....	3-2
Execution order .....	3-2
Seed for event randomization .....	3-3
Maximum events per block .....	3-4
Maximum events per model .....	3-4
Prevent duplicate events on multiport blocks and branched signals .....	3-5
<b>SimEvents Diagnostics Pane</b> .....	<b>3-6</b>
Diagnostics Pane Overview .....	3-6
Attribute output delayed relative to entities .....	3-7
Response to function call delayed relative to entities .....	3-8
Statistical output delayed relative to entities .....	3-9
Modification of attribute values used for decision making .....	3-10
Identical seeds for random number generators .....	3-11

## 4 **Upgrade Advisor Checks**

<b>SimEvents Upgrade Advisor Checks</b> .....	<b>4-2</b>
Checks Overview .....	4-2
Check for implicit event duplication caused by SimEvents blocks .....	4-2



# Functions

---

## matlab.DiscreteEventSystem class

**Package:** matlab

**Superclasses:** matlab.System

Base class for discrete-event system objects

### Description

`matlab.DiscreteEventSystem` is the base class for discrete-event System objects. In your class definition file, you must subclass your object from this base class (or from another class that derives from this base class). Subclassing allows you to use the implementation and service methods provided by this base class to build your object. For more information about implementing `matlab.DiscreteEventSystem` class with MATLAB Discrete-Event System block, see “Create Custom Blocks Using MATLAB Discrete-Event System Block”.

Type this syntax as the first line of your class definition file to directly inherit from the `matlab.DiscreteEventSystem` base class, where `ObjectName` is the name of your object:

```
classdef ObjectName < matlab.DiscreteEventSystem
```

For more information about implementing a discrete-event System object™, see “Create a Discrete-Event System Object”. For information about linking the discrete-event System object to a SimEvents model and creating a custom behavior, see “Delay Entities with a Custom Entity Storage Block”.

The `matlab.DiscreteEventSystem` class is a `handle` class.

### Class Attributes

Abstract	false
HandleCompatible	true
StrictDefaults	false

For information on class attributes, see “Class Attributes”.

### Methods

#### Public Methods

<code>entityType</code>	Define entity type
<code>blocked</code>	Event action when entity forward fails
<code>destroy</code>	Event action upon entity destruction
<code>entry</code>	Event action when entity enters storage element
<code>exit</code>	Event action before entity exit from storage
<code>generate</code>	Event action upon entity creation
<code>iterate</code>	Event action when entity iterates
<code>modified</code>	Event action upon entity modification by the Entity Find block
<code>resourceAcquired</code>	Event action upon successful resource acquisition
<code>resourceReleased</code>	Event action upon successful resource release
<code>testEntry</code>	Event action to accept or refuse entity
<code>timer</code>	Event action when timer completes
<code>setupEvents</code>	Initialize entity generation events



queueFIFO	Define first-in first-out (FIFO) queue storage
queueLIFO	Define last-in last-out (LIFO) stack storage
queuePriority	Define priority queue storage
queueSysPriority	Define system priority queue storage
resourceSpecification	Create specifications for a resource acquisition or a resource release event
resourceType	Specify an entity type and the name of the resources to be acquired by the specified entity
initResourceArray	Initialize a resource specification array
eventAcquireResource	Create a resource acquisition event
eventDestroy	Create entity destroy event
eventForward	Create entity forward event
eventGenerate	Create entity generate event
eventIterate	Create entity iterate event
eventReleaseResource	Create an event to release previously acquired resources
eventReleaseAllResources	Create an event to release all resources acquired by an entity
eventTestEntry	Create an event to indicate that the acceptance policy for the storage has changed and the storage retests arriving entities
eventTimer	Create entity timer event
cancelAcquireResource	Cancel previously scheduled resource acquisition event
cancelDestroy	Cancel previously scheduled entity destroy event
cancelForward	Cancel previously scheduled forward events
cancelGenerate	Cancel previously scheduled entity generation event
cancelIterate	Cancel previously scheduled iterate event
cancelTimer	Cancel previously scheduled timer event

### Protected Methods

initEventArray	Initialize event array
getEntityPortsImpl	Define input ports and output ports of discrete-event system
getEntityStorageImpl	Define entity storage elements of discrete-event system
getEntityTypesImpl	Define entity types of discrete-event system
getResourceNamesImpl	Define resource pools from which to acquire resources

## Examples

### Create a Custom Entity Storage Block to Delay Entities

This example shows how to use discrete-event System object methods to create a custom entity storage block that has one input port, one output port, and one storage element. The discrete-event System object is the instantiation of the `matlab.DiscreteEventSystem` class, which allows you to use the implementation and service methods provided by this class. Then, you use the MATLAB Discrete-Event System block to integrate the System object into a SimEvents model. The custom MATLAB Discrete-Event System block accepts an entity from its input port and forwards it to its output port with a specified delay. For more information, see “Delay Entities with a Custom Entity Storage Block”.

```
classdef CustomEntityStorageBlock < matlab.DiscreteEventSystem
    % A custom entity storage block with one input, one output, and one storage.

    % Nontunable properties
    properties (Nontunable)
        % Capacity
        Capacity = 1;
    end
end
```

```
% Delay
    Delay=4;
end

methods (Access=protected)
    function num = getNumInputsImpl(~)
        num = 1;
    end

    function num = getNumOutputsImpl(~)
        num = 1;
    end

    function entityType = getEntityTypesImpl(obj)
        entityType = obj.entityType('Car');
    end

    function [inputTypes,outputTypes] = getEntityPortsImpl(obj)
        inputTypes = {'Car'};
        outputTypes = {'Car'};
    end

    function [storageSpecs, I, O] = getEntityStorageImpl(obj)
        storageSpecs = obj.queueFIFO('Car', obj.Capacity);
        I = 1;
        O = 1;
    end

end

methods

    function [entity,event] = CarEntry(obj,storage,entity,source)
        % Specify event actions when entity enters storage.

        event = obj.eventForward('output', 1, obj.Delay);
    end

end

end
```

## Version History

Introduced in R2016a

### See Also

matlab.System | entityType | entry | eventForward | eventGenerate |  
getEntityStorageImpl | queueFIFO

### Topics

“Create Custom Blocks Using MATLAB Discrete-Event System Block”

“Create a Custom Entity Storage Block with Iteration Event”

“Create a Discrete-Event System Object”

“Custom Entity Storage Block with Multiple Timer Events”

Class Attributes

Property Attributes

# blocked

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Event action when entity forward fails

## Syntax

```
[entity,events]=blocked(obj,storage,entity,destination)
[entity,events,out1,...]=blocked(obj,storage,entity,destination,in1,...)
```

## Description

[entity,events]=blocked(obj,storage,entity,destination) specifies event actions of the object when an entity forward fails because the destination storage element has reached its maximum capacity.

[entity,events,out1,...]=blocked(obj,storage,entity,destination,in1,...) specifies such event actions of the object when the block has one or more input signal ports and/or signal output ports.

## Input Arguments

**obj** — Discrete-event System object

MATLAB® object

Discrete-event System object.

**storage** — Storage

double

Index of the storage element.

**entity** — Entity

MATLAB structure

Entity leaving storage element. Entity has these fields:

- **sys** (MATLAB structure) — It has these fields:
  - **id** (double) — Entity ID
  - **priority** (double) — Entity priority
- **data** — Entity data

**destination** — Destination

MATLAB structure

Destination of entity, such as an output port or a storage element. It has these fields:

- `type` (character vector) — Specify output or storage
- `index` (double) — Output or storage index

**in1 — Signal inputs**

any value

Any data inputs of the object. These input arguments exist only when the object has data inputs.

**Output Arguments****entity — Entity**

MATLAB structure

Entity leaving storage, possibly with changed data.

**events — Events**

vector of MATLAB structures

Events to be scheduled after the method returns. Use `matlab.DiscreteEventSystem` class methods to create events. Each event has these fields:

- `type` (character vector) — Type of the event
- `delay` (double) — Delay before the event
- `priority` (double) — Priority of the event
- `Storage` (double) — Index of the storage element
- `tag` (character vector) — Event tag
- `location` (MATLAB structure) — Source or destination location of entity

**out1 — Signal output**

any value

Data outputs of the object. You must specify these output arguments when the object has data outputs.

**Examples****Cancel Current Forward Event**

Cancel the current forward event upon blocking. Schedule an event to forward the entity to the next location. Destroy the entity if no storage can accept the entity.

```
function [entity,events] = blocked(obj,storage,entity,dst)
    % Cancel the current forward event. Schedule an event to
    % forward the entity to the next location. Destroy the entity
    % if no storage can accept the entity.
    if dst.index < obj.numStorage
        events = [...
            obj.cancelForward(dst.type, dst.index), ...
            obj.eventForward('storage', dst.index+1, 0)];
    else
        events = [...
```

```
        obj.cancelForward(dst.type, dst.index), ...  
        obj.eventDestroy()];  
    end  
end
```

## Version History

Introduced in R2016a

### See Also

matlab.DiscreteEventSystem | destroy | entry | exit | generate | getEntityPortsImpl |  
getEntityStorageImpl | getEntityTypesImpl | iterate | setupEvents | timer

### Topics

“Create Custom Blocks Using MATLAB Discrete-Event System Block”

# cancelAcquireResource

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Cancel previously scheduled resource acquisition event

## Syntax

```
event = cancelAcquireResource(tag)
```

## Description

`event = cancelAcquireResource(tag)` cancels a previously scheduled resource acquisition event.

## Input Arguments

**tag** — Tag of the previously scheduled resource acquisition event

character vector

Tag of the previously scheduled resource acquisition event to be canceled.

## Output Arguments

**event** — Cancellation Event

MATLAB structure

Event for canceling the previously scheduled resource acquisition.

## Examples

### Cancel a Resource Acquisition Event

Cancel a resource acquisition event when the previously scheduled resource acquisition event times out.

```
function [entity,events] = timer(obj,storage,entity,tag)
    % Assume that eventTimer() defines a timer with tag and that there is a previously
    % scheduled resource acquisition event with 'loadingWorker' tag.
    event = obj.cancelAcquireResource('loadingWorker');
    % This cancels the 'loadingWorker' acquisition event when the timer finishes.
end
```

## Version History

Introduced in R2019a

## **See Also**

`matlab.DiscreteEventSystem` | `eventForward` | `getResourceNamesImpl` | `resourceReleased` | `eventReleaseResource`

## **Topics**

“Create the Discrete-Event System Object with Multiple Timer Events”

“Create Custom Blocks Using MATLAB Discrete-Event System Block”

# cancelDestroy

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Cancel previously scheduled entity destroy event

## Syntax

```
event=cancelDestroy()
```

## Description

`event=cancelDestroy()` cancels a previously scheduled destroy event of the current entity. You can then schedule this event by returning it as the output argument when implementing an event action method, such as `entry` or `exit`.

## Output Arguments

### **event** — Event

MATLAB structure

Event for cancelling entity destroy.

## Examples

### Cancel Previously Scheduled Destroy Event

Cancel the previously scheduled destroy event of the entity in the current event action context.

```
function [entity,events] = timer(obj,storage,entity,tag)
    % Cancel the previously scheduled destroy event of the entity in
    % current event action context.
    event = obj.cancelDestroy();
end
```

## Version History

Introduced in R2016a

## See Also

`cancelGenerate` | `cancelForward` | `cancelIterate` | `cancelTimer` | `eventDestroy` | `eventForward` | `eventGenerate` | `eventIterate` | `eventTimer`

## Topics

“Create Custom Blocks Using MATLAB Discrete-Event System Block”



# cancelForward

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Cancel previously scheduled forward events

## Syntax

```
event=cancelForward(destinationType,destinationID)
```

## Description

`event=cancelForward(destinationType,destinationID)` cancel previously scheduled forward events on the current entity. You can then schedule this event by returning it as the output argument when implementing an event action method, such as `entry` or `exit`.

## Input Arguments

### **destinationType** — Destination type

character vector

Destination type. Its value can be either:

- `storage`, if destination of the forward event is a storage element.
- `output`, if destination of forward event is an output port.

### **destinationID** — Destination index

double

Destination index, specified as a double. Its value can be either:

- Storage index, when `destinationType` is `storage`.
- Output port index, when `destinationType` is `output`.

## Output Arguments

### **event** — Event

MATLAB structure

Event for cancelling an entity forward.

## Examples

### **Cancel Previously Schedule Forward Event**

Cancel a previously scheduled forward event of the entity in the current event action context.

```
function [entity,events] = timer(obj,storage,entity,tag)
    % Cancel a previously scheduled forward event of the entity in
    % current event action context. The entity was scheduled to go to
    % storage element 2.
    event1 = obj.cancelForward('storage', 2);

    % Cancel a previously scheduled forward event of the entity in
    % current event action context. The entity was scheduled to go to
    % output port 1.
    event2 = obj.cancelForward('output', 1);
end
```

## Version History

Introduced in R2016a

### See Also

cancelDestroy | cancelGenerate | cancelIterate | cancelTimer | eventDestroy |  
eventForward | eventGenerate | eventIterate | eventTimer

### Topics

“Create Custom Blocks Using MATLAB Discrete-Event System Block”

# cancelGenerate

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Cancel previously scheduled entity generation event

## Syntax

```
event=cancelGenerate(storageID,tag)
```

## Description

`event=cancelGenerate(storageID,tag)` cancels a previously scheduled generation event. You can then schedule this event by returning it as the output argument when implementing an event action method, such as `entry` or `exit`.

## Input Arguments

**storageID** — Storage index

double

Storage index of the to-be-cancelled entity generation event.

**tag** — Tag

character vector

Tag of the to-be-cancelled entity generation event.

## Output Arguments

**event** — Event

MATLAB structure

Event for cancelling an entity generation.

## Examples

### Cancel Previously Scheduled Entity Generation Event

Cancel a previously scheduled entity generation event.

```
function [entity,event] = entry(obj,storage,entity,src)
    % Cancel a previously scheduled entity generation event. The event
    % was scheduled for storage element 3, with a custom tag 'seed'.
```

```
        event = obj.cancelGenerate(3, 'seed');  
end
```

## **Version History**

**Introduced in R2016a**

### **See Also**

cancelDestroy | cancelIterate | cancelTimer | eventDestroy | eventForward |  
eventGenerate

### **Topics**

“Create Custom Blocks Using MATLAB Discrete-Event System Block”

# cancelIterate

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Cancel previously scheduled iterate event

## Syntax

```
event=cancelIterate(storageID,tag)
```

## Description

`event=cancelIterate(storageID,tag)` cancels a previously scheduled iterate event. You can commit the cancellation by returning it as the output argument when implementing an event action method, such as `entry`.

## Input Arguments

**storageID** — Storage index

double

Storage index of the to-be-cancelled iterate event.

**tag** — Tag

character vector

Tag of the to-be-cancelled iterate event.

## Output Arguments

**event** — Event

MATLAB structure

Event for cancelling the specified iterate event.

## Examples

### Cancel Previously Scheduled Iterate Event

Cancel a previously scheduled iterate event.

```
function [entity,event] = entry(obj,storage,entity,src)
    % Cancel a previously scheduled iterate event. The event was
    % scheduled for storage element 2, with a custom tag 'search'.
```

```
        event = obj.cancelIterate(2, 'search');  
end
```

## **Version History**

**Introduced in R2016a**

### **See Also**

cancelDestroy | cancelGenerate | cancelForward | cancelTimer | eventDestroy |  
eventForward | eventGenerate | eventIterate | eventTimer

### **Topics**

“Create Custom Blocks Using MATLAB Discrete-Event System Block”

# cancelTimer

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Cancel previously scheduled timer event

## Syntax

```
event=cancelTimer(tag)
```

## Description

`event=cancelTimer(tag)` cancels a previously scheduled timer event of the current entity. You can commit the cancellation by returning it as the output argument when implementing an event action method, such as `entry`.

## Input Arguments

**tag** — Tag

character vector

Tag of the to-be-cancelled timer event.

## Output Arguments

**event** — Event

MATLAB structure

Event for cancelling the specified timer.

## Examples

### Cancel Previously Scheduled Timer Event

Cancel a previously scheduled timer event of the entity in the current event action context.

```
function [entity,event] = entry(obj,storage,entity,src)
    % Cancel a previously scheduled timer event of the entity in
    % current event action context. The event was scheduled with a
    % custom tag 'timeout'.
    event = obj.cancelTimer('timeout');
end
```

## Version History

Introduced in R2016a

**See Also**

`cancelDestroy` | `cancelGenerate` | `cancelForward` | `cancelIterate` | `eventDestroy` | `eventForward` | `eventGenerate` | `eventIterate` | `eventTimer`

**Topics**

“Create Custom Blocks Using MATLAB Discrete-Event System Block”



# destroy

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Event action upon entity destruction

## Syntax

```
[events]=destroy(obj,storage,entity)
[events,out1,...]=destroy(obj,storage,entity,in1,...)
```

## Description

[events]=destroy(obj,storage,entity) specifies event actions of the object before an entity is destroyed.

[events,out1,...]=destroy(obj,storage,entity,in1,...) specifies such event actions of the object when the block has one or more input signal ports and/or signal output ports.

## Input Arguments

### **obj** — Discrete-event System object

MATLAB object

Discrete-event System object.

### **storage** — Storage

double

Index of the storage element.

### **entity** — Entity

MATLAB structure

Entity leaving storage element. Entity has these fields:

- **sys** (MATLAB structure) — It has these fields:
  - **id** (double) — Entity ID
  - **priority** (double) — Entity priority
- **data** — Entity data

### **in1** — Signal input

any value

Any data inputs of the object. These input arguments exist only when the object has data inputs.

## Output Arguments

### events — Events

vector of MATLAB structures

Events to be scheduled. Use `matlab.DiscreteEventSystem` class methods to create events. Each event has these fields:

- `type` (character vector) — Type of the event
- `delay` (double) — Delay before the event
- `priority` (double) — Priority of the event
- `Storage` (double) — Index of the storage element
- `tag` (character vector) — Event tag
- `location` (MATLAB structure) — Source or destination location of entity

### out1 — Signal output

any value

Data outputs of the object. You must specify these output arguments when the object has data outputs.

## Examples

### Event Action Upon Entity Destruction

Specify event action upon entity destruction in storage.

```
function events = destroy(obj,storage,entity)
    % Upon destroy of an entity, display its ID and schedule to
    % generate a new entity.
    disp(['Entity of ID ' num2str(entity.sys.id) ' is destroyed']);
    events = obj.eventGenerate(storage, 'Refill', 1, entity.sys.priority);
end
```

## Version History

Introduced in R2016a

### See Also

`matlab.DiscreteEventSystem` | `blocked` | `entry` | `exit` | `generate` | `getEntityPortsImpl` | `getEntityStorageImpl` | `getEntityTypesImpl` | `iterate` | `setupEvents` | `timer`

### Topics

“Create Custom Blocks Using MATLAB Discrete-Event System Block”

# entityType

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Define entity type

## Syntax

```
entitytype=entityType(name)
entitytype=entityType(name,datatype)
entitytype=entityType(name,datatype,dimensions)
entitytype=entityType(name,datatype,dimensions,complexity)
```

## Description

`entitytype=entityType(name)` defines a named entity type.

`entitytype=entityType(name,datatype)` defines a named entity type that takes real values and with specified data type of size 1.

`entitytype=entityType(name,datatype,dimensions)` defines a named entity type that takes real values and with specified data type and size.

`entitytype=entityType(name,datatype,dimensions,complexity)` defines a named entity type with a specified data type, dimensions, and complexity.

## Input Arguments

### **name — Entity type name**

character vector

Entity type name.

### **datatype — Data type (optional)**

character vector

Data type that specifies the data type of the entity. The data type must be a built-in data type or a bus object.

### **dimensions — Dimensions (optional)**

vector of doubles

Dimensions, specified as a vector of doubles, specifying the dimensions of the entity.

### **complexity — Complexity (optional)**

logical | double

Complexity, specified as a logical or double value, specifying the complexity of the entity:

- `false` or `0` — If the entity contains real values.

- true or any positive number — If the entity contains complex values.

## Output Arguments

### **entitytype** — Entity type

MATLAB structure

Entity type, specified as a MATLAB structure.

## Examples

### Define Entity Type

Define entity types type1, type2, and type3.

```
function entityTypes = getEntityTypesImpl(obj)
    % Define entity type 'type1' with inherited data type, dimension
    % and complexity
    t1 = obj.entityType('type1');

    % Define entity type 'type2' with specified data type ('mybus'),
    % default dimension and complexity (i.e. scalar real values)
    t2 = obj.entityType('type2', 'mybus');

    % Define entity type 'type3' with specified data type ('double'),
    % dimension (2 by 3 matrix), and complexity (complex)
    t3 = obj.entityType('type3', 'double', [2 3], true);

    entityTypes = [t1, t2, t3];
end
```

## Version History

Introduced in R2016a

### See Also

matlab.DiscreteEventSystem | getEntityTypesImpl

### Topics

“Create Custom Blocks Using MATLAB Discrete-Event System Block”

# entry

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Event action when entity enters storage element

## Syntax

```
[entity,events]=entry(obj,storage,entity,source)
[entity,events,out1,...]=entry(obj,storage,entity,source,in1,...)
```

## Description

[entity,events]=entry(obj,storage,entity,source) specifies event actions of the object when an entity enters storage.

[entity,events,out1,...]=entry(obj,storage,entity,source,in1,...) such event actions of the object when the block has one or more input signal ports and/or signal output ports.

## Input Arguments

### obj — Discrete-event System object

MATLAB object

Discrete-event System object.

### storage — Storage

double

Index of the storage element.

### entity — Entity

MATLAB structure

Entity entering storage component. Entity has these fields:

- sys (MATLAB structure) — It has these fields:
  - id (double) — Entity ID
  - priority (double) — Entity priority
- data — Entity data

### source — Source location

MATLAB structure

Source location of entity, such as an input port or a storage element. It has these fields:

- type (character vector) — Specify input or storage
- index (double) — Input or storage index

**in1 – Signal input**

any value

Any data inputs of the object. These input arguments exist only when the object has data inputs.

**Output Arguments****entity – Entity**

MATLAB structure

Entity entering storage, possibly with changed data. See “entity” on page 1-0 .

**events – Events**

vector of MATLAB structures

Events to be scheduled. Use `matlab.DiscreteEventSystem` class methods to create events. Each event has these fields:

- `type` (character vector) — Type of the event
- `delay` (double) — Delay before the event
- `priority` (double) — Priority of the event
- `Storage` (double) — Index of the storage element
- `tag` (character vector) — Event tag
- `location` (MATLAB structure) — Source or destination location of entity, see “source” on page 1-0

**out1 – Signal output**

any value

Data outputs of the object. You must specify these output arguments when the object has data outputs.

**Examples****Event Action Upon Entity Entry**

Event action for entity entry to storage.

```
function [entity,events] = entry(obj,storage,entity,src)
% Specify event actions when entity entered storage.
disp(['Entity of ID ' num2str(entity.sys.id) ...
      ' has entered storage element ' num2str(storage)]);
switch src.type
case 'input'
disp(['Entity came from input port ' num2str(src.index)]);
case 'storage'
disp(['Entity came from storage element ' num2str(src.index)]);
end
events = [ ...
obj.eventDestroy(), ... % Destroy the newly entered entity
obj.eventIterate(2, '')]; % Iterate entities in storage element 2
end
```

**Create a Custom Entity Storage Block to Delay Entities**

This example shows how to use discrete-event System object methods to create a custom entity storage block that has one input port, one output port, and one storage element. The discrete-event

System object is the instantiation of the `matlab.DiscreteEventSystem` class, which allows you to use the implementation and service methods provided by this class. Then, you use the MATLAB Discrete-Event System block to integrate the System object into a SimEvents model.

The custom MATLAB Discrete-Event System block accepts an entity from its input port and forwards it to its output port with a specified delay. For more information, see “Delay Entities with a Custom Entity Storage Block”.

```
classdef CustomEntityStorageBlock < matlab.DiscreteEventSystem

    % A custom entity storage block with one input, one output, and one storage.

    % Nontunable properties
    properties (Nontunable)
        % Capacity
        Capacity = 1;
        % Delay
        Delay=4;
    end

    methods (Access=protected)
        function num = getNumInputsImpl(~)
            num = 1;
        end

        function num = getNumOutputsImpl(~)
            num = 1;
        end

        function entityTypes = getEntityTypesImpl(obj)
            entityTypes = obj.entityType('Car');
        end

        function [inputTypes,outputTypes] = getEntityPortsImpl(obj)
            inputTypes = {'Car'};
            outputTypes = {'Car'};
        end

        function [storageSpecs, I, O] = getEntityStorageImpl(obj)
            storageSpecs = obj.queueFIFO('Car', obj.Capacity);
            I = 1;
            O = 1;
        end
    end

    end

    methods

        function [entity,event] = CarEntry(obj,storage,entity,source)
            % Specify event actions when entity enters storage.

            event = obj.eventForward('output', 1, obj.Delay);
        end
    end

end
```

## Version History

Introduced in R2016a

## See Also

`matlab.DiscreteEventSystem` | `blocked` | `destroy` | `exit` | `generate` | `getEntityPortsImpl` | `getEntityStorageImpl` | `getEntityTypesImpl` | `iterate` | `setupEvents` | `timer`

## Topics

“Create Custom Blocks Using MATLAB Discrete-Event System Block”

# eventAcquireResource

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Create a resource acquisition event

## Syntax

```
event = eventAcquireResource(resourceSpec, tag)
```

## Description

`event = eventAcquireResource(resourceSpec, tag)` creates an event to acquire resources from existing Resource Pool blocks. You can specify names and amount of resources to acquire. For more details, see `resourceSpecification`.

If all the requested resources are not available during the event execution, the acquisition event remains active. When the requested resources become available, the event is rescheduled for immediate execution.

## Input Arguments

**resourceSpec — Specify name and amount of resources for acquisition**

array of MATLAB structures

Specify the name and the amount of resources to be acquired by the entities.

**tag — Identifier tag for the resource acquisition event**

character vector

Custom tag of this entity resource acquisition event. You can use the tag to identify an event when multiple events act on the same entity. For more information about managing multiple events, see “Custom Entity Storage Block with Multiple Timer Events”.

## Output Arguments

**event — Resource acquisition event**

MATLAB structure

Event that acquires resources for the entity.

## Examples

### Acquire Resources upon Entry

On entity entry to a storage element, an entity acquires one resource of type `Test1`. The tag of this resource acquisition event is `TestTag`.

```
function [entity,events] = entry(obj, storage, entity, source)
% On entity entry, acquire a resource from the specified pool.
```



```
resourceSpec = obj.resourceSpecification('Test1', 1);
event = obj.eventAcquireResource(resourceSpec, 'TestTag');
end
```

## Custom Block to Acquire Resources

This example shows how to use resource management methods to create a custom entity storage block in which entities acquire resources from specified Resource Pool blocks.

Suppose that you manage a facility that produces parts from two different materials, material 1 and material 2, to fulfill orders. After a part is produced, it is evaluated for quality assurance.

Two testing methods for quality control are:

- Test 1 is used for parts that are produced from material 1.
- Test 2 is used for parts that are produced from material 2

After the production phase, parts are tagged based on their material to apply the correct test.

For more information, see “Create a Custom Resource Acquirer Block”.

```
classdef CustomBlockAcquireResources < matlab.DiscreteEventSystem
    % Custom resource acquire block example.

    methods(Access = protected)

        function num = getNumInputsImpl(obj)
            num = 1;
        end

        function num = getNumOutputsImpl(obj)
            num = 1;
        end

        function entityTypes = getEntityTypesImpl(obj)
            entityTypes(1) = obj.entityType('Part');
        end

        function [input, output] = getEntityPortsImpl(obj)
            input = {'Part'};
            output = {'Part'};
        end

        function [storageSpec, I, O] = getEntityStorageImpl(obj)
            storageSpec(1) = obj.queueFIFO('Part', 1);
            I = 1;
            O = 1;
        end

        function resNames = getResourceNamesImpl(obj)
            % Define the names of the resources to be acquired.
            resNames = obj.resourceType('Part', {'Test1', 'Test2'});
        end

    end

    methods

        function [entity,events] = entry(obj, storage, entity, source)
            % On entity entry, acquire a resource from the specified pool.
            if entity.data.Test == 1
                % If the entity is produced from Material1, request Test1.
                resReq = obj.resourceSpecification('Test1', 1);
            else
                % If the entity is produced from Material2, request Test2.
                resReq = obj.resourceSpecification('Test2', 1);
            end
            % Acquire the resource from the corresponding pool.
            events = obj.eventAcquireResource(resReq, 'TestTag');
        end

    end
end
```

```
function [entity,events] = resourceAcquired(obj, storage,...
    entity, resources, tag)
% After the resource acquisition, forward the entity to the output.
    events = obj.eventForward('output', storage, 0.0);
end

end

end
```

## Version History

Introduced in R2019a

### See Also

matlab.DiscreteEventSystem | eventForward | eventReleaseResource |  
cancelAcquireResource | getResourceNamesImpl | resourceReleased |  
resourceSpecification

### Topics

“Create a Custom Resource Acquirer Block”

“Create Custom Blocks Using MATLAB Discrete-Event System Block”

# eventDestroy

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Create entity destroy event

## Syntax

```
event=eventDestroy()
```

## Description

`event=eventDestroy()` creates an event to destroy an entity. You can then schedule this event by returning it as an output argument when implementing an event action method, such as `timer`.

## Output Arguments

### **event** — Event

MATLAB structure

Event that destroys the entity in current event action context.

## Examples

### **Destroy Entity in Current Event Action Context**

Define an event to destroy the entity in current event action context.

```
function [entity,event] = entry(obj,storage,entity,src)
    % Define an event to destroy the entity in current event action
    % context.
    event = obj.eventDestroy();
end
```

## Version History

Introduced in R2016a

## See Also

`cancelDestroy` | `cancelGenerate` | `cancelForward` | `cancelIterate` | `cancelTimer` | `eventForward` | `eventGenerate` | `eventIterate` | `eventTimer`

## Topics

“Create Custom Blocks Using MATLAB Discrete-Event System Block”

# eventForward

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Create entity forward event

## Syntax

```
event=eventForward(locationType,locationIndex,delay)
```

## Description

`event=eventForward(locationType,locationIndex,delay)` creates an event to forward an entity from the current location to a new location. You can then schedule this event by returning it as the output argument when implementing an event action method, such as `entry`.

## Input Arguments

### **locationType** — Location type

character vector

Type of the new location. Specify 'storage' if the new location is a storage element of the discrete-event system. Specify 'output' if you want the entity to exit from an output port of the discrete-event system.

### **locationIndex** — Location index

double

Index of the new location. If location type is 'storage', it indicates the index of a storage element. If location type is 'output', it indicates the index of an output port.

### **delay** — Delay

double

Time delay between current simulation time and the time the entity will be forwarded.

## Output Arguments

### **event** — Event

MATLAB structure

Event that forwards the entity in current event action context to a new location.

## Examples

### **Forward Current Entity to Storage**

Define an event that forwards the current entity to storage.

```

function [entity,events] = entry(obj,storage,entity,src)

    % Define an event that forwards the current entity to storage
    % element 2. Event shall be scheduled to execute 0.8 second later.
    event1 = obj.eventForward('storage', 2, 0.8);

    % Define an event that forwards the current entity to output port 1.
    % Event shall be scheduled to execute at current simulation clock time.
    event2 = obj.eventForward('output', 1, 0);

    % Define events as event1 and event2
    events = [event1, event2];
end

```

## Create a Custom Entity Storage Block to Delay Entities

This example shows how to use discrete-event System object methods to create a custom entity storage block that has one input port, one output port, and one storage element. The discrete-event System object is the instantiation of the `matlab.DiscreteEventSystem` class, which allows you to use the implementation and service methods provided by this class. Then, you use the MATLAB Discrete-Event System block to integrate the System object into a SimEvents model.

The custom MATLAB Discrete-Event System block accepts an entity from its input port and forwards it to its output port with a specified delay. For more information, see “Delay Entities with a Custom Entity Storage Block”.

```

classdef CustomEntityStorageBlock < matlab.DiscreteEventSystem

    % A custom entity storage block with one input, one output, and one storage.

    % Nontunable properties
    properties (Nontunable)
        % Capacity
        Capacity = 1;
        % Delay
        Delay=4;
    end

    methods (Access=protected)
        function num = getNumInputsImpl(~)
            num = 1;
        end

        function num = getNumOutputsImpl(~)
            num = 1;
        end

        function entityTypes = getEntityTypesImpl(obj)
            entityTypes = obj.entityType('Car');
        end

        function [inputTypes,outputTypes] = getEntityPortsImpl(obj)
            inputTypes = {'Car'};
            outputTypes = {'Car'};
        end

        function [storageSpecs, I, O] = getEntityStorageImpl(obj)
            storageSpecs = obj.queueFIFO('Car', obj.Capacity);
            I = 1;
            O = 1;
        end
    end

    methods
        function [entity,event] = CarEntry(obj,storage,entity,source)
            % Specify event actions when entity enters storage.

            event = obj.eventForward('output', 1, obj.Delay);
        end
    end
end

```

end

## **Version History**

**Introduced in R2016a**

### **See Also**

cancelForward | eventDestroy | eventGenerate | eventIterate | eventTimer

### **Topics**

“Create Custom Blocks Using MATLAB Discrete-Event System Block”

# eventGenerate

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Create entity generate event

## Syntax

```
event=eventGenerate(storageID,tag,delay,priority)
```

## Description

`event=eventGenerate(storageID,tag,delay,priority)` creates an event to generate an entity. You can then schedule this event by returning it as the output argument when implementing an event action method, such as `entry`.

## Input Arguments

### **storageID** — Storage index

double

Index of the storage element, where a new entity will be generated.

### **tag** — Tag

character vector

Custom tag of this entity generate event.

### **delay** — Delay

double

Time delay between current simulation time and the time the entity will be generated.

### **priority** — Priority

double

Positive integer value indicating system priority of the new entity. A smaller numeric value indicates a higher priority.

## Output Arguments

### **event** — Event

MATLAB structure

Event that generates an new entity in the specified storage element.

## Examples

### Define Entity Generation Event

Define entity generation event in storage element 3.

```
function event = setupEvents(obj)
    % Define an entity generation event
    % - A new entity shall be created in storage element 3
    % - The event has a custom tag 'seed'
    % - The event shall be executed 0.5 second later
    % - The new entity shall be initialized with a priority of 200
    event = obj.eventGenerate(3, 'seed', 0.5, 200);
end
```

### Create a Custom Block to Generate Entities

This example shows how to create a custom source block that generates entities and to manage discrete states when implementing the discrete-event System object methods.

For more information, see “Custom Entity Generator Block with Signal Input and Signal Output”.

```
classdef CustomEntityStorageBlockGeneration < matlab.DiscreteEventSystem...
    & matlab.System
    % A custom entity generator block.

    % Nontunable properties
    properties (Nontunable)
        % Generation period
        period = 1;
    end

    properties(DiscreteState)
        % Entity priority
        priority;
        % Entity value
        value;
    end

    % Discrete-event algorithms
    methods
        function [events, out1] = setupEvents(obj)
            % Set up entity generation events at simulation start.
            events = obj.eventGenerate(1,'mygen',obj.period,obj.priority);
            % Set up the initial value of the output signal.
            out1 = 10;
        end

        function [entity,events,out1] = generate(obj,storage,entity,tag,in1)
            % Specify event actions when entity is generated in storage.
            entity.data = obj.value;
            % The priority value is assigned from the input signal.
            obj.priority = in1;
            % Output signal is the assigned priority value.
            out1 = obj.priority;
            events = [obj.eventForward('output',1,0) ...
                    obj.eventGenerate(1,'mygen',obj.period,obj.priority)];
        end
    end

    methods(Access = protected)

        function entityTypes = getEntityTypesImpl(obj)
            entityTypes = obj.entityType('Material');
        end

        function [inputTypes,outputTypes] = getEntityPortsImpl(obj)
            % Specify entity input and output ports. Return entity types at
            % a port as strings in a cell array. Use empty string to
            % indicate a data port.
            inputTypes = {''};
            outputTypes = {'Material',''};
        end
    end
end
```



```

end

function resetImpl(obj)
    % Initialize / reset discrete-state properties.
    obj.priority = 10;
    obj.value = 1:12;
end

function [storageSpecs, I, O] = getEntityStorageImpl(obj)
    storageSpecs = obj.queueFIFO('Material', 1);
    I = 0;
    O = [1 0];
end

function num = getNumInputsImpl(obj)
    % Define total number of inputs for system with optional
    % inputs.
    num = 1;
end

function num = getNumOutputsImpl(~)
    % Define total number of outputs.
    num = 2;
end

function [out1 out2] = getOutputSizeImpl(obj)
    % Return size for each output port.
    out1 = [1 12];
    out2 = 1;
end

function [out1 out2] = getOutputDataTypeImpl(obj)
    % Return data type for each output port.
    out1 = "double";
    out2 = "double";
end

function [out1 out2] = isOutputComplexImpl(obj)
    % Return true for each output port with complex data.
    out1 = false;
    out2 = false;
end

function [sz,dt,cp] = getDiscreteStateSpecificationImpl(obj,name)
    % Return size, data type, and complexity of discrete-state
    % specified in name.
    switch name
        case 'priority'
            sz = [1 1];
        case 'value'
            sz = [1 12];
    end
    dt = "double";
    cp = false;
end
end
end

```

## Version History

Introduced in R2016a

### See Also

cancelGenerate | cancelTimer | eventDestroy | eventForward | eventIterate | eventTimer

### Topics

“Create Custom Blocks Using MATLAB Discrete-Event System Block”

## eventIterate

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Create entity iterate event

### Syntax

```
event=eventIterate(storageID,tag,priority)
```

### Description

`event=eventIterate(storageID,tag,priority)` creates an event to repeatedly process entities of a storage element. You can then schedule this event by returning it as the output argument when implementing an event action method, such as `exit`.

### Input Arguments

**storageID** — Storage index

double

Index of a storage element. Entities inside this storage element will be iterated.

**tag** — Tag

character vector

Custom tag of this entity iterate event.

**priority** — Priority (optional)

double

Priority of the entity iterate event. This value must be a positive integer, where a smaller value indicates a higher event priority.

### Output Arguments

**event** — Event

MATLAB structure

Event that processes entities of a specific storage element.

### Examples

#### Iterate Entities of a Storage Element

Define an event to iterate entities of a storage element..

```
function event = exit(obj,storage,entity,dst)
    % Define an event to iterate entities of a storage element
```

```

% - The event is regarding to storage element 2
% - The event has a custom tag 'search'
% - The event shall be executed at current simulation clock time
% - The event has a priority of 10 (a smaller numeric value
%   indicates a higher event priority)
event = obj.eventIterate(2, 'search', 10);
end

```

## Custom Entity Storage Block with Iterate Event

In this example, a custom block allows entities to enter its storage element through its input port. The storage element is a priority queue that sorts the entities based on their `Diameter` attribute in ascending order. Every entity entry to the block's storage invokes an iteration event to display the diameter and the position of each entity in the storage.

For more information, see “Create a Custom Entity Storage Block with Iteration Event”.

```

classdef CustomEntityStorageBlockIteration < matlab.DiscreteEventSystem

    % A custom entity storage block with one input port and one storage element.

    % Nontunable properties
    properties (Nontunable)
        % Capacity
        Capacity = 5;
    end
    % Create the storage element with one input and one storage.
    methods (Access=protected)

        function num = getNumInputsImpl(obj)
            num = 1;
        end

        function num = getNumOutputsImpl(obj)
            num = 0;
        end

        function entityType = getEntityTypesImpl(obj)
            entityType1 = obj.entityType('Wheel');
            entityType = entityType1;
        end

        function [inputTypes,outputTypes] = getEntityPortsImpl(obj)
            inputTypes = {'Wheel'};
            outputTypes={};
        end

        function [storageSpecs, I, O] = getEntityStorageImpl(obj)
            storageSpecs = obj.queuePriority('Wheel',obj.Capacity, 'Diameter','ascending');
            I = 1;
            O = [];
        end

    end
end
% Entity entry event action
methods

    function [entity, event] = WheelEntry(obj,storage,entity, source)
        % Entity entry invokes an iterate event.
        event = obj.eventIterate(1, '');
    end

    % The itarate event action
    function [entity,event,next] = WheelIterate(obj,storage,entity,tag,cur)
        % Display wheel id, position in the storage, and diameter.
        coder.extrinsic('fprintf');
        fprintf('Wheel id %d, Current position %d, Diameter %d\n', ...
            entity.sys.id, cur.position, entity.data.Diameter);
        if cur.size == cur.position
            fprintf('End of Iteration \n')
        end
        next = true;
        event=[];
    end
end

```

```
end  
end  
end
```

## **Version History**

**Introduced in R2016a**

### **See Also**

`cancelIterate` | `cancelTimer` | `eventDestroy` | `eventForward` | `eventGenerate` | `eventTimer`

### **Topics**

“Create Custom Blocks Using MATLAB Discrete-Event System Block”

# eventReleaseAllResources

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Create an event to release all resources acquired by an entity

## Syntax

```
event = eventReleaseAllResources(tag)
```

## Description

`event = eventReleaseAllResources(tag)` creates an event to release all the resources acquired by an entity.

## Input Arguments

**tag** — Identifier tag for the resource release event

character vector

Custom tag of this entity resource release event. You can use the tag to identify an event when multiple events act on the same entity. For more information about managing multiple events, see “Custom Entity Storage Block with Multiple Timer Events”.

## Output Arguments

**event** — Resource release event

MATLAB structure

Event that releases all resources from the entity.

## Examples

On entity entry to a storage element, an entity releases all of the previously acquired resources. The tag of this resource acquisition event is `ReleaseAll`.

```
function [entity, event] = entry(obj, storage, entity, source)
    event = obj.eventReleaseAllResources('releaseAll');
end
```

## Version History

Introduced in R2019a

## See Also

matlab.DiscreteEventSystem | eventForward | getResourceNamesImpl | resourceReleased | eventReleaseResource

**Topics**

“Create a Custom Resource Acquirer Block”

“Create Custom Blocks Using MATLAB Discrete-Event System Block”

# eventReleaseResource

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Create an event to release previously acquired resources

## Syntax

```
event = eventReleaseResource(resourceSpec, tag)
```

## Description

`event = eventReleaseResource(resourceSpec, tag)` creates an event for entities to release previously acquired resources. You can specify the type and amount of resources to release. For more details, see [resourceSpecification](#).

If the amount of resources to be released is larger than the previously acquired resources, then all the resources are released.

## Input Arguments

**resourceSpec — Specify name and amount of resources for release**

array of MATLAB structures

Specify the name and the amount of resources to be released by the entities.

**tag — Identifier tag for the resource release event**

character vector

Custom tag of this entity resource release event. You can use the tag to identify an event when multiple events act on the same entity. For more information about managing multiple events, see “Custom Entity Storage Block with Multiple Timer Events”.

## Output Arguments

**event — Resource release event**

MATLAB structure

Event that releases resources from the entity.

## Examples

### Acquire Resources on Entry

On entity entry to a storage element, an entity releases one resource of type `Test1`. The tag of this resource acquisition event is `myTag`.

```
function [entity,events] = entry(obj, storage, entity, source)
% On entity entry, release a resource from the specified pool.
```

```
resourceSpec = obj.resourceSpecification('Test1', 1);  
event = obj.eventReleaseResource(resourceSpec, 'myTag');  
end
```

## Version History

Introduced in R2019a

## See Also

matlab.DiscreteEventSystem | eventForward | cancelAcquireResource |  
getResourceNamesImpl | resourceAcquired | eventAcquireResource |  
resourceSpecification

## Topics

“Create a Custom Resource Acquirer Block”

“Create Custom Blocks Using MATLAB Discrete-Event System Block”



# eventTestEntry

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Create an event to indicate that the acceptance policy for the storage has changed and the storage retests arriving entities

## Syntax

```
event = eventTestEntry(storageId)
```

## Description

`event = eventTestEntry(storageId)` creates an event to retest entities arriving at storage.

## Input Arguments

**storageId** — Index of storage element

scalar

Index of the storage element that is to be unblocked, specified as a scalar.

Data Types: double

## Output Arguments

**event** — Unblock

MATLAB structure

Event that unblocks a storage or input port for processing entities, specified as a MATLAB structure.

## Examples

### Event Action to Retest Entity Entry

Retest entity entry to storage.

```
function [entity,events] = exit(obj,storage,entity,dst)
    % Indicates that more entities can be accepted and acceptance policy has changed
    events = [events,obj.eventTestEntry(storage)];
end
```

## Version History

Introduced in R2018a

## See Also

matlab.DiscreteEventSystem | iterate

**Topics**

“Create Custom Blocks Using MATLAB Discrete-Event System Block”

# eventTimer

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Create entity timer event

## Syntax

```
event=eventTimer(tag, delay)
```

## Description

`event=eventTimer(tag, delay)` creates an event to delay an entity for a period of time. You can then schedule the timer by returning it as the output argument when implementing an event action method, such as `entry`.

## Input Arguments

### tag — Tag

character vector

Custom tag of this entity timer event.

### delay — Delay

double

Time delay between current simulation time and the time that this timer event will be executed.

## Output Arguments

### event — Event

MATLAB structure

Event that delays the entity in current event action context for a period of time.

## Examples

### Define Timer Event

Define a timer event.

```
function [entity,event] = entry(obj,storage,entity,src)
    % Define a timer event
    % - The event is regarding the entity in current event action context
    % - The event has a custom tag 'timeout'
    % - The event will be executed 3.0 seconds later
    event = obj.eventTimer('timeout', 3.0);
end
```

### Custom Block with Timer Events

This example uses a custom entity storage block with one input, two outputs, and a storage element. An entity of type Part with TimeOut attribute enters the storage of the custom block to be

processed. `Timeout` determines the maximum allowed processing time of the parts. When a part enters the storage, two timer events are activated. One timer tracks the processing time of the part in the oven. When this timer expires, the entity is forwarded to output 1. Another timer acts as a fail-safe and tracks if the maximum allowed processing time is exceeded or not. When this timer expires, the process is terminated and the entity is forwarded to the output 2.

For more information, see “Custom Entity Storage Block with Multiple Timer Events”.

```
classdef CustomEntityStorageBlockTimer < matlab.DiscreteEventSystem

    % A custom entity storage block with one input port, two output ports, and one storage.

    % Nontunable properties
    properties (Nontunable)
        % Capacity
        Capacity = 1;
    end

    methods (Access=protected)

        function num = getNumInputsImpl(~)
            num = 1;
        end

        function num = getNumOutputsImpl(~)
            num = 2;
        end

        function entityTypes = getEntityTypesImpl(obj)
            entityTypes = obj.entityType('Part');
        end

        function [inputTypes,outputTypes] = getEntityPortsImpl(obj)
            inputTypes = {'Part'};
            outputTypes = {'Part' 'Part'};
        end

        function [storageSpecs, I, O] = getEntityStorageImpl(obj)
            storageSpecs = obj.queueFIFO('Part', obj.Capacity);
            I = 1;
            O = [1 1];
        end

    end

    methods

        function [entity,event] = PartEntry(obj,storage,entity,source)
            % Specify event actions when entity enters storage.
            ProcessingTime=randi([1 15]);
            event1 = obj.eventTimer('Timeout', entity.data.Timeout);
            event2 = obj.eventTimer('ProcessComplete', ProcessingTime);
            event = [event1 event2];
        end

        function [entity, event] = timer(obj,storage,entity,tag)
            % Specify event actions for when scheduled timer completes.
            event = obj.initEventArray;
            switch tag
                case 'ProcessComplete'
                    event = obj.eventForward('output', 1, 0);
                case 'Timeout'
                    event = obj.eventForward('output', 2, 0);
            end
        end

    end

end
```

## Version History

### Introduced in R2016a

**See Also**

cancelDestroy | cancelGenerate | cancelForward | cancelIterate | cancelTimer |  
eventDestroy | eventForward | eventGenerate | eventIterate

**Topics**

“Create Custom Blocks Using MATLAB Discrete-Event System Block”

## exit

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Event action before entity exit from storage

### Syntax

```
[events]=exit(obj,storage,entity,destination)
[events,out1,...]=exit(obj,storage,entity,destination,in1,...)
```

### Description

[events]=exit(obj,storage,entity,destination) specifies event actions of the object when an entity exits a storage.

[events,out1,...]=exit(obj,storage,entity,destination,in1,...) specifies such event actions of the object when the block has one or more input signal ports and/or signal output ports.

### Input Arguments

#### **obj** — Discrete-event System object

MATLAB object

Discrete-event System object.

#### **storage** — Storage

double

Index of the storage element.

#### **entity** — Entity

MATLAB structure

Entity leaving storage element. Entity has these fields:

- **sys** (MATLAB structure) — It has these fields:
  - **id** (double) — Entity ID
  - **priority** (double) — Entity priority
- **data** — Entity data

#### **destination** — Destination

MATLAB structure

Destination of entity, such as an output port or a storage element. It has these fields:

- **type** (character vector) — Specify output, storage, or extract
- **index** (double) — Output or storage index

The type is specified as `extract` if an entity is being extracted from a Discrete-Event System block.

### **in1 — Data inputs**

any value

Any data inputs of the object. These input arguments exist only when the object has data inputs.

## **Output Arguments**

### **events — Events**

vector of MATLAB structures

Events to be scheduled after the method returns. Use `matlab.DiscreteEventSystem` class methods to create events. Each event has these fields:

- `type` (character vector) — Type of the event
- `delay` (double) — Delay before the event
- `priority` (double) — Priority of the event
- `Storage` (double) — Index of the storage element
- `tag` (character vector) — Event tag
- `location` (MATLAB structure) — Source or destination location of entity

### **out1 — Signal output**

any value

Data outputs of the object. You must specify these output arguments when the object has data outputs.

## **Examples**

### **Refill Upon Entity Exit Storage**

Create a new entity when an existing entity exits the storage element.

```
function events = exit(obj,storage,entity,dst)
    % Upon exit of an entity, display its ID and schedule to
    % generate a new entity.
    disp(['Entity of ID ' num2str(entity.sys.id) ' has exited']);
    events = obj.eventGenerate(storage, 'Refill', 1, entity.sys.priority);
end
```

## **Version History**

Introduced in R2016a

### **See Also**

`matlab.DiscreteEventSystem` | `blocked` | `destroy` | `entry` | `generate` | `getEntityPortsImpl` | `getEntityStorageImpl` | `getEntityTypesImpl` | `iterate` | `setupEvents` | `timer`

**Topics**

“Create Custom Blocks Using MATLAB Discrete-Event System Block”



# generate

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Event action upon entity creation

## Syntax

```
[entity,events]=generate(obj,storage,entity,tag)
[entity,events,out1,...]=generate(obj,storage,entity,tag,in1,...)
```

## Description

[entity,events]=generate(obj,storage,entity,tag) specifies event actions of the object when an entity is created inside a storage component.

[entity,events,out1,...]=generate(obj,storage,entity,tag,in1,...) specifies such event actions of the object when the block has one or more input signal ports and/or signal output ports.

## Input Arguments

**obj** — Discrete-event System object

MATLAB object

Discrete-event System object.

**storage** — Storage

double

Index of the storage element.

**entity** — Entity

MATLAB structure

Entity to create inside storage element. Entity has these fields:

- **sys** (MATLAB structure) — It has these fields:
  - **id** (double) — Entity ID
  - **priority** (double) — Entity priority
- **data** — Entity data

**tag** — Tag

character vector

Tag of the current entity generation event.

**in1** — Input

any value

Any data inputs of the object. These input arguments exist only when the object has data inputs.

## Output Arguments

### **entity** – Entity

MATLAB structure

Entities created with possibly changed values.

### **events** – Events

vector of MATLAB structures

Events to be scheduled for just after entities are created. Use `matlab.DiscreteEventSystem` class methods to create events. Each event has these fields:

- `type` (character vector) – Type of the event
- `delay` (double) – Delay before the event
- `priority` (double) – Priority of the event
- `Storage` (double) – Index of the storage element
- `tag` (character vector) – Event tag
- `location` (MATLAB structure) – Source or destination location of entity

### **out1** – Data output

any value

Data outputs of the object. You must specify these output arguments when the object has data outputs.

## Examples

### Set Initial Values When Entity is Generated

Initialize attribute values when entity is generated in a storage element.

```
function [entity,events] = generate(obj,storage,entity,tag)
    % Specify event actions when entity generated in storage.
    % - For entity generation event of tag 'Adam', initialize the
    %   entity so that its attribute 'gender' has value '0', and its
    %   priority is '200'.
    % - For entity generation event of tag 'Eve', initialize the
    %   entity so that its attribute 'gender' has value '1', and its
    %   priority is '100'.
    % - An event is returned to forward the entity to storage
    %   element 2 with a time delay of 0.6.
    switch tag
        case 'Adam'
            entity.data.gender = 0;
            entity.sys.priority = 200;
        case 'Eve'
            entity.data.gender = 1;
            entity.sys.priority = 100;
    end
```

```

    events = obj.eventForward('storage',2,0.6);
end

```

## Create a Custom Block to Generate Entities

This example shows how to create a custom source block that generates entities and to manage discrete states when implementing the discrete-event System object methods.

For more information, see “Custom Entity Generator Block with Signal Input and Signal Output”.

```

classdef CustomEntityStorageBlockGeneration < matlab.DiscreteEventSystem...
    & matlab.System
    % A custom entity generator block.

    % Nontunable properties
    properties (Nontunable)
        % Generation period
        period = 1;
    end

    properties(DiscreteState)
        % Entity priority
        priority;
        % Entity value
        value;
    end

    % Discrete-event algorithms
    methods
        function [events, out1] = setupEvents(obj)
            % Set up entity generation events at simulation start.
            events = obj.eventGenerate(1,'mygen',obj.period,obj.priority);
            % Set up the initial value of the output signal.
            out1 = 10;
        end

        function [entity,events,out1] = generate(obj,storage,entity,tag,in1)
            % Specify event actions when entity is generated in storage.
            entity.data = obj.value;
            % The priority value is assigned from the input signal.
            obj.priority = in1;
            % Output signal is the assigned priority value.
            out1 = obj.priority;
            events = [obj.eventForward('output',1,0) ...
                    obj.eventGenerate(1,'mygen',obj.period,obj.priority)];
        end
    end

    methods(Access = protected)

        function entityTypes = getEntityTypesImpl(obj)
            entityTypes = obj.entityType('Material');
        end

        function [inputTypes,outputTypes] = getEntityPortsImpl(obj)
            % Specify entity input and output ports. Return entity types at
            % a port as strings in a cell array. Use empty string to
            % indicate a data port.
            inputTypes = {''};
            outputTypes = {'Material',''};
        end

        function resetImpl(obj)
            % Initialize / reset discrete-state properties.
            obj.priority = 10;
            obj.value = 1:12;
        end

        function [storageSpecs, I, O] = getEntityStorageImpl(obj)
            storageSpecs = obj.queueFIFO('Material', 1);
            I = 0;
            O = [1 0];
        end

        function num = getNumInputsImpl(obj)
            % Define total number of inputs for system with optional
            % inputs.
            num = 1;
        end
    end
end

```

```
end

function num = getNumOutputsImpl(~)
    % Define total number of outputs.
    num = 2;
end
function [out1 out2] = getOutputSizeImpl(obj)
    % Return size for each output port.
    out1 = [1 12];
    out2 = 1;
end

function [out1 out2] = getOutputDataTypeImpl(obj)
    % Return data type for each output port.
    out1 = "double";
    out2 = "double";
end

function [out1 out2] = isOutputComplexImpl(obj)
    % Return true for each output port with complex data.
    out1 = false;
    out2 = false;
end

function [sz,dt,cp] = getDiscreteStateSpecificationImpl(obj,name)
    % Return size, data type, and complexity of discrete-state
    % specified in name.
    switch name
        case 'priority'
            sz = [1 1];
        case 'value'
            sz = [1 12];
    end
    dt = "double";
    cp = false;
end
end
end
```

## Version History

Introduced in R2016a

### See Also

[matlab.DiscreteEventSystem](#) | [blocked](#) | [destroy](#) | [entry](#) | [exit](#) | [getEntityPortsImpl](#) | [getEntityStorageImpl](#) | [getEntityTypesImpl](#) | [iterate](#) | [setupEvents](#) | [timer](#)

### Topics

“Create Custom Blocks Using MATLAB Discrete-Event System Block”

# getEntityPortsImpl

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Define input ports and output ports of discrete-event system

## Syntax

```
[inputTypes,outputTypes]=getEntityPortsImpl(obj)
```

## Description

[inputTypes,outputTypes]=getEntityPortsImpl(obj) defines input ports and output ports of a discrete-event system.

## Input Arguments

**obj** — Discrete-event System object

MATLAB object

Discrete-event System object.

## Output Arguments

**inputTypes** — Input types

cell vector of character vectors

Input port types of a discrete-event system, specified as a cell vector of character vectors with a length that is the same as the number of input ports.

The *N*th element of the vector that specifies the type of the *N*th input port.

- If the port is an entity port, the character vector indicates the entity type name of this port. The name must match one of the entity types specified in `getEntityTypesImpl`.
- If the port is a signal port, the character vector must be empty ( '' ).

**outputTypes** — Output types

cell vector of character vectors

Output port types of a discrete-event system, specified as a cell vector with a length that is the same as the number of output ports.

The *N*th element of the vector that specifies type of the *N*th output port.

- If the port is an entity port, the character vector indicates the entity type name of this port. The name must match one of the entity types specified in `getEntityTypesImpl`.
- If the port is a signal port, the character vector must be empty ( '' ).

## Examples

### Get Entity Inputs and Outputs for Discrete-Event System

Get entity input and output port types for discrete-event system.

```
function [inputTypes,outputTypes] = getEntityPortsImpl(obj)
    % Specify input and output port types.
    %
    % This implementation further specifies port type and entity
    % type at these inputs and outputs:
    % Inputs:
    % 1. Signal port
    % 2. Entity port receiving entities of type 'entity1'
    % 3. Entity port receiving entities of type 'entity2'
    % Outputs:
    % 1. Signal port
    % 2. Entity port sending entities of type 'entity2'
    %
    % The discrete-event system must have already defined:
    % - 3 inputs (by method 'getNumInputsImpl') and
    % - 2 outputs (by method 'getNumOutputsImpl')
    inputTypes = {'', 'entity1', 'entity2'};
    outputTypes = {'', 'entity2'};
end
```

## Version History

Introduced in R2016a

### See Also

[matlab.DiscreteEventSystem](#) | [getEntityStorageImpl](#) | [getEntityTypesImpl](#)

### Topics

“Create Custom Blocks Using MATLAB Discrete-Event System Block”

# getEntityStorageImpl

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Define entity storage elements of discrete-event system

## Syntax

```
[storageSpecs,I,0]=getEntityStorageImpl(obj)
```

## Description

[storageSpecs,I,0]=getEntityStorageImpl(obj) defines entity storage elements of a discrete-event system.

## Input Arguments

**obj** — Discrete-event System object

MATLAB object

Discrete-event System object.

## Output Arguments

**storageSpecs** — Storage specifications

vector of MATLAB structures

Entity storage specifications of a discrete-event system, specified as a vector of MATLAB structures with its length indicating number of entity storage elements of the discrete-event system. The  $N$ th element of the vector defines an entity storage element with index  $N$ . Use utility methods such as queueFIFO to create such definition as a MATLAB structure.

**I** — Connections between input ports and entity storage elements

cell array

Define connections between input ports and entity storage elements as a cell array. The length of the cell array must match the number of input ports of this discrete-event system. The  $N$ th element of the cell array defines the connection between the  $N$ th input port and any entity storage element. If the input port is an entity port, a valid entity storage index must be specified. If the input port is a signal port, the element takes a value of zero.

You can connect multiple entity input ports to a common storage element.

**0** — Connections between output ports and entity storage elements

cell array

Define connections between output ports and entity storage elements as a cell array. The length of the cell array must match the number of output ports of this discrete-event system. The  $M$ th element of the cell array defines the connections between the  $M$ th output port and any entity storage elements. If the output port is an entity port, specify one of these:

- A scalar indicating a single connection from a storage element to the output port.
- A vector indicating multiple connections from multiple storage elements to the output port.

If the output port is a signal port, the element takes a value of zero.

You can connect multiple entity output ports to a common storage element.

## Examples

### Specify Entity Storage Elements

Specify entity storage elements and connections between entity input ports and storage elements for the discrete-event system object.

```
function [storageSpecs, I] = getEntityStorageImpl(obj)
    % Specify entity storage elements and connections between
    % entity input ports and storage elements.
    %
    % The implementation specifies two storage elements for the
    % discrete-event system:
    % 1. A priority queue
    % - Stores entities of type 'student'
    % - Has maximal capacity of 25
    % - Sort entities by an attribute named 'age', in ascending
    %   direction
    % 2. A FIFO queue
    % - Stores entities of type 'student'
    % - Has maximal capacity of 10
    % - Sort entities in a First-In-First-Out order
    %
    % The implementation also specifies that the entity input port
    % of the discrete-event system is connected to the 2nd storage
    % element.
    %
    % Other methods of the discrete-event system must have defined:
    % - An entity type named 'student' (by method 'getEntityTypesImpl')
    % - An entity input port (by method 'getEntityPortsImpl')
    %
    storageSpecs = [...
        obj.queuePriority('student', 25, 'age', 'ascending'), ...
        obj.queueFIFO('student', 10)];
    I = 2;
end
```

## Version History

Introduced in R2016a

### See Also

matlab.DiscreteEventSystem | getEntityTypesImpl | queueFIFO | queueLIFO | queuePriority | queueSysPriority

### Topics

“Delay Entities with a Custom Entity Storage Block”



“Create a Custom Entity Storage Block with Iteration Event”  
“Create Custom Blocks Using MATLAB Discrete-Event System Block”

## getEntityTypesImpl

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Define entity types of discrete-event system

### Syntax

```
entityTypes=getEntityTypesImpl(obj)
```

### Description

`entityTypes=getEntityTypesImpl(obj)` defines entity types of a discrete-event system.

### Input Arguments

**obj** — Discrete-event System object

MATLAB object

Discrete-event System object.

### Output Arguments

**entityTypes** — Entity types

vector of MATLAB structures

Entity types returned as a vector whose length is the same as the number of entity types. Each vector element is a structure containing the entity type properties:

- Name
- Data dimension
- Data type
- Complexity

### Examples

#### Get Entity Types

Get entity types *entity1* and *entity2* for discrete-event system, *obj*.

```
function entityTypes = getEntityTypesImpl(obj)
    % Define entity type 'type1' with inherited data type, dimension
    % and complexity
    t1 = obj.entityType('type1');

    % Define entity type 'type2' with specified data type ('mybus'),
    % default dimension and complexity (i.e. scalar real values)
```

```
t2 = obj.entityType('type2', 'mybus');

% Define entity type 'type3' with specified data type ('double'),
% dimension (2 by 3 matrix), and complexity (complex)
t3 = obj.entityType('type3', 'double', [2 3], true);

entityTypes = [t1, t2, t3];
end
```

## Version History

Introduced in R2016a

### See Also

[matlab.DiscreteEventSystem](#) | [entityType](#) | [getEntityPortsImpl](#) | [getEntityStorageImpl](#)

### Topics

“Create Custom Blocks Using MATLAB Discrete-Event System Block”

# getResourceNamesImpl

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Define resource pools from which to acquire resources

## Syntax

```
resourceNames = getResourceNamesImpl(obj)
```

## Description

`resourceNames = getResourceNamesImpl(obj)` defines resource pools from which the discrete event system acquires resources.

## Input Arguments

**obj** — Discrete-event System object

MATLAB object

Discrete-event System object.

## Output Arguments

**resourceNames** — Resource names

vector of MATLAB structures

Resource pools from which to acquire resources, specified as a vector of MATLAB structures. Use `resourceType` method to create this array.

## Examples

### Define Resource Pools

Use this method together with `resourceType` to specify the resources of types `Test1` and `Test2` to be acquired by the entity type `Part`.

```
function resourceNames = getResourceNamesImpl(obj)
    % Define the names of the resources to be acquired.
    resourceNames = obj.resourceType('Part', {'Test1', 'Test2'}) ;
end
```

## Version History

Introduced in R2019a

**See Also**

matlab.DiscreteEventSystem | eventForward | cancelAcquireResource | resourceAcquired | eventReleaseResource | resourceType

**Topics**

“Create a Custom Resource Acquirer Block”

“Create Custom Blocks Using MATLAB Discrete-Event System Block”

# initEventArray

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Initialize event array

## Syntax

```
event = initEventArray()
```

## Description

`event = initEventArray()` creates an empty array of event structures, to initialize the return of an event action method such as `matlab.DiscreteEventSystem.entry`. This method enables you to append elements to the array in the MATLAB Discrete-Event System block when the Code generation is selected for the **Simulate using** parameter.

## Output Arguments

**event** — Array of event structures

Array

Array of event structures, specified as a MATLAB structures.

## Examples

### Initialize Returned Event Array

Initialize returned event array for the `exit` method.

```
function events = exit(obj, ~, entity, ~)
    events = obj.initEventArray;
    if entity.data == 1
        events = obj.eventTimer('exit', 0);
    end
```

## Version History

Introduced in R2017b

## See Also

`getEntityStorageImpl` | `queueLIFO` | `queuePriority` | `queueSysPriority`

## Topics

“Create Custom Blocks Using MATLAB Discrete-Event System Block”

# initResourceArray

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Initialize a resource specification array

## Syntax

```
resArray = initResourceArray()
```

## Description

`resArray = initResourceArray()` initializes an empty array of `resourceSpecification`. This method enables you to append elements to the array in the MATLAB Discrete-Event System block when you select the Code generation for the **Simulate using** parameter.

## Output Arguments

**resArray — Resource specifications**

Array

Resource specifications specified as an empty array.

## Examples

### Initialize Resource Array

An entity entry to the storage element invokes two timer events. `resourceSpecification` defines the type and the amount of resources an entity acquires. The entity acquires a resource of type `Test1` if the timer with tag `ProcessComplete` expires. The entity acquires a resource of type `Test2` if the timer with tag `TimeOut` expires. The `resRequest` array is initialized by the `initResourceArray` method for code generation.

```
function [entity,event] = entry(obj,storage,entity,source)
% Specify event actions when entity enters storage.
ProcessingTime=randi([1 15]);
% Define two timer events.
event1 = obj.eventTimer('TimeOut',10);
event2 = obj.eventTimer('ProcessComplete', ProcessingTime);
event=[event1 event2];
end

function [entity, event] = timer(obj,storage,entity,tag)
% Specify event actions when a timer expires.
resRequest = obj.initResourceArray();
switch tag
case 'ProcessComplete'
resRequest = obj.resourceSpecification('Test1', 1);
case 'TimeOut'
resRequest = obj.resourceSpecification('Test2', 1);
end
```

```
    event = obj.eventAcquireResource(resRequest, 'MyResourceAcquireEvent');  
end
```

## Version History

Introduced in R2019a

### See Also

[matlab.DiscreteEventSystem](#) | [eventForward](#) | [getResourceNamesImpl](#) | [resourceReleased](#) | [eventReleaseResource](#) | [resourceSpecification](#)

### Topics

“Create a Custom Resource Acquirer Block”

“Create Custom Blocks Using MATLAB Discrete-Event System Block”



# iterate

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Event action when entity iterates

## Syntax

```
[entity,events,next]=iterate(obj,storage,entity,tag,cur)
[entity,events,next,out1,...]=iterate(obj,storage,entity,tag,cur,in1,...)
```

## Description

`[entity,events,next]=iterate(obj,storage,entity,tag,cur)` specifies event actions for when an entity is processed as a part of an iterate event.

`[entity,events,next,out1,...]=iterate(obj,storage,entity,tag,cur,in1,...)` specifies such event actions when the block has one or more input signal ports and/or signal output ports.

## Input Arguments

**obj** — Discrete-event System object

MATLAB object

Discrete-event System object.

**storage** — Storage

double

Index of the storage element.

**entity** — Entity

MATLAB structure

Entity currently being processed. Entity has these fields:

- `sys` (MATLAB structure) — It has these fields:
  - `id` (double) — Entity ID
  - `priority` (double) — Entity priority
- `data` — Entity data

**tag** — Tag

character vector

Tag of the current entity iterate event.

**cur** — Current state

MATLAB structure

MATLAB structure indicating current state of iteration. The structure has these fields:

- **size**  
Total number of entities the storage has
- **position**  
Position of the current iterating entity

**in1 – Signal input**

any value

Any data inputs of the object. These input arguments exist only when the object has data inputs.

**Output Arguments****entity – Entity**

MATLAB structure

Entity being processed, possibly with changed data.

**events – Events**

vector of MATLAB structures

Events to be scheduled after the method returns. Use `matlab.DiscreteEventSystem` class methods to create events. Each event has these fields:

- **type** (character vector) – Type of the event
- **delay** (double) – Delay before the event
- **priority** (double) – Priority of the event
- **Storage** (double) – Index of the storage element
- **tag** (character vector) – Event tag
- **location** (MATLAB structure) – Source or destination location of entity

**next – Iteration**

logical | double

- **True**  
Continue to process the next entity in the storage element.
- **False**  
Terminate the iterate event, and leave the rest of the entities of the storage element unprocessed.

**out1 – Signal output**

any value

Data outputs of the object. You must specify these output arguments when the object has data outputs.

## Examples

### Forward the First Entity

Forward the first entity with matching data value to output port 1 of the discrete-event system.

```
function [entity,events,next] = iterate(obj,storage,entity,tag,status)
    % Forward the first entity with matching data value to output
    % port 1 of the discrete-event system.
    disp(['Searching in storage element ' num2str(storage)]);
    disp(['    Total size = ' num2str(status.size)]);
    disp(['    Current position = ' num2str(status.position)]);
    if (entity.data == obj.dataToSearch)
        events = obj.eventForward('output', 1, 0);
        next = false;    % Found -- early terminate
    else
        events = [];
        next = true;    % Not yet found -- continue
    end
end
```

### Custom Entity Storage Block with Iteration Event

In this example, a custom block allows entities to enter its storage element through its input port. The storage element is a priority queue that sorts the entities based on their Diameter attribute in ascending order. Every entity entry to the block's storage invokes an iteration event to display the diameter and the position of each entity in the storage.

For more information, see “Create a Custom Entity Storage Block with Iteration Event”.

```
classdef CustomEntityStorageBlockIteration < matlab.DiscreteEventSystem

    % A custom entity storage block with one input port and one storage element.

    % Nontunable properties
    properties (Nontunable)
        % Capacity
        Capacity = 5;
    end
    % Create the storage element with one input and one storage.
    methods (Access=protected)

        function num = getNumInputsImpl(obj)
            num = 1;
        end

        function num = getNumOutputsImpl(obj)
            num = 0;
        end

        function entityTypes = getEntityTypesImpl(obj)
            entityType1 = obj.entityType('Wheel');
            entityTypes = entityType1;
        end

        function [inputTypes,outputTypes] = getEntityPortsImpl(obj)
            inputTypes = {'Wheel'};
            outputTypes={};
        end

        function [storageSpecs, I, 0] = getEntityStorageImpl(obj)
            storageSpecs = obj.queuePriority('Wheel',obj.Capacity, 'Diameter','ascending');
            I = 1;
            0 = [];
        end
    end
end
```

```
% Entity entry event action
methods

function [entity, event] = WheelEntry(obj,storage,entity, source)
    % Entity entry invokes an iterate event.
    event = obj.eventIterate(1, '');
end

% The itarate event action
function [entity,event,next] = WheelIterate(obj,storage,entity,tag,cur)
    % Display wheel id, position in the storage, and diameter.
    coder.extrinsic('fprintf');
    fprintf('Wheel id %d, Current position %d, Diameter %d\n', ...
        entity.sys.id, cur.position, entity.data.Diameter);
    if cur.size == cur.position
        fprintf('End of Iteration \n')
    end
    next = true;
    event=[];
end

end

end
```

## Version History

Introduced in R2016a

### See Also

[matlab.DiscreteEventSystem](#) | [blocked](#) | [destroy](#) | [entry](#) | [exit](#) | [generate](#) | [getEntityPortsImpl](#) | [getEntityStorageImpl](#) | [getEntityTypesImpl](#) | [setupEvents](#) | [timer](#)

### Topics

“Create a Custom Entity Storage Block with Iteration Event”

“Create Custom Blocks Using MATLAB Discrete-Event System Block”

# modified

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Event action upon entity modification by the Entity Find block

## Syntax

```
[events] = modified(obj,storage,entity)
[events, out1, ...] = modified(obj,storage,entity,in1,...)
```

## Description

[events] = modified(obj,storage,entity) specifies event actions of the object after an entity is modified.

[events, out1, ...] = modified(obj,storage,entity,in1,...) specifies event actions of the object when the block has one or more input signal ports and/or signal output ports.

## Input Arguments

### **obj** — Discrete-event System object

MATLAB object

Discrete-event System object.

### **storage** — Storage

double

Index of the storage element where the entity is being modified.

### **entity** — Entity

MATLAB structure

Entity that is being modified. Entity has these fields:

- **sys** (MATLAB structure) — It has these fields:
  - **id** (double) — Entity ID
  - **priority** (double) — Entity priority
- **data** — Entity data

### **in1** — Signal input

any value

Any data inputs of the object. These input arguments exist only when the object has data inputs.

## Output Arguments

### events — Events

vector of MATLAB structures

Events to be scheduled after the method returns. Use `matlab.DiscreteEventSystem` class methods to create events. Each event has these fields:

- `type` (character vector) — Type of the event
- `delay` (double) — Delay before the event
- `priority` (double) — Priority of the event
- `Storage` (double) — Index of the storage element
- `tag` (character vector) — Event tag
- `location` (MATLAB structure) — Source or destination location of entity

### out1 — Signal output

any value

Data outputs of the object. You must specify these output arguments when the object has data outputs.

## Examples

### Event Action Upon Entity Modification

Specify event action to be performed after entity modification in a storage

```
function events = modified(obj,storage,entity)
    events = [];
    % If the delay attribute of the entity exceeds 100, destroy the entity
    if entity.data.delay > 100
        events = obj.destroy();
    end
end
```

## Version History

Introduced in R2018b

### See Also

`matlab.DiscreteEventSystem` | `blocked` | `destroy` | `entry` | `generate` | `iterate` | `setupEvents` | `timer`

### Topics

“Create Custom Blocks Using MATLAB Discrete-Event System Block”

# queueFIFO

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Define first-in first-out (FIFO) queue storage

## Syntax

```
storage = queueFIFO(entityType,capacity)
```

## Description

`storage = queueFIFO(entityType,capacity)` defines a FIFO queue storage element. Use this function when implementing the `getEntityStorageImpl` method.

---

**Note** This page describes the `queueFIFO` method to create a custom discrete-event system block. See Entity Queue block provided by the SimEvents library to use a queue with FIFO sorting policy.

---

## Input Arguments

### **entityType** – Entity type

character vector

Type of entities that the new storage element works with.

### **capacity** – Maximum number of entities

double

Maximum number of entities that the storage can contain, specified as a double.

## Output Arguments

### **storage** – Storage

MATLAB structure

Queue storage that contains entities and sorts them in FIFO order.

## Examples

### **Specify FIFO Queue Entity Storage**

Specify FIFO queue entity storage for the discrete-event system object.

```
% Define a storage element as a FIFO queue
% - Entities in the queue are sorted in First-In-First-Out (FIFO) order
% - Queue can store entities of type 'myEntity'
```

```
% - Queue can store no more than 25 entities
storage = obj.queueFIFO('myEntity', 25);
```

## Create a Custom Entity Storage Block to Delay Entities

This example shows how to use discrete-event System object methods to create a custom entity storage block that has one input port, one output port, and one storage element. The discrete-event System object is the instantiation of the `matlab.DiscreteEventSystem` class, which allows you to use the implementation and service methods provided by this class. Then, you use the MATLAB Discrete-Event System block to integrate the System object into a SimEvents model. The custom MATLAB Discrete-Event System block accepts an entity from its input port and forwards it to its output port with a specified delay. For more information, see “Delay Entities with a Custom Entity Storage Block”.

```
classdef CustomEntityStorageBlock < matlab.DiscreteEventSystem

    % A custom entity storage block with one input, one output, and one storage.

    % Nontunable properties
    properties (Nontunable)
        % Capacity
        Capacity = 1;
        % Delay
        Delay=4;
    end

    methods (Access=protected)
        function num = getNumInputsImpl(~)
            num = 1;
        end

        function num = getNumOutputsImpl(~)
            num = 1;
        end

        function entityType = getEntityTypesImpl(obj)
            entityType = obj.entityType('Car');
        end

        function [inputTypes,outputTypes] = getEntityPortsImpl(obj)
            inputTypes = {'Car'};
            outputTypes = {'Car'};
        end

        function [storageSpecs, I, O] = getEntityStorageImpl(obj)
            storageSpecs = obj.queueFIFO('Car', obj.Capacity);
            I = 1;
            O = 1;
        end

    end

    methods

        function [entity,event] = CarEntry(obj,storage,entity,source)
            % Specify event actions when entity enters storage.

            event = obj.eventForward('output', 1, obj.Delay);

        end

    end
end
```

## Version History

Introduced in R2016a



**See Also**

[getEntityStorageImpl](#) | [queueLIFO](#) | [queuePriority](#) | [queueSysPriority](#)

**Topics**

[“Delay Entities with a Custom Entity Storage Block”](#)

[“Create a Custom Entity Storage Block with Iteration Event”](#)

[“Create Custom Blocks Using MATLAB Discrete-Event System Block”](#)

## queueLIFO

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Define last-in last-out (LIFO) stack storage

### Syntax

```
storage=queueLIFO(entityType,capacity)
```

### Description

`storage=queueLIFO(entityType,capacity)` defines a LIFO stack storage element. Use this function when implementing the `getEntityStorageImpl` method.

### Input Arguments

**entityType — Entity type**

character vector

Type of entities that the new storage element works with.

**capacity — Capacity**

double

Maximum number of entities that the storage can contain, specified as a double.

### Output Arguments

**storage — Storage**

MATLAB structure

Stack storage that contains entities and sorts them in a LIFO order.

### Examples

**Define LIFO Stack Storage**

Define LIFO stack storage.

```
% Define a storage element as a LIFO queue  
% - Entities in the queue are sorted in Last-In-First-Out (LIFO) order  
% - Queue can store entities of type 'myEntity'
```

```
% - Queue can store no more than 25 entities  
storage = obj.queueLIFO('myEntity', 25);
```

## Version History

Introduced in R2016a

### See Also

[getEntityStorageImpl](#) | [queueFIFO](#) | [queuePriority](#) | [queueSysPriority](#)

### Topics

“Create Custom Blocks Using MATLAB Discrete-Event System Block”

# queuePriority

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Define priority queue storage

## Syntax

```
storage=queuePriority(entityType,capacity,key,order)
```

## Description

`storage=queuePriority(entityType,capacity,key,order)` defines a priority queue that sorts entities by custom attribute. Use this function when implementing the `getEntityStorageImpl` method.

## Input Arguments

### **entityType** — Entity type

character vector

Type of entities that the new storage element works with.

### **capacity** — Capacity

double

Maximum number of entities that the storage can contain, specified as a double.

### **key** — Key

character vector

Name of the attribute that is used as the key for sorting.

### **order** — Sorting order

character vector

Direction of sorting. Specify `'ascending'` if you want entities with smaller key values to appear in front of the queue. Specify `'descending'` if you want entities with greater key values to appear in front of the queue.

## Output Arguments

### **storage** — Storage

MATLAB structure

Queue storage element that contains entities and sorts them using a custom attribute.

## Examples

### Define Storage Element as a Priority Queue

Define storage element as a priority queue.

```
% Define a storage element as a priority queue
% - Queue sorts entities using a specific attribute of the entities
% - Queue can store entities of type 'myEntity'
% - Queue can store no more than 25 entities
% - Queue uses the attribute 'age' to sort entities
% - Sorting direction is 'ascending', resulting entities with
%   smaller 'age' attribute values to appear in front of the queue
storage = obj.queuePriority('myEntity', 25, 'age', 'ascending');
```

### Custom Entity Storage Block with Iteration Event

In this example, a custom block allows entities to enter its storage element through its input port. The storage element is a priority queue that sorts the entities based on their Diameter attribute in ascending order. Every entity entry to the block's storage invokes an iteration event to display the diameter and the position of each entity in the storage.

For more information, see “Create a Custom Entity Storage Block with Iteration Event”.

```
classdef CustomEntityStorageBlockIteration < matlab.DiscreteEventSystem

    % A custom entity storage block with one input port and one storage element.

    % Nontunable properties
    properties (Nontunable)
        % Capacity
        Capacity = 5;
    end
    % Create the storage element with one input and one storage.
    methods (Access=protected)

        function num = getNumInputsImpl(obj)
            num = 1;
        end

        function num = getNumOutputsImpl(obj)
            num = 0;
        end

        function entityType = getEntityTypesImpl(obj)
            entityType1 = obj.entityType('Wheel');
            entityType = entityType1;
        end

        function [inputTypes,outputTypes] = getEntityPortsImpl(obj)
            inputTypes = {'Wheel'};
            outputTypes={};
        end

        function [storageSpecs, I, O] = getEntityStorageImpl(obj)
            storageSpecs = obj.queuePriority('Wheel',obj.Capacity, 'Diameter','ascending');
            I = 1;
            O = [];
        end

    end

    % Entity entry event action
    methods

        function [entity, event] = WheelEntry(obj,storage,entity, source)
            % Entity entry invokes an iterate event.
            event = obj.eventIterate(1, '');
        end

    end

end
```

```
% The itarate event action
function [entity,event,next] = WheelIterate(obj,storage,entity,tag,cur)
    % Display wheel id, position in the storage, and diameter.
    coder.extrinsic('fprintf');
    fprintf('Wheel id %d, Current position %d, Diameter %d\n', ...
        entity.sys.id, cur.position, entity.data.Diameter);
    if cur.size == cur.position
        fprintf('End of Iteration \n')
    end
    next = true;
    event=[];
end
end
end
```

## Version History

Introduced in R2016a

### See Also

[getEntityStorageImpl](#) | [queueFIFO](#) | [queueLIFO](#) | [queueSysPriority](#)

### Topics

“Create a Custom Entity Storage Block with Iteration Event”

“Create Custom Blocks Using MATLAB Discrete-Event System Block”

# queueSysPriority

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Define system priority queue storage

## Syntax

```
storage=queueSysPriority(entityType,capacity,order)
```

## Description

`storage=queueSysPriority(entityType,capacity,order)` defines a priority queue storage element that sorts entities by their system priorities. Use this method when implementing the `getEntityStorageImpl` method.

## Input Arguments

### **entityType** – Entity type

character vector

Type of entities that the new storage element works with.

### **capacity** – Capacity

double

Maximum number of entities that the storage can contain, specified as a double.

### **order** – Sorting order

character vector

Direction of sorting. Specify 'ascending' if you want entities with smaller system priority values (higher priority) to appear in front of the queue. Use 'descending' if you want entities with higher system priority values (lower priority) to appear in front of the queue.

## Output Arguments

### **storage** – Storage

MATLAB structure

Queue storage element that contains entities and sorts them by the entities' system priorities.

## Examples

### **Define Storage Element as System Priority Queue**

Define a storage element that uses an entity system priority for sorting.

```
% - Queue sorts entities using entity priority (i.e.  
%   the field 'entVar.sys.priority' on a MATLAB variable 'entVar'  
%   representing a SimEvents entity)  
% - Queue can store entities of type 'myEntity'  
% - Queue can store no more than 25 entities  
% - Sorting direction is 'ascending', resulting entities with  
%   higher priority (or smaller entity priority values) to appear  
%   in the front of the queue  
storage = obj.queueSysPriority('myEntity', 25, 'ascending');
```

## **Version History**

**Introduced in R2016a**

### **See Also**

[getEntityStorageImpl](#) | [queueFIFO](#) | [queueLIFO](#) | [queuePriority](#)

### **Topics**

“Create Custom Blocks Using MATLAB Discrete-Event System Block”



# resourceAcquired

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Event action upon successful resource acquisition

## Syntax

```
[entity,event,out1,...] = resourceAcquired(obj,storage,entity,resources,tag,
in1,...)
```

## Description

[entity,event,out1,...] = resourceAcquired(obj,storage,entity,resources,tag,in1,...) specifies event action for a discrete-event System object upon successful acquisition of a resource. Resource acquisition is successful only if all of the specified resources are acquired.

## Input Arguments

**obj** — Discrete-event System object

MATLAB object

Discrete-event System object.

**storage** — Storage that the entity resides in

double

Index of the storage element.

**entity** — Entity that is acquiring the resources

MATLAB structure

Entity that acquires the resource. Entity has these fields:

- sys (MATLAB structure) consisting of:
  - id (double) — Entity ID
  - priority (double) — Entity priority
- data — Entity data

**resources** — Acquired resources

MATLAB structure

An array of structures that specifies the resources that have been acquired.

**tag** — Tag of the resource acquisition event

character vector

Tag of the currently executing resource acquisition event.

**in1 – First data input**

character vector

First data input.

**Output Arguments****entity – Entity with changed value**

MATLAB structure

Entity acquiring the resource.

**event – Events to be scheduled**

vector of MATLAB structures

Events to be scheduled. Use `matlab.DiscreteEventSystem` class methods to create events. Each event has these fields:

- `type` (character vector) — Type of the event
- `delay` (double) — Delay before the event
- `priority` (double) — Priority of the event
- `storage` (double) — Index of the storage element
- `tag` (character vector) — Event tag
- `location` (MATLAB structure) — Source or destination (see “source” on page 1-0 )

**out1 – First data output**

character vector

First data output.

**Examples****Event Action on Resource Acquisition**

Suppose that an entity acquires resources successfully with a scheduled `eventAcquireResource` and the tag of this event is `MyResourceAcquireEvent`. Then this acquisition invokes the `resourceAcquired` method to forward entities to the output.

```
function [entity,events] = entry(obj, storage, entity, source)
    % On entry, acquire one resource of type Resource1.
    resRequest = obj.resourceSpecification('Resource1', 1);
    events = obj.eventAcquireResource(resRequest, 'MyResourceAcquireEvent');
end

function [entity,events] = resourceAcquired(obj, storage,...
    entity, resources, tag)
    % After resource acquisition, forward the entity to the output.
    events = obj.eventForward('output', storage, 0.0);
end
```

**Custom Resource Acquirer**

This example shows how to use resource management methods to create a custom entity storage block in which entities acquire resources from specified Resource Pool blocks.

Suppose that you manage a facility that produces parts from two different materials, material 1 and material 2, to fulfill orders. After a part is produced, it is evaluated for quality assurance.

Two testing methods for quality control are:

- Test 1 is used for parts that are produced from material 1.
- Test 2 is used for parts that are produced from material 2

After the production phase, parts are tagged based on their material to apply the correct test.

For more information, see “Create a Custom Resource Acquirer Block”.

```
classdef CustomBlockAcquireResources < matlab.DiscreteEventSystem
    % Custom resource acquire block example.

    methods(Access = protected)

        function num = getNumInputsImpl(obj)
            num = 1;
        end

        function num = getNumOutputsImpl(obj)
            num = 1;
        end

        function entityTypes = getEntityTypesImpl(obj)
            entityTypes(1) = obj.entityType('Part');
        end

        function [input, output] = getEntityPortsImpl(obj)
            input = {'Part'};
            output = {'Part'};
        end

        function [storageSpec, I, O] = getEntityStorageImpl(obj)
            storageSpec(1) = obj.queueFIFO('Part', 1);
            I = 1;
            O = 1;
        end

        function resNames = getResourceNamesImpl(obj)
            % Define the names of the resources to be acquired.
            resNames = obj.resourceType('Part', {'Test1', 'Test2'});
        end

    end

    methods

        function [entity,events] = entry(obj, storage, entity, source)
            % On entity entry, acquire a resource from the specified pool.
            if entity.data.Test == 1
                % If the entity is produced from Material1, request Test1.
                resReq = obj.resourceSpecification('Test1', 1);
            else
                % If the entity is produced from Material2, request Test2.
                resReq = obj.resourceSpecification('Test2', 1);
            end
            % Acquire the resource from the corresponding pool.
            events = obj.eventAcquireResource(resReq, 'TestTag');
        end

        function [entity,events] = resourceAcquired(obj, storage,...
            entity, resources, tag)
            % After the resource acquisition, forward the entity to the output.
            events = obj.eventForward('output', storage, 0.0);
        end

    end

end
```

## **Version History**

**Introduced in R2019a**

### **See Also**

`matlab.DiscreteEventSystem` | `eventForward` | `cancelAcquireResource` | `getResourceNamesImpl` | `resourceReleased` | `eventAcquireResource`

### **Topics**

“Create a Custom Resource Acquirer Block”

“Create Custom Blocks Using MATLAB Discrete-Event System Block”

# resourceReleased

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Event action upon successful resource release

## Syntax

```
[entity,event,out1,...] = resourceReleased(obj,storage,entity,resources,tag, in1,...)
```

## Description

[entity,event,out1,...] = resourceReleased(obj,storage,entity,resources,tag, in1,...) specifies event actions for a discrete-event System object upon successful resource release.

## Input Arguments

**obj** — Discrete-event System object

MATLAB object

Discrete-event System object.

**storage** — Storage

double

Index of the storage element.

**entity** — Entity

MATLAB structure

Entity releasing the resource. Entity has these fields:

- **sys** (MATLAB structure) consisting of:
  - **id** (double) — Entity ID
  - **priority** (double) — Entity priority
- **data** — Entity data

**resources** — Released resources

MATLAB structure

An array of structures that specifies the resources that have been released.

**tag** — Tag of the resource release event

character vector

Tag of the currently executing resource release event.

**in1 – First data input**

character vector

First data input.

**Output Arguments****entity – Entity**

MATLAB structure

Entity releasing the resource.

**event – Event**

vector of MATLAB structures

Events to be scheduled. Use `matlab.DiscreteEventSystem` class methods to create events. Each event has these fields:

- `type` (character vector) — Type of the event
- `delay` (double) — Delay before the event
- `priority` (double) — Priority of the event
- `storage` (double) — Index of the storage element
- `tag` (character vector) — Event tag
- `location` (MATLAB structure) — Source or destination location of entity (see “source” on page 1-0 )

**out1 – First data output**

character vector

First data output.

**Examples****Event Action on Resource Release**

Suppose that an entity releases resources successfully with a scheduled `eventReleaseResource` method and the tag of this event is `MyResourceAcquireEvent`. The successful release of the resources invokes the `resourceReleased` method to forward the entity to the output.

```
function [entity,events] = entry(obj, storage, entity, source)
% On entry, release one resource of type Resource1.
resRequest = obj.resourceSpecification('Resource1', 1);
events = obj.eventReleaseResource(resRequest, 'MyResourceAcquireEvent');
end

function [entity,events] = resourceReleased(obj, storage,...
entity, resources, tag)
% After resource release, forward the entity to the output.
events = obj.eventForward('output', storage, 0.0);
end
```

**Version History**

Introduced in R2019a

**See Also**

`matlab.DiscreteEventSystem` | `eventForward` | `cancelAcquireResource` | `getResourceNamesImpl` | `resourceAcquired` | `eventReleaseResource`

**Topics**

“Create a Custom Resource Acquirer Block”

“Create Custom Blocks Using MATLAB Discrete-Event System Block”

## resourceSpecification

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Create specifications for a resource acquisition or a resource release event

### Syntax

```
resRequest = resourceSpecification(resource,amount)
```

### Description

`resRequest = resourceSpecification(resource,amount)` specifies the names and the amount of the resources for the `eventAcquireResource` or the `eventReleaseResource` requests.

For example, this code specifies one unit of `Resource1` and two units of `Resource2`.

```
resRequest = obj.resourceSpecification('Resource1', 1);  
resRequest = [resReq obj.resourceSpecification('Resource2', 2)];
```

If you specify an amount for the `eventReleaseResource` larger than the amount that was acquired earlier, all the previously acquired resources are released.

For example, suppose that an entity previously acquired three resources of type `Resource1` and four resources of type `Resource2`. This code specifies the amounts for `eventReleaseResource`.

```
resRequest = obj.resourceSpecification('Resource1', 2);  
resRequest = [resReq obj.resourceSpecification('Resource2', 5)];  
event = eventReleaseResource(resReq, 'relinquish');
```

After the release, the entity has one resource of type `Resource1` and zero resources of type `Resource2`.

You can specify the release of all previously acquired resources by using `eventReleaseAllResources`.

### Input Arguments

**resource** — Specify the name of resources for acquisition or release requests

character vector

Specify the name of the resources for the `eventAcquireResource` or the `eventReleaseResource` requests. You can specify more than one resource.

**amount** — Specify the amount of resources for acquisition or release requests

double

Specify the amount of resources for the `eventAcquireResource` or the `eventReleaseResource` requests.



## Output Arguments

### resRequest — Resource request

vector of MATLAB structures

Resource request for an acquisition or a release event specified as a vector of MATLAB structures.

## Examples

### A Simple Resource Specification Example

When an entity enters the storage element, it acquires resources. The entity acquires one resource of type Resource1 and one resource of type Resource2, which are defined as specifications. The specifications are then used for eventAcquireResource with tag MyResourceAcquireEvent.

```
function [entity,events] = entry(obj, storage, entity, source)
    % On entry, acquire one resource of type Resource1 and one resource of type Resource2.
    resRequest(1) = obj.resourceSpecification('Resource1', 1);
    resRequest(2) = obj.resourceSpecification('Resource2', 1);
    events = obj.eventAcquireResource(resRequest, 'MyResourceAcquireEvent');
end
```

### Resource Specification in a Custom Resource Acquirer Block

This example shows how to use resource management methods to create a custom entity storage block in which entities acquire resources from specified Resource Pool blocks.

Suppose that you manage a facility that produces parts from two different materials, material 1 and material 2, to fulfill orders. After a part is produced, it is evaluated for quality assurance.

Two testing methods for quality control are:

- Test 1 is used for parts that are produced from material 1.
- Test 2 is used for parts that are produced from material 2

After the production phase, parts are tagged based on their material to apply the correct test.

For more information, see “Create a Custom Resource Acquirer Block”.

```
classdef CustomBlockAcquireResources < matlab.DiscreteEventSystem
    % Custom resource acquire block example.

    methods(Access = protected)

        function num = getNumInputsImpl(obj)
            num = 1;
        end

        function num = getNumOutputsImpl(obj)
            num = 1;
        end

        function entityTypes = getEntityTypesImpl(obj)
            entityTypes(1) = obj.entityType('Part');
        end

        function [input, output] = getEntityPortsImpl(obj)
            input = {'Part'};
            output = {'Part'};
        end

        function [storageSpec, I, O] = getEntityStorageImpl(obj)
            storageSpec(1) = obj.queueFIFO('Part', 1);
            I = 1;
        end
    end
end
```

```
    0 = 1;
end

function resNames = getResourceNamesImpl(obj)
    % Define the names of the resources to be acquired.
    resNames = obj.resourceType('Part', {'Test1', 'Test2'});
end

end

methods

function [entity,events] = entry(obj, storage, entity, source)
    % On entity entry, acquire a resource from the specified pool.
    if entity.data.Test == 1
        % If the entity is produced from Material1, request Test1.
        resReq = obj.resourceSpecification('Test1', 1);
    else
        % If the entity is produced from Material2, request Test2.
        resReq = obj.resourceSpecification('Test2', 1);
    end
    % Acquire the resource from the corresponding pool.
    events = obj.eventAcquireResource(resReq, 'TestTag');
end

function [entity,events] = resourceAcquired(obj, storage,...
    entity, resources, tag)
    % After the resource acquisition, forward the entity to the output.
    events = obj.eventForward('output', storage, 0.0);
end

end

end
```

## Version History

Introduced in R2019a

### See Also

matlab.DiscreteEventSystem | eventForward | cancelAcquireResource |  
getResourceNamesImpl | resourceAcquired | eventReleaseResource |  
eventAcquireResource

### Topics

“Create a Custom Resource Acquirer Block”

“Create Custom Blocks Using MATLAB Discrete-Event System Block”

# resourceType

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Specify an entity type and the name of the resources to be acquired by the specified entity

## Syntax

```
resType = resourceType(entityType,resourceNames)
```

## Description

`resType = resourceType(entityType,resourceNames)` specifies an entity type and the corresponding resources that this entity type acquires.

## Input Arguments

### entityType — Name of the entity

character vector

The entity type name used in a discrete-event system. For more information, see `getEntityTypesImpl`.

### resourceNames — Name of the resource the entity acquires

character vector

Array of resources from which the system intends to acquire resources for the defined entity type.

## Output Arguments

### resType — Resource Type

vector of MATLAB structures

Resource types returned as a vector.

## Examples

Use this method together with `getResourceNamesImpl` to specify the resources of types `Test1` and `Test2` to be acquired by the entity type `Part`.

```
function resNames = getResourceNamesImpl(obj)
    % Define the names of the resources to be acquired.
    resType = obj.resourceType('Part', {'Test1', 'Test2'}) ;
end
```

## Version History

Introduced in R2019a

### **See Also**

`matlab.DiscreteEventSystem` | `eventForward` | `cancelAcquireResource` |  
`getResourceNamesImpl` | `resourceAcquired` | `eventReleaseResource` |  
`eventAcquireResource`

### **Topics**

“Create a Custom Resource Acquirer Block”

“Create Custom Blocks Using MATLAB Discrete-Event System Block”

# setupEvents

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Initialize entity generation events

## Syntax

```
events=setupEvents(obj)  
[events,out1,...]=setupEvents(obj)
```

## Description

`events=setupEvents(obj)` sets up the first set of entity generation events at the start of simulation.

`[events,out1,...]=setupEvents(obj)` specifies such event actions of the object when the block has one or more signal output ports.

## Input Arguments

**obj** — Discrete-event System object

MATLAB object

Discrete-event System object.

## Output Arguments

**events** — Events

vector of MATLAB structures

A vector of events to create initial entities. The discrete-event system schedules these events at the start of simulation.

**out1** — Data output

any value

Data outputs of the object. You must specify these output arguments when the object has data outputs.

## Examples

### Schedule Two Entity Generation Events

Schedules two entity generation events at the start of the simulation

```
function events = setupEvents(obj)  
    % Schedules two entity generation events at the start of the
```

```
% simulation
% - An event with tag 'Adam' to generate an entity in storage element 1.
% - An event with tag 'Eve' to generate an entity in storage element 2.
events = [...
    obj.eventGenerate(1, 'Adam', 0.5, 200), ...
    obj.eventGenerate(2, 'Eve', 0.8, 100)];
end
```

## Version History

Introduced in R2016a

### See Also

[matlab.DiscreteEventSystem](#) | [blocked](#) | [entry](#) | [exit](#) | [generate](#) | [iterate](#) | [timer](#)

### Topics

“Create Custom Blocks Using MATLAB Discrete-Event System Block”

# testEntry

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Event action to accept or refuse entity

## Syntax

```
accept = testEntry(obj,storage,entity,source,in1,...)
```

## Description

`accept = testEntry(obj,storage,entity,source,in1,...)` specifies if the storage can accept entity.

## Input Arguments

**obj** — Discrete-event System object

MATLAB object

Discrete-event System object.

**storage** — Storage

double

Index of the storage element.

**entity** — Entity

MATLAB structure

Entity entering storage component. Entity has these fields:

- `sys` (MATLAB structure) — It has these fields:
  - `id` (double) — Entity ID
  - `priority` (double) — Entity priority
- `data` — Entity data

**source** — Source location

double

Source location of entity, such as an input port or a storage element. It has these fields:

- `type` (character vector) — Specify input or storage
- `index` (double) — Input or storage index

**in1** — Signal input

any value

Any data inputs of the object. These input arguments exist only when the object has data inputs.

## Output Arguments

### **accept** — Accept or Refuse

boolean

Storage accepts entity if `true`. Otherwise, if `false`, it does not accept the entity.

## Examples

### Event Action to Accept or Refuse Entity

Accept or refuse an entity entering the storage.

```
function bool = testEntry(obj,storage,entity,src)
    % Test if entity is accepted
    bool = obj.isEntityAcceptable(obj, entity);
end
```

## Version History

Introduced in R2018a

### See Also

`matlab.DiscreteEventSystem | iterate`

### Topics

“Create Custom Blocks Using MATLAB Discrete-Event System Block”



# timer

**Class:** matlab.DiscreteEventSystem

**Package:** matlab

Event action when timer completes

## Syntax

```
[entity,events]=timer(obj,storage,entity,tag)
[entity,events,out1,...]=timer(obj,storage,entity,tag,in1,...)
```

## Description

[entity,events]=timer(obj,storage,entity,tag) specifies event actions for when scheduled timer completes.

[entity,events,out1,...]=timer(obj,storage,entity,tag,in1,...) specifies such event actions when the block has one or more input signal ports and/or signal output ports.

## Input Arguments

**obj** — Discrete-event System object

MATLAB object

Discrete-event System object.

**storage** — Storage

double

Index of the storage element.

**entity** — Entity

MATLAB structure

Entity for the timer event. Entity has these fields:

- sys (MATLAB structure) — It has these fields:
  - id (double) — Entity ID
  - priority (double) — Entity priority
- data — Entity data

**tag** — Tag

character vector

Tag of the currently executing timer event.

**in1** — Signal input

any value

Any data inputs of the object. These input arguments exist only when the object has data inputs.

## Output Arguments

### **entity** – Entity

MATLAB structure

Entity with changed value.

### **events** – Events

vector of MATLAB structures

Events to be scheduled after the method returns. Use `matlab.DiscreteEventSystem` class methods to create events. Each event has these fields:

- `type` (character vector) – Type of the event
- `delay` (double) – Delay before the event
- `priority` (double) – Priority of the event
- `Storage` (double) – Index of the storage element
- `tag` (character vector) – Event tag
- `location` (MATLAB structure) – Source or destination location of entity

### **out1** – Signal output

any value

Data outputs of the object. You must specify these output arguments when the object has data outputs.

## Examples

### Event Action When Timer Completes

Forward entity when timer completes for discrete-event system object *obj*.

```
function [entity,events] = timer(obj,storage,entity,tag)
    % Check which timer of the entity has expired, and forward the
    % entity to the next location accordingly.
    switch tag
        case 'ServiceComplete'
            entity.done = 1;
            events = obj.eventForward('output', 1, 0);
        case 'Timeout'
            entity.done = 0;
            events = obj.eventForward('storage', 2, 0);
    end
end
```

### Custom Entity Storage Block with Timer Events

This example uses a custom entity storage block with one input, two outputs, and a storage element. An entity of type `Part` with `TimeOut` attribute enters the storage of the custom block to be processed. `TimeOut` determines the maximum allowed processing time of the parts. When a part enters the storage, two timer events are activated. One timer tracks the processing time of the part in

the oven. When this timer expires, the entity is forwarded to output 1. Another timer acts as a fail-safe and tracks if the maximum allowed processing time is exceeded or not. When this timer expires, the process is terminated and the entity is forwarded to the output 2.

For more information, see “Custom Entity Storage Block with Multiple Timer Events”.

```
classdef CustomEntityStorageBlockTimer < matlab.DiscreteEventSystem

    % A custom entity storage block with one input port, two output ports, and one storage.

    % Nontunable properties
    properties (Nontunable)
        % Capacity
        Capacity = 1;
    end

    methods (Access=protected)

        function num = getNumInputsImpl(~)
            num = 1;
        end

        function num = getNumOutputsImpl(~)
            num = 2;
        end

        function entityTypes = getEntityTypesImpl(obj)
            entityTypes = obj.entityType('Part');
        end

        function [inputTypes,outputTypes] = getEntityPortsImpl(obj)
            inputTypes = {'Part'};
            outputTypes = {'Part' 'Part'};
        end

        function [storageSpecs, I, O] = getEntityStorageImpl(obj)
            storageSpecs = obj.queueFIFO('Part', obj.Capacity);
            I = 1;
            O = [1 1];
        end

    end

    methods

        function [entity,event] = PartEntry(obj,storage,entity,source)
            % Specify event actions when entity enters storage.
            ProcessingTime=randi([1 15]);
            event1 = obj.eventTimer('TimeOut', entity.data.TimeOut);
            event2 = obj.eventTimer('ProcessComplete', ProcessingTime);
            event = [event1 event2];
        end

        function [entity, event] = timer(obj,storage,entity,tag)
            % Specify event actions for when scheduled timer completes.
            event = obj.initEventArray;
            switch tag
                case 'ProcessComplete'
                    event = obj.eventForward('output', 1, 0);
                case 'TimeOut'
                    event = obj.eventForward('output', 2, 0);
            end
        end

    end

end

end
```

## Version History

### Introduced in R2016a

**See Also**

matlab.DiscreteEventSystem | blocked | destroy | entry | exit | generate |  
getEntityTypesImpl | iterate | setupEvents

**Topics**

“Custom Entity Storage Block with Multiple Timer Events”

“Create Custom Blocks Using MATLAB Discrete-Event System Block”

# simevents

Open SimEvents library

## Syntax

simevents

## Description

simevents opens the main SimEvents library.

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

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

In SimEvents:

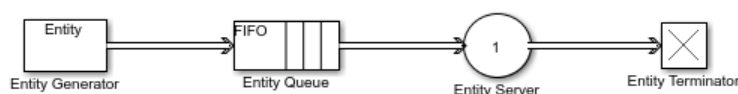
- Entity — A discrete item or object of interest based on the application domain. For example an entity can represent vehicles arriving at a gas station, messages within a communication network, planes on a runway, or trains within a signaling system.
- Event — Asynchronous discrete incidents. For example, an event can represent an entity entry to a block or entity departure from a block.
- Event Action — A custom action invoked by an event. You can customize event actions using MATLAB code that performs calculations and Simulink function calls.

SimEvents blocks can produce, process, and route entities. The blocks can also attach data to entities and manipulate entity data using event actions.

## Examples

### Create A Simple Queuing System

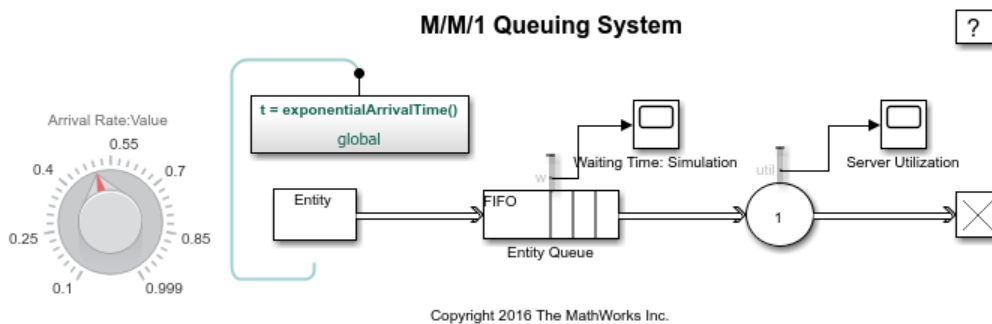
Open the SimEvents library and use the blocks from the library to build a queuing system that generates entities, queues them in a specified order, services them to change their attributes, and terminates them to represent their departure. For more information, see “A Simple Queuing System”.



```
simevents;
```

### Model an M/M/1 Queuing System

Open the SimEvents library and use the blocks to model a single-queue single-server system with a single traffic source and an infinite storage capacity. In the notation, the M stands for Markovian; M/M/1 means that the system has a Poisson arrival process, an exponential service time distribution, and one server. Queuing theory provides exact theoretical results for some performance measures of an M/M/1 queuing system and this model makes it easy to compare empirical results with the corresponding theoretical results. For more information, see “M/M/1 Queuing System”.



```
simevents;
```

## Version History

Introduced in R2011b

### See Also

Entity Queue | Entity Server | Entity Generator | matlab.DiscreteEventSystem | entityType

### Topics

- “Create a Hybrid Model with Time-Based and Event-Based Components”
- “Discrete-Event Simulation in Simulink Models”
- “Create a Discrete-Event Model”
- “M/M/1 Queuing System”

# simeventslib

Open legacy SimEvents library

## Syntax

```
simeventslib
```

## Description

simeventslib opens the version 4.4.1 of the legacy SimEvents library.

---

**Note** This page is for the legacy library. To open the new SimEvents library that is introduced in R2016a, see [simevents](#).

---

## Version History

**Introduced before R2006a**

## See Also

[simevents](#)

## Topics

“Migration Considerations”

## simevents.SimulationObserver class

**Package:** simevents

**Superclasses:** handle

Interface to create your custom observer for models with SimEvents blocks

### Description

This class is an interface for creating custom observers for models with SimEvents blocks. Subclass this class to create your own observer, using the methods below. Some utility functions are also provided to interact with event calendars, blocks, and entities. Do not overwrite these utility functions.

### Class Attributes

Abstract	false
HandleCompatible	true
StrictDefaults	false

For information on class attributes, see “Class Attributes”.

### Creation

`obj = SimulationObserver(modelName)` returns an object of the `SimulationObserver` class, used to create a model observer for a SimEvents model.

### Input Arguments

#### **modelName — Model to observe**

character vector

The name of the model to observe.

### Methods

#### **Public Methods**

<code>simStarted</code>	Specify behavior when simulation starts
<code>simPaused</code>	Specify behavior when simulation pauses
<code>simResumed</code>	Specify behavior when simulation resumes
<code>simTerminating</code>	Define observer behavior when simulation is terminating
<code>getBlocksToNotify</code>	Specify list of blocks to be notified of entity entry and exit events
<code>notifyEventCalendarEvents</code>	Specify whether you want notification for all events in event calendar
<code>postEntry</code>	Specify behavior after an entity enters a block that has entity storage
<code>preExit</code>	Specify behavior before an entity exits a block with entity storage
<code>preExecute</code>	Specify behavior before execution of an event

#### **Protected Methods**

<code>addBlockNotification</code>	Add block to list of blocks to be notified
-----------------------------------	--



removeBlockNotification	Remove block from list of blocks being notified
getEventCalendars	Get handles to event calendars
getAllBlockWithStorages	Get list of blocks that store entities
getHandleToBlock	Return block handle for a given block path
getHandlesToBlockStorages	Return storage handles of specified block

## Examples

### Construct Animator

This example shows how to construct an animator.

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

### Create an observer to count entity departures and acquire departure timestamps

This example shows how to create a simulation observer object and use it to observe entities in a model. For more information, see “Observe Entities Using simevents.SimulationObserver Class”.

Create the observer.

```
classdef myObserverPreexit < simevents.SimulationObserver
    % Add the observer properties.
    properties
        Model
        % Initialize the property count.
        count
    end
    properties (Constant, Access=private)
        increment = 1;
    end
    methods
        % Observe any model by incorporating its name to MyObserverPreexit.
        function this = myObserverPreexit(Model)
            % Input model name to the simulation observer.
            this@simevents.SimulationObserver(Model);
            this.Model = Model;
        end
        % Initialize the count in the simulation start.
        function simStarted(this)
            this.count = 0;
        end
        % Specify list of blocks to be notified of entity entry and exit
        % events.
        function Block = getBlocksToNotify(this)
            Block = this.getAllBlockWithStorages();
        end
        function preExit(this, evSrc, Data)
            % Get the names of all storage blocks that the entities depart.
            % This returns the block with its path.
            Block = Data.Block.BlockPath;
            % Remove the path to display only the
            % block name.
            Block = regexp(Block, 'ObserverPreexitModel/' , '');
            % Initialize the blocks to observe.
            BlockName = 'Entity Server';
            % If the block that entity exits contains the block name
            % acquire data for exit time and block name.
            if contains(Block, BlockName)
                % Get time for entity preexit from event calendar.
            end
        end
    end
end
```

```
        evCal = this.getEventCalendars;
        Time = evCal(1).TimeNow;
        % Increase the count for departing entities.
        this.count = this.count + this.increment;
        myInfo = [' At time ', num2str(Time), ...
            ' an entity departs ', Block, ', Total entity count is ', ...
            num2str(this.count)];
        disp(myInfo);
    end
end
end
end
```

Save the file as `myObserverPreexit.m`.

Enable the observer object to monitor `ObserverPreexitModel` model.

```
obj = myObserverPreexit('ObserverPreexitModel');
```

## Version History

Introduced in R2016a

### See Also

[addBlockNotification](#) | [getAllBlockWithStorages](#) | [getBlocksToNotify](#)

### Topics

“Observe Entities Using `simevents.SimulationObserver` Class”

# addBlockNotification

**Class:** simevents.SimulationObserver

**Package:** simevents

Add block to list of blocks to be notified

## Syntax

```
addBlockNotification(obj,blkPath)
```

## Description

`addBlockNotification(obj,blkPath)` is a utility function for adding a block to the list of blocks to be notified. Specify the full path of the block to be added in `blkPath`.

## Input Arguments

**obj** — **SimulationObserver object**

character vector

Object of class SimulationObserver

**blkPath** — **Full path of the block to be notified**

character vector

Full path of the block to be added to the list of blocks to be notified.

## Examples

### Add Block to List of Blocks for Notification

Add block to list of blocks for notification.

```
function postEntry(obj,eventSource,eventData)
    if someConditionIsTrue
        addBlockNotification(obj,[this.mModel '/Patron Enter']);
    end
end
```

## Version History

Introduced in R2016a

## See Also

`getAllBlockWithStorages` | `getBlocksToNotify` | `getEventCalendars` | `getHandleToBlock` | `getHandlesToBlockStorages` | `notifyEventCalendarEvents` | `postEntry` | `preExecute` | `preExit` | `removeBlockNotification` | `simPaused` | `simResumed` | `simStarted` | `simTerminating`

**Topics**

“Use SimulationObserver Class to Monitor a SimEvents Model”

# getAllBlockWithStorages

**Class:** simevents.SimulationObserver

**Package:** simevents

Get list of blocks that store entities

## Syntax

```
getAllBlockWithStorages(obj)
```

## Description

getAllBlockWithStorages(obj) is a utility function that returns the paths of all blocks that store entities.

## Input Arguments

**obj** — **SimulationObserver object**

character vector

Object of class SimulationObserver

## Output Arguments

**allBlkPaths** — **Paths of all blocks that store entities**

cell array of character vectors

Cell array of all blocks that store entities, provided as full block paths.

## Examples

### Return Paths of All Blocks that Store Entities

Return the paths of all blocks that store entities.

```
function blks=getBlocksToNotify(obj)
    blks=getAllBlockWithStorages(obj);
end
```

## Version History

Introduced in R2016a

## See Also

addBlockNotification | getBlocksToNotify | getEventCalendars | getHandleToBlock | getHandlesToBlockStorages | notifyEventCalendarEvents | postEntry | preExecute |

preExit | removeBlockNotification | simPaused | simResumed | simStarted |  
simTerminating

**Topics**

“Use SimulationObserver Class to Monitor a SimEvents Model”

# getBlocksToNotify

**Class:** simevents.SimulationObserver

**Package:** simevents

Specify list of blocks to be notified of entity entry and exit events

## Syntax

```
getBlocksToNotify(obj)
```

## Description

`getBlocksToNotify(obj)` is used to specify a cell array of block paths that are notified by the `SimulationObserver` object. These blocks have to be discrete event blocks with entity storages. Override this function in your subclass to specify a cell array of blocks for which `preExit` and `postEntry` methods will be called. Specify 'ALL' to run these methods on all the discrete-event blocks with entity storages in the model. If you do not want any blocks to be notified, specify an empty cell array, `{}`.

## Input Arguments

**obj** — **SimulationObserver object**

character vector

Object of class `SimulationObserver`

## Output Arguments

**blks** — **List of blocks being notified of runtime events**

`{}` (default) | cell array of character vectors

Cell array of full block paths of all blocks being notified of runtime events

## Examples

### Blocks to Observe in Model

Return the list of blocks you want to observe in the model.

```
function blks = getBlocksToNotify(this)
    % Return list of blocks to observe in the model
    %
    % For this example, we are only interested in the following
    % blocks as they are sufficient for us to know all events of
    % interest
    blks = { ...
        [this.mModel '/Patron Enter'], ...
        [this.mModel '/Have Dinner'], ...
```

```
        [this.mModel '/Patron Leave'] ...  
    };  
end
```

## Version History

Introduced in R2016a

### See Also

addBlockNotification | getAllBlockWithStorages | getEventCalendars |  
getHandleToBlock | getHandlesToBlockStorages | notifyEventCalendarEvents |  
postEntry | preExecute | preExit | removeBlockNotification | simPaused | simResumed |  
simStarted | simTerminating

### Topics

“Use SimulationObserver Class to Monitor a SimEvents Model”



# getEventCalendars

**Class:** simevents.SimulationObserver

**Package:** simevents

Get handles to event calendars

## Syntax

```
getEventCalendars(obj)
```

## Description

`getEventCalendars(obj)` is a utility method that returns handles to all event calendars in your model.

## Input Arguments

**obj** — **SimulationObserver object**

character vector

Object of class SimulationObserver

## Output Arguments

**evCal** — **Array of event calendars in model**

array of handles to EventCalendar objects

Array of handles to the event calendars in your model.

## Examples

### Get Handles to Event Calendars in Model

Get handles to all event calendars in your model.

```
function postEntry(obj, evSrc, evData)
    % Print simulation time
    evcal=getEventCalendars(obj);
    tNow=evcal(1).TimeNow;
    disp(tNow);
end
```

## Version History

Introduced in R2016a

**See Also**

addBlockNotification | getAllBlockWithStorages | getBlocksToNotify |  
getHandleToBlock | getHandlesToBlockStorages | notifyEventCalendarEvents |  
postEntry | preExecute | preExit | removeBlockNotification | simPaused | simResumed |  
simStarted | simTerminating

**Topics**

“Use SimulationObserver Class to Monitor a SimEvents Model”

# getHandlesToBlockStorages

**Class:** simevents.SimulationObserver

**Package:** simevents

Return storage handles of specified block

## Syntax

```
getHandlesToBlockStorages(obj,blkPath)
```

## Description

`getHandlesToBlockStorages(obj,blkPath)` returns the storage handles for the block specified by `blkPath`. If the block does not store entities, this method returns a 0x0 array of `simevents.Storage` objects.

## Input Arguments

**obj — SimulationObserver object**

character vector

Object of class `SimulationObserver`

**blkPath — Full path to block**

character vector

Full path to the block that stores entities

## Output Arguments

**storagesForBlock — Storage handles for the block**

array of handles to `simevents.Storage` objects

Array of storage handles of the block. If the block does not store entities, output is a 0x0 array of storage.

## Examples

### Get Handles for All Block Storage Elements

Get handles for all block storage elements in the model.

```
function postEntry(obj,evSrc,evData)
    % Number of entities in server;
    storage=getHandlesToBlockStorages(obj,[this.mModel '/Have Dinner']);
```

```
    disp(length(storage.Entity));  
end
```

## **Version History**

**Introduced in R2016a**

### **See Also**

addBlockNotification | getAllBlockWithStorages | getBlocksToNotify |  
getEventCalendars | getHandleToBlock | notifyEventCalendarEvents | postEntry |  
preExecute | preExit | removeBlockNotification | simPaused | simResumed | simStarted |  
simTerminating

### **Topics**

“Use SimulationObserver Class to Monitor a SimEvents Model”

# getHandleToBlock

**Class:** simevents.SimulationObserver

**Package:** simevents

Return block handle for a given block path

## Syntax

```
getHandleToBlock(obj, blkPath)
```

## Description

`getHandleToBlock(obj, blkPath)` is a utility function that returns the handle to the block whose full path is specified by `blkPath`.

## Input Arguments

**obj** — **SimulationObserver object**

character vector

Object of class SimulationObserver

**blkPath** — **Full path to block**

character vector

## Output Arguments

**blkHandle** — **Handle to block**

handle to block

Handle to the block specified in `blkPath`.

## Examples

### Return Handle to Specified Block

Return handle to specified block.

```
function postEntry(obj, evSrc, evData)
   hdl=getHandleToBlock(obj, [this.mModel ' /Have Dinner']);
    ...
end
```

## Version History

**Introduced in R2016a**

**See Also**

addBlockNotification | getAllBlockWithStorages | getBlocksToNotify |  
getEventCalendars | getHandlesToBlockStorages | notifyEventCalendarEvents |  
postEntry | preExecute | preExit | removeBlockNotification | simPaused | simResumed |  
simStarted | simTerminating

**Topics**

“Use SimulationObserver Class to Monitor a SimEvents Model”

# notifyEventCalendarEvents

**Class:** simevents.SimulationObserver

**Package:** simevents

Specify whether you want notification for all events in event calendar

## Syntax

```
notifyEventCalendarEvents(obj)
```

## Description

`notifyEventCalendarEvents(obj)` specifies whether you want notification for all events in the event calendar before they are executed. Set the output of this method to `true` to call the `preExecute` method for all events in the event calendar.

## Input Arguments

**obj** — **SimulationObserver** object

character vector

Object of class SimulationObserver

## Output Arguments

**n** — **Boolean specifying whether all events in event calendar are notified before executing**

false (default) | true

Boolean that specifies whether you are notified of all events in the event calendar before executing. If set to `true`, the `preExecute` method is called for every event before its execution.

## Examples

### Specify Notification for All Events in Event Calendar

Specify whether you want notification for all events in event calendar.

```
function status=notifyEventCalendarEvents(obj)
    status=false;
end
```

## Version History

Introduced in R2016a

## See Also

[addBlockNotification](#) | [getAllBlockWithStorages](#) | [getBlocksToNotify](#) | [getEventCalendars](#) | [getHandleToBlock](#) | [getHandlesToBlockStorages](#) | [postEntry](#) |

preExecute | preExit | removeBlockNotification | simPaused | simResumed | simStarted |  
simTerminating

**Topics**

“Use SimulationObserver Class to Monitor a SimEvents Model”



# postEntry

**Class:** simevents.SimulationObserver

**Package:** simevents

Specify behavior after an entity enters a block that has entity storage

## Syntax

```
postEntry(obj, evSrc, evData)
```

## Description

`postEntry(obj, evSrc, evData)` is used to specify behavior after an entity enters a block that has entity storage. The simulation observer uses this method as a callback for post-entry event notification and provides handles to the entity, the block and its storage, and the event.

## Input Arguments

**obj — SimulationObserver object**

character vector

Object of class SimulationObserver

**evSrc — Handle to block storage**

handle to simevents.Storage object

Handle to block storage in which the entity entered. The handle will be populated by the simulation observer.

**evData — List of handles for block, storage, and entities, and event type**

cell array of handles

List of handles for block, storage, and entities. The list will be populated by the simulation observer.

## Examples

### Specify Listener for Storage Entry

Specify listener to execute when entity enters a storage element such as a queue or server.

```
function postEntry(this, evSrc, evData)
    % Override to specify listener for entry into a storage (queue/server)

    entity = evData.CurrentEntity;

    if strcmp(evData.Block.BlockPath, [this.mModel '/Have Dinner'])

        % Identify which table the customer is going to
        tblId = this.occupyTable(entity);
```

```
        % Schedule motion for this customer to the appropriate
        % table
        target = this.cTablePos(tblId, :);
        this.scheduleMotion(entity, target);

        % Decrement the waiting statistic
        this.updateStats(this.mTxtWaiting, this.DECREMENT);

    elseif strcmp(evData.Block.BlockPath, [this.mModel '/Patron Leave'])
        % Schedule motion for this entity from its current position
        % to the exit position
        if isKey(this.mEntityGlyphs, num2str(entity.ID))
            this.scheduleMotion(entity, this.cExitPos);
        end

        % Schedule for the entity dot to be destroyed when it has
        % completed its pending motion
        this.scheduleMotion(entity, [NaN, NaN]);

    end
end
```

## Version History

Introduced in R2016a

### See Also

[addBlockNotification](#) | [getAllBlockWithStorages](#) | [getBlocksToNotify](#) |  
[getEventCalendars](#) | [getHandleToBlock](#) | [getHandlesToBlockStorages](#) |  
[notifyEventCalendarEvents](#) | [preExecute](#) | [preExit](#) | [removeBlockNotification](#) |  
[simPaused](#) | [simResumed](#) | [simStarted](#) | [simTerminating](#)

### Topics

“Use SimulationObserver Class to Monitor a SimEvents Model”

# preExecute

**Class:** simevents.SimulationObserver

**Package:** simevents

Specify behavior before execution of an event

## Syntax

```
preExecute(obj, evSrc, evData)
```

## Description

`preExecute(obj, evSrc, evData)` is used to specify behavior before the execution of an event in the event calendar. The simulation observer uses this method as a callback for pre-execute event notifications and provides a handle to the event calendar.

## Input Arguments

**obj — SimulationObserver object**

character vector

Object of class SimulationObserver

**evSrc — Handle to event calendar**

handle to simevents.EventCalendar object

Handle to event calendar. The handle will be populated by the simulation observer.

**evData — Event name and handle to event calendar**

cell array of handles

Event name and handle to event calendar

## Examples

### Specify Behavior Before Execution of Event

Specify behavior before the execution of an event in the event calendar.

```
function preExecute(obj, evSrc, evData)
    fprintf('Specify behavior before the execution of an event in the event calendar.');
```

## Version History

Introduced in R2016a

**See Also**

addBlockNotification | getAllBlockWithStorages | getBlocksToNotify |  
getEventCalendars | getHandleToBlock | getHandlesToBlockStorages |  
notifyEventCalendarEvents | postEntry | preExit | removeBlockNotification |  
simPaused | simResumed | simStarted | simTerminating

**Topics**

“Use SimulationObserver Class to Monitor a SimEvents Model”

## preExit

**Class:** simevents.SimulationObserver

**Package:** simevents

Specify behavior before an entity exits a block with entity storage

### Syntax

```
preExit(obj, evSrc, evData)
```

### Description

`preExit(obj, evSrc, evData)` is used to specify behavior before an entity exits a block that stores entities. The simulation observer uses this method as a callback for pre-exit event notification and provides handles to the entity, the block and its storage, and the event.

### Input Arguments

**obj** — **SimulationObserver object**

character vector

Object of class SimulationObserver

**evSrc** — **Handle to event calendar**

handle to simevents.EventCalendar object

Handle to event calendar. The handle will be populated by the simulation observer.

**evData** — **List of handles of block, storage, and entities, and event type**

cell array of handles

List of handles of block, storage, and entities. The list will be populated by the simulation observer.

### Examples

#### Specify Listener for Storage Exit

Specify listener to execute when entity exits a storage element such as a queue or server.

```
function preExit(this, ~, evData)
    % Override to specify listener for exit from a storage (queue/server)
    % evData contains block, storage, and entity handles

    entity = evData.CurrentEntity;

    if strcmp(evData.Block.BlockPath, [this.mModel '/Patron Enter'])
        % Create a new "dot" on the figure at the entry position
        h = plot(this.cEntryPos(1), this.cEntryPos(2), '.');
        set(h, 'MarkerSize', 32);
```

```
% Add a mouse-click function to the dot so we can retrieve
% attribute data when user clicks on this customer
set(h, 'ButtonDownFcn', @(h,e)entityClickFcn(this,h,e));

% Cache away the entity identifier on this dot
set(h, 'Tag', num2str(entity.ID));

% Cache away this dot handle so that we can move it in
% future events
this.mEntityGlyphs(num2str(entity.ID)) = h;

% Cache away the entity handle
this.mEntities(num2str(entity.ID)) = entity;

% Increment the entry statistics
this.updateStats(this.mTxtEntry, this.INCREMENT);

% Schedule motion for this entity from its current position
% to a random position in the waiting area
this.scheduleMotion(entity, this.getRandWaitingPos());

% Increment waiting statistic
this.updateStats(this.mTxtWaiting, this.INCREMENT);
elseif strcmp(evData.Block.BlockPath, [this.mModel '/Have Dinner'])
    this.releaseTable(entity);
end
end
end
```

## Version History

Introduced in R2016a

### See Also

[addBlockNotification](#) | [getAllBlockWithStorages](#) | [getBlocksToNotify](#) | [getEventCalendars](#) | [getHandleToBlock](#) | [getHandlesToBlockStorages](#) | [notifyEventCalendarEvents](#) | [postEntry](#) | [preExecute](#) | [removeBlockNotification](#) | [simPaused](#) | [simResumed](#) | [simStarted](#) | [simTerminating](#)

### Topics

“Use SimulationObserver Class to Monitor a SimEvents Model”

# removeBlockNotification

**Class:** simevents.SimulationObserver

**Package:** simevents

Remove block from list of blocks being notified

## Syntax

```
removeBlockNotification(obj,blkPath)
```

## Description

removeBlockNotification(obj,blkPath) is a utility function used to remove a block from the list of blocks being notified. Specify the full path of the block to be added in blkPath.

## Input Arguments

**obj** — **SimulationObserver object**

character vector

Object of class SimulationObserver

**blkPath** — **Full path of the block to be notified**

character vector

Full path of the block to be added to the list of blocks being notified.

## Examples

### Remove Block

Remove block from list of blocks being notified.

```
function postEntry(obj,eventSource,eventData)
    if someConditionIsTrue
        removeBlockNotification(obj,[this.mModel '/Patron Enter']);
    end
end
```

## Version History

Introduced in R2016a

## See Also

addBlockNotification | getAllBlockWithStorages | getBlocksToNotify |  
 getEventCalendars | getHandleToBlock | getHandlesToBlockStorages |  
 notifyEventCalendarEvents | postEntry | preExecute | preExit | simPaused | simResumed  
 | simStarted | simTerminating

**Topics**

“Use SimulationObserver Class to Monitor a SimEvents Model”



# simPaused

**Class:** simevents.SimulationObserver

**Package:** simevents

Specify behavior when simulation pauses

## Syntax

```
simPaused(obj)
```

## Description

`simPaused(obj)` determines the behavior when the simulation is paused. Override this function to specify the behavior of your visualization when the simulation pauses, as determined by the `SimulationStatus` parameter.

## Input Arguments

**obj** — **SimulationObserver** object

character vector

Object of class `SimulationObserver`

## Examples

### Call Method When Pausing Model

Call this method when model is paused.

```
function simPaused(this)
    % Called when model is paused

    % Schedule the timer to stop when all pending animation is
    % completed
    this.mTimerRequestPause = true;
end
```

## Version History

Introduced in R2016a

## See Also

`addBlockNotification` | `getAllBlockWithStorages` | `getBlocksToNotify` | `getEventCalendars` | `getHandleToBlock` | `getHandlesToBlockStorages` | `notifyEventCalendarEvents` | `postEntry` | `preExecute` | `preExit` | `removeBlockNotification` | `simResumed` | `simStarted` | `simTerminating`

**Topics**

“Use SimulationObserver Class to Monitor a SimEvents Model”

# simResumed

**Class:** simevents.SimulationObserver

**Package:** simevents

Specify behavior when simulation resumes

## Syntax

```
simResumed(obj)
```

## Description

`simResumed(obj)` determines the behavior when the simulation resumes after pausing. Override this function to specify the behavior of your visualization when the simulation resumes, as determined by the `SimulationStatus` parameter.

## Input Arguments

**obj** — **SimulationObserver** object

character vector

Object of class `SimulationObserver`

## Examples

### Call Method When Model Continues

Call this method when model continues after pausing.

```
function simResumed(this)
    % Called when model continues from being paused

    % Restart the timer
    this.mTimerRequestPause = false;
    start(this.mTimer);
end
```

## Version History

Introduced in R2016a

## See Also

`addBlockNotification` | `getAllBlockWithStorages` | `getBlocksToNotify` | `getEventCalendars` | `getHandleToBlock` | `getHandlesToBlockStorages` | `notifyEventCalendarEvents` | `postEntry` | `preExecute` | `preExit` | `removeBlockNotification` | `simPaused` | `simStarted` | `simTerminating`

**Topics**

“Use SimulationObserver Class to Monitor a SimEvents Model”

# simStarted

**Class:** simevents.SimulationObserver

**Package:** simevents

Specify behavior when simulation starts

## Syntax

simStarted(obj)

## Description

simStarted(obj) determines the behavior when the simulation starts. Override this function to specify the behavior of your visualization when the simulation starts, as determined by the SimulationStatus parameter.

## Input Arguments

**obj** — SimulationObserver object

character vector

Object of class SimulationObserver

## Examples

### Initialize Animation Canvas

Initialize the animation canvas.

```
function simStarted(this)
    % Initialize the animation canvas

    % Re-initialize runtime work variables for simulation
    this.mEntityGlyphs = containers.Map('keytype', 'char', 'valuetype', 'any');
    this.mEntities = containers.Map('keytype', 'char', 'valuetype', 'any');
    this.mCombineMap = containers.Map('keytype', 'char', 'valuetype', 'char');
    this.mCachePostRun = containers.Map('keytype', 'char', 'valuetype', 'char');
    this.mTableOccupy = zeros(1, size(this.cTablePos,1)) - 1;

    % Setup the figure with the restaurant floor as background
    close all;
    im = imread('restaurant.png');
    image(im);
    this.mFig =(gcf);
    set(this.mFig, 'Tag', 'Begin');
    this.mAx = gca;
    set(this.mFig, 'toolbar', 'none');
    set(this.mFig, 'menubar', 'none');
    set(this.mAx, 'XTickLabel', '');
    set(this.mAx, 'YTickLabel', '');
    set(this.mAx, 'Box', 'on');
    set(this.mAx, 'TickLength', [0 0]);
    set(this.mAx, 'position', [0 0 1 1]);
    hold on;

    % Set up the numeric statistics text labels on the figure
    this.mTxtEntry = text(170,850, '0');
    this.mTxtWaiting = text(10,160, '0');
```

```
this.mTxtExit = text(920,330, '0');
this.mTxtSelectedEnt = text(50,600,'');

set(this.mTxtEntry, 'Color', [0.8500 0.3250 0.0980], 'FontWeight', 'bold', 'FontSize', 14);
set(this.mTxtWaiting, 'Color', [0.8500 0.3250 0.0980], 'FontWeight', 'bold', 'FontSize', 14);
set(this.mTxtExit, 'Color', [0.8500 0.3250 0.0980], 'FontWeight', 'bold', 'FontSize', 14);

this.mLineSelectedEnt = plot(0,0,'.');

% Set up the timer
this.mTimer = timer(...
    'TimerFcn',      @(t,e)animate(this,t,e), ...
    'ExecutionMode', 'fixedSpacing', ...
    'Period',        this.cTimerPeriod);
this.mTimerData = containers.Map('keytype', 'char', 'valuetype', 'any');
this.mTimerRequestStop = false;
this.mTimerRequestPause = false;
start(this.mTimer);
end
```

## Version History

Introduced in R2016a

### See Also

[addBlockNotification](#) | [getAllBlockWithStorages](#) | [getBlocksToNotify](#) | [getEventCalendars](#) | [getHandleToBlock](#) | [getHandlesToBlockStorages](#) | [notifyEventCalendarEvents](#) | [postEntry](#) | [preExecute](#) | [preExit](#) | [removeBlockNotification](#) | [simPaused](#) | [simResumed](#) | [simTerminating](#)

### Topics

“Use SimulationObserver Class to Monitor a SimEvents Model”

# simTerminating

**Class:** simevents.SimulationObserver

**Package:** simevents

Define observer behavior when simulation is terminating

## Syntax

simTerminating(obj)

## Description

simTerminating(obj) determines the behavior when the simulation is terminating. Override this function to specify the behavior of your visualization when the simulation is terminating, as determined by the SimulationStatus parameter.

## Input Arguments

**obj** — SimulationObserver object

character vector

Object of class SimulationObserver

## Examples

### Call Method When Simulation is Terminating

Call this method when simulation is terminating.

```
function simTerminating(this)
    % Called when simulation is terminating
    %
    % After the simulation terminates, in order to support clicking
    % on entity to see attributes, we gather up all of the entities
    % that exist in the model and save their attribute information

    ents = this.mEntityGlyphs.keys;
    for idx = 1 : length(ents)
        ent = ents{idx};
        try
            enStruct = this.mEntities(ent);
            str = evalc('disp(enStruct.Attributes)');
            this.mCachePostRun(ent) = str;
        catch me
            end
    end

    % If animation timer is still running, schedule a stop
    if strcmp(this.mTimer.Running, 'on')
        this.mTimerRequestStop = true;
    end
end
```

```
else
    % If timer is not running, delete it
    delete(this.mTimer);
end
end
```

## Version History

Introduced in R2016a

### See Also

addBlockNotification | getAllBlockWithStorages | getBlocksToNotify |  
getEventCalendars | getHandleToBlock | getHandlesToBlockStorages |  
notifyEventCalendarEvents | postEntry | preExecute | preExit |  
removeBlockNotification | simPaused | simResumed | simStarted

### Topics

“Use SimulationObserver Class to Monitor a SimEvents Model”

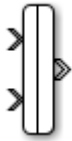


# Blocks

---

# Composite Entity Creator

Create composite entities



**Libraries:**  
SimEvents

## Description

The Composite Entity Creator block creates a composite entity for each set of entities arriving simultaneously at all input ports. The newly created entity can include information about the structure, attributes, and timers of the arriving entities.

You can combine entities from different paths using this block. The combined entity represents different parts within a larger item, such as the header, payload, and trailer that are parts of a data packet. Alternatively, you can model resource allocation by combining an entity that represents a resource with an entity that represents a part or other item.

The Composite Entity Creator block detects when all necessary component entities are present and when the composite entity that results from the combining operation will be able to advance to the next block. You can also configure the Composite Entity Creator block to make the combining operation reversible via the Composite Entity Splitter block.

## Ports

### Input

**port\_1** — Input entity  
scalar | vector | matrix

Input entity port for entities entering the block.

Data Types: single | double | int8 | int16 | int32 | uint8 | uint16 | uint32 | Boolean | enumerated | bus | fixed point

**port\_2** — Input entity  
scalar | vector | matrix

Input entity port for entities entering the block.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | fixed point

### Output

**port\_1** — Output composite entity  
scalar | vector | matrix

Output entity port for composite entities exiting the clock.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `enumerated` | `bus` | `fixed point`

## Parameters

**Number of input ports** — Specify the number of input ports  
2 (default) | scalar

Specify the number of input ports to create the composite entity.

### Programmatic Use

**Block Parameter:** `NumberInputPorts`

**Type:** character vector

**Values:** '2' | scalar

**Default:** '2'

**Entity type name** — Specify the type name of the composite entity  
`Combined` (default) | character vector

Specify the type name of the composite entity that is created after combining incoming entities.

### Programmatic Use

**Block Parameter:** `EntityType`

**Type:** character vector

**Values:** 'Combined' | character vector

**Default:** 'Combined'

**Bus object** — Specify the output  
`off` (default) | `on`

Specify whether to output the composite entity as a bus object.

### Programmatic Use

**Block Parameter:** `BusObject`

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

**Define input entity names** — Specify the names of the input entities used to generate the composite entity  
{E1, E2} (default) | character vector

Names of the entities in the composite entity.

### Programmatic Use

**Block Parameter:** `InputEntityName`

**Type:** character vector

**Values:** 'E1|E2' | character vector

**Default:** 'E1|E2'

## **Version History**

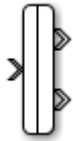
**Introduced in R2016a**

### **See Also**

Entity Queue | Entity Gate | Entity Generator | Composite Entity Splitter | Entity Input Switch | Entity Output Switch | Entity Server | Entity Terminator | Resource Acquirer | Resource Releaser | Resource Pool

# Composite Entity Splitter

Split composite entities



**Libraries:**  
SimEvents

## Description

The Composite Entity Splitter block splits a composite entity into its individual entities and outputs them through each unblocked entity output port. A composite entity can be the output of the Composite Entity Creator block, a structured type or a bus type entity from Entity Generator or Message Send blocks.

## Ports

### Input

**port\_1** — Input composite entity  
scalar | vector | matrix

Input entity port for composite entities entering the block.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | fixed point

### Output

**port\_1** — Output entity  
scalar | vector | matrix

Output port for entities exiting the block.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | fixed point

**port\_2** — Output entity  
scalar | vector | matrix

Output port for entities exiting the block.

Data Types: single | double | int8 | int16 | int32 | uint8 | uint16 | uint32 | Boolean | enumerated | bus | fixed point

## Parameters

**Number of output ports** — Specify the number of output ports  
2 (default) | scalar

Specify the number of output ports to output entities.

**Programmatic Use**

**Block Parameter:** NumberOutputPorts

**Type:** character vector

**Values:** '2' | scalar

**Default:** '2'

## **Version History**

**Introduced in R2016a**

### **See Also**

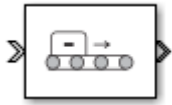
Entity Generator | Composite Entity Creator

### **Topics**

“SimEvents Common Design Patterns”

# Conveyor System

Transport entities



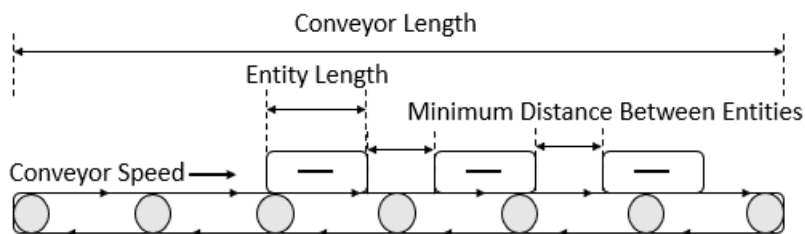
**Libraries:**  
SimEvents

## Description

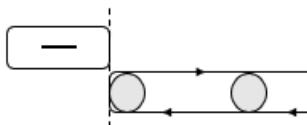
The Conveyor System block transports entities across the surface of a conveyor. Entities enter the block by sliding onto the conveyor surface and they depart the block by sliding off. You can specify the speed of the conveyor. If the conveyor speed is variable, a second input port appears on the block to accept anonymous entities that carry data for specifying the new conveyor speed. Use this block to model transportation applications involving production systems, or logistical systems.

In the Conveyor System block:

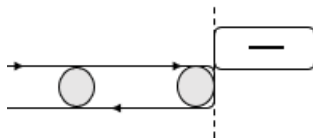
- You can specify the speed and the surface length of the conveyor. You can specify the length and the minimum distance between the transported entities.



- Entities slide into the conveyor surface and an entity is considered as inside the conveyor surface when its front side coincides with the surface entry.



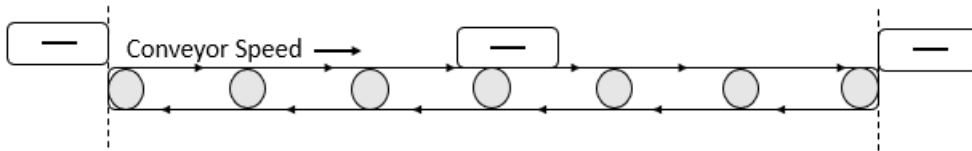
- Entities slide out of the conveyor surface and an entity is considered as outside the conveyor surface when its back side coincides with the surface entry.



- Conveyor speed determines the total time between an entity entry to the surface and its exit from the surface.

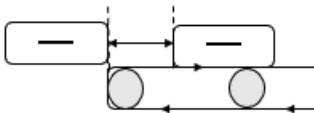
Entity can enter and exit the conveyor surface provided that:

- There are no other entities blocking the entity when the **Blocked output behavior** is set to **Accumulate**.
- The conveyor surface is not paused when the **Blocked output behavior** is set to **Pause**. For more information, see “Blocked output behavior” on page 2-0 .



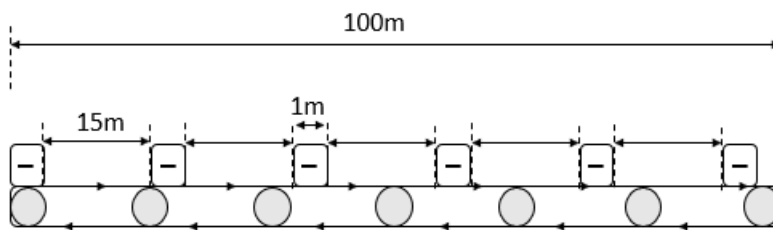
For instance, suppose that the conveyor length is 10, entity length is 1, and conveyor speed is 5. Then, it takes 2.2 simulation time for the entity to depart the surface.  $10/5 = 2$  to travel through the surface and  $1/5 = 0.2$  to depart from the surface because its length is 1.

- After an entity enters the conveyor surface, the next entity enters after the first one travels to the specified minimum distance between entities.



- The capacity of the conveyor system is the maximum number of entities allowed on the surface. The capacity is determined by the total surface length, entity length, and the minimum distance between entities.

For example, suppose that the entity length is 1 meter, conveyor system surface length is 100 meters, and the distance between entities is 15 meters. The capacity of the conveyor system becomes 6 entities.



- When using the **Conveyor length**, **Conveyor speed**, **Minimum distance between entities**, and **Entity length value** parameters ensure that the values are consistent with each other.

For example, specify the entity length of 10 cm and a conveyor system of length 100 m to achieve consistency:

- Set the **Conveyor length** parameter to 100.
- Set the **Entity length** parameter to 0.1.



## Ports

### Input

#### **Port\_1** — Incoming entity

scalar | vector | matrix

Input entity port for entities entering the queue. Entities are not accepted by the block when the speed of the conveyor is 0.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | fixed point

#### **S** — Incoming anonymous entity

scalar

Input port for anonymous entities entering the block. The entity carries data that specifies the new conveyor speed upon its arrival. The entity data must be a non-negative value between 0 and Inf. The new speed applies to all entities in the block including the incoming and existing ones.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64

### Output

#### **Port\_1** — Exiting entity

scalar | vector | matrix

Output entity port for entities leaving the queue

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | fixed point

#### **Port\_d** — Number of entities departed

off (default) | on

Selecting this check box outputs the number of entities that have exited the block.

### Dependencies

To enable this port, select the **Statistics > Number of entities departed, d** check box.

Data Types: double

#### **Port\_n** — Number of entities in block

off (default) | on

Selecting this check box outputs the number of entities in the block.

### Dependencies

To enable this block, select the **Statistics > Number of entities in block, n** check box.

Data Types: double

#### **Port\_pe** — Pending entity in block

off (default) | on

Selecting this check box outputs the value 1 for a pending entity in the block, and 0 otherwise.

**Dependencies**

To enable this port, select the **Statistics > Pending entity in block, pe**.

Data Types: double

**Port\_s** — Conveyor speed  
off (default) | on

Selecting this check box outputs the conveyor speed.

**Dependencies**

To enable this port, select the **Statistics > Conveyor speed, s**.

Data Types: double

**Parameters**

**Conveyor length** — Length of surface  
100 (default) | numeric

Length of surface that entities travel on. For more information, see “Description” on page 2-7.

**Programmatic Use**

**Block Parameter:** mConveyorLength

**Type:** character vector

**Values:** '100' | scalar

**Default:** '100'

**Conveyor speed** — Speed of the conveyor surface  
1000 (default) | numeric

Speed of surface that entities travel on. The speed can take values greater than 0 and its value can be zero only when the **Variable conveyor speed** check box is selected. For more information, see “Description” on page 2-7.

**Programmatic Use**

**Block Parameter:** mConveyorSpeed

**Type:** character vector

**Values:** '1000' | scalar

**Default:** '1000'

**Minimum distance between entities** — Minimum physical separation  
0 (default) | numeric

Minimum physical separation entities maintain while moving across the conveyor system. For more information, see “Description” on page 2-7.

**Programmatic Use**

**Block Parameter:** mMinDisBetEntity

**Type:** character vector

**Values:** '0' | scalar

**Default:** '0'

**Entity length source** — Entity length source

Dialog (default) | Attribute

Provide entity length, selected from the drop-down list.

**Dependencies**

- Dialog — Selecting this option enables the **Entity length value** parameter.
- Attribute — Selecting this option enables the **Entity length attribute name** parameter.

**Programmatic Use**

**Block Parameter:** mEntityLengthSource

**Type:** character vector

**Values:** 'Dialog' | 'Attribute'

**Default:** 'Dialog'

**Entity length value** — Length of entities

1 (default) | numeric

Length of entities, specified as a numeric value. For more information, see “Description” on page 2-7.

**Dependencies**

To enable this parameter, select Dialog for **Entity length source**.

**Programmatic Use**

**Block Parameter:** mEntityLength

**Type:** character vector

**Values:** '1' | scalar

**Default:** '1'

**Entity length attribute name** — Name of entity length attribute

Length (default) | character vector

Name of entity length attribute, specified as a character vector.

**Dependencies**

To enable this parameter, select Attribute for **Entity length source**.

**Programmatic Use**

**Block Parameter:** mEntityLengthAttrName

**Type:** character vector

**Values:** 'Length' | character vector

**Default:** 'Length'

**Blocked output behavior** — Behavior when the output is blocked

Accumulate (default) | Pause | Error

Behavior when output is blocked, selected from drop-down list:

- **Accumulate** — Accumulate entities. In **Accumulate** mode, an entity continues to move on the conveyor surface until its movement is blocked by another entity ahead.

For example, if one or more entities are extracted out of the conveyor surface by the **Entity Find** block, entities that are behind the extracted entity continue to move forward until they occupy the empty space due to the extraction.

- **Pause** — Pause conveyor system. In **Pause** mode, all entities on the conveyor surface stop and move together. The conveyor surface stops moving when an entity at the exit is unable to depart. During the pause, the conveyor system does not accept new entities.
- **Error** — Return an error. In **Error** mode, when an entity is blocked from entering the conveyor surface the error is displayed.

#### Programmatic Use

**Block Parameter:** `mOutputBlockedOpt`

**Type:** character vector

**Values:** 'Accumulate' | 'Pause' | 'Error'

**Default:** 'Accumulate'

**Error if conveyor full** — Conveyor behavior when the maximum number of entities is reached  
on (default) | off

Conveyor behavior when the conveyor surface is full.

on

Return an error if the conveyor system is full.

off

Do not return an error if the conveyor system is full.

#### Programmatic Use

**Block Parameter:** `mErrorUponFullOpt`

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'on'

**Variable conveyor speed** — Constant or variable conveyor speed  
off (default) | on

Specify if the conveyor system has constant or variable speed.

off

Constant speed conveyor system. You can specify the speed using the **Conveyor speed** parameter.

on

Variable speed conveyor system. An input port appears to accept anonymous entities which carry data to specify the new conveyor speed.

#### Programmatic Use

**Block Parameter:** `mIsVariableSpeed`

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'on'

**Number of entities departed, d** — Number of entities departed  
off (default) | on

Number of entities that have departed the block.

**Programmatic Use**

**Block Parameter:** mNumEntitiesDepOpt

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

**Number of entities in block, n** — Number of existing entities  
off (default) | on

Outputs the number of entities present in the block.

**Programmatic Use**

**Block Parameter:** mNumEntitiesInBlockOpt

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

**Pending entity in block, pe** — Pending entities  
off (default) | on

Indicates whether an entity that is yet to depart is present in the block. The value is 1 for a pending entity, and 0 otherwise.

**Programmatic Use**

**Block Parameter:** mEntityPendingOpt

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

**Number of entities extracted, ex** — Number of entities extracted from this block  
off (default) | on

Outputs the number of extracted entities which are pulled out from this block by the Entity Find block. When an entity is extracted, **Number of entities departed, d**, and **Number of entities in block, n** statistics are updated accordingly. For more information about finding and extracting entities, see “Find and Extract Entities in SimEvents Models”.

**Programmatic Use**

**Block Parameter:** mNumExtractedFromBlockOpt

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

**Conveyor speed, s** — Speed of the conveyor for the duration of the simulation  
off (default) | on

Outputs the speed of the conveyor during the course of the simulation.

**Programmatic Use**

**Block Parameter:** mConveyorSpeedOpt

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

## Version History

**Introduced in R2017b**

### See Also

MATLAB Discrete-Event System | MATLAB System | Entity Queue | Entity Server | Entity Store

### Topics

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

# Discrete-Event Chart

Discrete event chart



**Libraries:**  
SimEvents

## Description

The Discrete-Event Chart block is similar to a Stateflow® chart but is used for discrete events. The block requires a Stateflow license.

The distinguishing characteristic of the Discrete-Event Chart block is that it executes in an event-based rather than time-based fashion. The Discrete-Event Chart block provides these advantages for discrete-event modeling:

- Precise timing — The time resolution for occurrence of events can be arbitrarily precise and is not limited by the sample time of the model.
- Trigger on arrival — A Discrete-Event Chart block executes immediately on message arrival. It does not wait for the next sample time hit.
- Variable execution order — A Discrete-Event Chart block does not have a fixed sorted execution order. The order of execution depends on the run-time conditions of the model.
- Multiple executions per time step — A Discrete-Event Chart block can execute zero or multiple times in a single time step.

The Discrete Event Chart can be used in a similar fashion to the Stateflow Chart (Stateflow).

To access the chart properties, right-click the chart and select **Properties**. For more information about the block properties, see “Create Custom Queuing Systems Using Discrete-Event Stateflow Charts”.

For information about SimEvents common design patterns with Discrete-Event Chart block, see “SimEvents Common Design Patterns”.

## Version History

**Introduced in R2016a**

## See Also

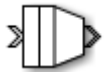
Entity Generator | MATLAB Discrete-Event System

## Topics

“Create Custom Queuing Systems Using Discrete-Event Stateflow Charts”  
“SimEvents Common Design Patterns”

## Entity Batch Creator

Create batch of entities



**Libraries:**  
SimEvents

### Description

The Entity Batch Creator receives the expected number of entities and creates a batch entity that contains all these entities. The batched entity is an array of entities. Any acquired resources have to be released using a Resource Releaser block before batching an input entity.

To customize actions when entities enter, exit, or are batched or blocked, enter MATLAB code in the **Entry action**, **Exit action**, **Batch generate action**, or **Blocked action** field of the **Event actions** tab. For more information about event actions, see “Events and Event Actions”.

You can write MATLAB code to manipulate the attributes of the batched entity. For example, to access attributes after an entity batch is generated, select **Batch generate action** and use the code.

```
entity.batch
```

If the number of entities in a batch is 4, then *entity.batch* is a 4-by-1 structure array. To manipulate **Attribute1** of the third entity in the batch enter the code.

```
entity.batch(3).Attribute1
```

You can reference batched entity attributes in event actions. You cannot reference them in:

- Priority queues — Do not set **Priority source** parameter to **PriorityAttribute**.
- Entity Server block — Do not set **Service time source** parameter to **Attribute**.
- Output Switch block — Do not set **Switching criterion** parameter to **From attribute**.

To output the batch as a bus object, select the **Bus object** parameter. Consider creating a bus object for the batched entity when:

- Sending or receiving a batched entity to or from a MATLAB Discrete-Event System block.
- Sending or receiving a batched entity to or from a Discrete-Event Chart block.
- When passing full entity data to a Simulink Function block.
- When converting a batched entity to a signal using the Message Receive block.

### Ports

#### Input

**Port\_1** — Input entity  
scalar | vector | matrix

Input entity port for entities entering the block.



Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | fixed point

### Output

**Port\_1** — Output batch entity

scalar | vector | matrix

Output entity port for batch entities exiting the block.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | fixed point

**Port\_a** — Number of entities arrived

scalar

Number of entities that have arrived at the block.

### Dependencies

To enable this port, click the **Statistics** tab and select the box labeled **Number of entities arrived, a**.

Data Types: double

**Port\_d** — Number of entities that have departed the block

scalar

Number of entities that have departed the block.

### Dependencies

To enable this port, click the **Statistics** tab and select the box labeled **Number of entities departed, d**.

Data Types: double

**Port\_rem** — Number of entities remaining for the next batch

scalar

Number of entities that remain for the next batch.

### Dependencies

To enable this port, click the **Statistics** tab and select the box labeled **Number of entities remaining for the next batch, rem**.

Data Types: double

**Port\_pe** — Pending entity in block

off (default) | on

Selecting this check box outputs 1 for a pending entity in the block, and 0 otherwise.

### Dependencies

To enable this port, select the **Statistics > Pending entity in block, pe**.

Data Types: double

## Parameters

**Number of entities in batch** — Number of entities in one batch  
4 (default) | scalar

Specify the number of entities in a batch.

**Programmatic Use**

**Block Parameter:** NumberOfEntitiesInBatch

**Type:** character vector

**Values:** '4' | scalar

**Default:** '4'

**Entity type name** — Name of the batched entity that is created after combining incoming entities  
Batch (default) | character vector

Specify the name of the batched entity that is created after combining incoming entities.

**Programmatic Use**

**Block Parameter:** EntityTypeName

**Type:** character vector

**Values:** 'Batch' | character vector

**Default:** 'Batch'

**Bus object** — Specify whether to output the batched entity as a bus object  
off (default) | on

Specify whether to output the batched entity as a bus object.

**Programmatic Use**

**Block Parameter:** BusObject

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

**Input entity name** — Specify names to be attached to the input entities  
batch (default) | character vector

Specify names to be attached to the input entities, which can be used for referencing these entities in the batched entity.

**Programmatic Use**

**Block Parameter:** InputEntityName

**Type:** character vector

**Values:** 'batch' | character vector

**Default:** 'batch'

**Event actions** — Specify the event action  
Entry (default) | Batch generate | Exit | Blocked

Specify the behavior of the entity on certain events. For example, the **Entry** action is called when the entity enters the block. To customize actions when entities enter, exit, or are batched or blocked,

enter MATLAB code in the Entry action, Exit action, Batch generate action, or Blocked action field of the **Event actions** tab. For more information about event actions, see “Events and Event Actions”. For an example, see “Manage Entities Using Event Actions”

**Programmatic Use**

**Block Parameter:** EntryAction, BatchGenerateAction, ExitAction, BlockedAction

**Type:** character vector

**Values:** MATLAB code

**Default:** ''

**Number of entities arrived, a** — Outputs the number of entities that have arrived at the block  
off (default) | on

Number of entities that have arrived at the block.

**Programmatic Use**

**Block Parameter:** NumberOfEntitiesArrived

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

**Number of entities departed, d** — Outputs the number of entities departed the block  
off (default) | on

Number of entities that have departed the block.

**Programmatic Use**

**Block Parameter:** NumberOfEntitiesDeparted

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

**Number of entities remaining for next batch, rem** — Outputs the number of entities remaining for the next batch  
off (default) | on

Outputs the number of entities still in the block for the next batch of entities.

**Programmatic Use**

**Block Parameter:** NumberOfEntitiesRequiredForNextBatch

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

**Pending entity in block, pe** — Pending entities  
off (default) | on

Indicates whether an entity that is yet to depart is present in the block. The value is 1 for a pending entity, and 0 otherwise.

**Programmatic Use**

**Block Parameter:** PendingEntity

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

## **Version History**

**Introduced in R2016b**

### **See Also**

Entity Generator | Entity Batch Splitter

# Entity Batch Splitter

Split batch entities



**Libraries:**  
SimEvents

## Description

The Entity Batch Splitter block splits a batched entity into its individual entities and outputs each entity through the output port. A batched entity is the output of the Entity Batch Creator block.

To customize actions when entities enter, exit, and are blocked or unbatched, enter MATLAB code in the **Entry** action, **Exit** action, **Blocked** action, or **Unbatch** action field of the **Event actions** tab.

## Ports

### Input

**port\_1** — Input batch entity

scalar | vector | matrix

Input entity port for entities entering the block.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | fixed point

### Output

**port\_1** — Output entity

scalar | vector | matrix

Output entity port for entities exiting the block.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | fixed point

**port\_a** — Number of entities arrived

scalar

Number of entities that have arrived at the block.

### Dependencies

To enable this port, click the **Statistics** tab and select the box labeled **Number of entities arrived, a**.

Data Types: double

**port\_d** — Number of entities that have departed the block

scalar

Number of entities that have departed the block.

**Dependencies**

To enable this port, click the **Statistics** tab and select the box labeled **Number of entities departed, d**.

Data Types: double

**port\_rem** — Number of entities remaining in the block  
scalar

Number of entities still in the block that have yet to depart.

**Dependencies**

To enable this port, click the **Statistics** tab and select the box labeled **Number of entities remaining to depart, rem**.

Data Types: double

**port\_pe** — Pending entity in block  
off (default) | on

Selecting this check box outputs the value 1 for a pending entity in the block, and 0 otherwise.

**Dependencies**

To enable this port, select the **Statistics > Pending entity in block, pe**.

Data Types: double

**Parameters**

**Event actions** — Specify the event action  
Entry (default) | Unbatch | Exit | Blocked

Specify the behavior of the entity on certain events. For example, the **Entry** action is called when the entity enters the block. To customize actions when entities enter, exit, or are unbatched or blocked, enter MATLAB code in the **Entry action**, **Exit action**, **Unbatch action**, or **Blocked action** field of the **Event actions** tab. For more information about event actions, see “Events and Event Actions”. For an example, see “Manage Entities Using Event Actions”

**Programmatic Use**

**Block Parameter:** EntryAction, UnbatchAction, ExitAction, BlockedAction

**Type:** character vector

**Values:** MATLAB code

**Default:** ''

**Number of entities arrived, a** — Outputs the number of entities that have arrived at the block  
off (default) | on

Number of entities that have arrived at the block.

**Programmatic Use****Block Parameter:** NumberOfEntitiesArrived**Type:** character vector**Values:** 'on' | 'off'**Default:** 'off'

**Number of entities departed, d** — Outputs the number of entities that have departed the block  
off (default) | on

Selecting this check box outputs the number of entities that have exited the block.

**Programmatic Use****Block Parameter:** NumberOfEntitiesDeparted**Type:** character vector**Values:** 'on' | 'off'**Default:** 'off'

**Number of entities remaining to depart, rem** — Outputs the number of entities that remain in the block  
off (default) | on

Outputs the number of entities still in the block that have yet to depart.

**Programmatic Use****Block Parameter:** NumberOfEntitiesWaitingToDepart**Type:** character vector**Values:** 'on' | 'off'**Default:** 'off'

**Pending entity in block, pe** — Pending entities  
off (default) | on

Indicates whether an entity that is yet to depart is present in the block. The value is 1 for a pending entity, and 0 otherwise.

**Programmatic Use****Block Parameter:** PendingEntity**Type:** character vector**Values:** 'on' | 'off'**Default:** 'off'

## Version History

Introduced in R2016b

### See Also

Entity Generator | Entity Batch Creator

## Entity Find

Find entities



**Libraries:**  
SimEvents

### Description

The Entity Find block finds entities that use a specific resource. The block receives a trigger entity from its input port. Upon receiving the trigger entity, it initiates a search across all blocks in a model for entities with a specific resource. You can further filter the search for finding entities by adding additional filtering conditions. The block can manipulate the found entities in these ways.

- Search entities that satisfy a specific condition across blocks in a model to find and examine them.
- Extract found entities from the model without modifying them.
- Change the attributes of the found entities at their location in the model without extraction.

In this case, to customize actions when the entity is found, in the **Event actions** tab, in the **On found** action field enter MATLAB code.

- Extract and change entity attributes. The extracted entities are queued in the block and rerouted through its output port.

In this case, to customize actions when entities enter, exit, or are blocked, enter MATLAB code in the **Entry action**, **Exit action**, or **Blocked action** field of the **Event actions** tab. For more information, see “Write Event Actions for Legacy Models”.

The block can extract entities from Entity Server, Entity Queue, Entity Store, Resource Acquirer, Entity Replicator, Conveyor System, Entity Selector, MATLAB Discrete-Event System, and Discrete-Event Chart blocks.

When an entity is extracted, pending events and statistics are updated accordingly. For instance, if an entity is extracted by an Entity Find block from an Entity Server block during the service period, the rest of the service is canceled and the output of the statistics are updated.

---

**Note** The block can find only one entity type that you specify in the model and it cannot find or extract entities across model reference boundaries.

---

For more information about common workflows involving Entity Find block, see “Find and Extract Entities in SimEvents Models”.



## Ports

### Input

**Port\_1** — Input entities to trigger finding entities

scalar | vector | matrix

Input port for entities to trigger the event of finding entities using a specific resource.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | fixed point

### Output

**Port\_1** — Output found entity

scalar | vector | matrix

Output entity port for entities that are found by the block.

### Dependencies

To enable this port, select the **Extract found entities** checkbox.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | fixed point

**Port\_f** — Number of entities found

scalar

Number of entities that are found by the block.

### Dependencies

To enable this port, click the **Statistics** tab and select the box labeled **Number of entities found, f**.

Data Types: double

**Port\_d** — Number of entities that have departed the block

scalar

Number of entities that have departed the block.

### Dependencies

To enable this port, click the **Statistics** tab and select the box labeled **Number of entities departed, d**.

Data Types: double

**Port\_n** — Number of found entities that have not yet departed the block

scalar

Number of found entities that have not yet departed the block.

### Dependencies

To enable this port, click the **Statistics** tab and select the box labeled **Number of found entities in block, n**.

Data Types: double

**Port\_w** — Average wait time for found entities in the block  
scalar

Average wait time for found entities in the block.

**Dependencies**

To enable this port, click the **Statistics** tab and select the box labeled **Average wait, w.**

Data Types: double

**Port\_l** — Average length of the found entity queue  
scalar

Outputs the average length of the found entity queue.

**Dependencies**

To enable this port, click the **Statistics** tab and select the box labeled **Average queue length, l.**

Data Types: double

**Port\_ex** — Number of entities extracted  
scalar

Number of entities that are extracted out of this block.

**Dependencies**

To enable this port, click the **Statistics** tab and select the box labeled **Number of entities extracted, ex.**

Data Types: double

## Parameters

### Main

**Resource** — Specify the name of the reference resource the entities use  
Resource1 (default) | character vector

Name of the reference resource that is used by the entities to be found by the block.

### Programmatic Use

**Block Parameter:** ResourceName

**Type:** character vector

**Values:** resource name

**Default:** 'Resource1'

**Extract found entities** — Extract and reroute found entities  
off (default) | on

Extract and output the entities that are found in the search.

**Programmatic Use****Block Parameter:** EnableOutput**Type:** character vector**Values:** 'on' | 'off'**Default:** 'off'**Additional filtering condition** — Add optional filtering condition

off (default) | on

Enable the option to define additional matching conditions for finding entities. If selected, a MATLAB editor appears where you can define additional matching conditions.

**Programmatic Use****Block Parameter:** EntityFilter**Type:** character vector**Values:** 'on' | 'off'**Default:** 'off'**Matching condition** — Define the matching condition

match = true; (default)

Use MATLAB code to define additional matching conditions for finding entities. The block searches for entities such that the value of the boolean variable `match` becomes `true`. For instance, when

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

the block finds entities that have the same `Attribute1` value with the trigger entity, because only their equality sets the `match` as `true`.

While authoring additional matching conditions, you can use these variables.

- `trigger` — Use to access the trigger entity attributes.
- `entity` — Use to access the attributes of the entity that is being found.
- `match` — Use as a Boolean value to be returned by the matching condition. The value of `match` is initialized as `false`.

**Dependencies**

This check box appears if the **Additional filtering condition** check box in the **Main** tab is selected.

**Programmatic Use****Block Parameter:** MatchingCondition**Type:** character vector**Values:** MATLAB code**Default:** '% match = isequal(trigger.Attribute1, entity.Attribute1);match = true;'**Event Actions****Event actions** — Specify the event action

OnFound (default) | Entry | Exit | Blocked

Specify the behavior of the entity on certain events. For example, the **Entry** action is called when the entity enters the block. To customize actions when entities enter, exit, or are found or blocked, enter MATLAB code in the **Entry action**, **Exit action**, **On found action**, or **Blocked action** field of the **Event actions** tab. For more information about event actions, see “Events and Event Actions”. For an example, see “Manage Entities Using Event Actions”

### Dependencies

**On found action** appears only if the **Extract found entities** check box is not selected.

**Entry action**, **Exit action**, and **Blocked action** appear only if the **Extract found entities** check box is selected.

### Programmatic Use

**Block Parameter:** EntryAction, ExitAction, BlockedAction

**Type:** character vector

**Values:** MATLAB code

**Default:** ''

### Statistics

**Number of entities found, f** — Outputs the number of entities that are found  
off (default) | on

Number of entities that are found by the block during simulation.

### Programmatic Use

**Block Parameter:** NumberEntitiesDeparted

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

**Number of entities departed, d** — Outputs the number of entities that have departed the block  
off (default) | on

Number of entities that have departed the block.

### Dependencies

This check box appears if the **Extract found entities** check box in the **Main** tab is selected.

### Programmatic Use

**Block Parameter:** NumberEntitiesDeparted

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

**Number of found entities in block, n** — Outputs the number of found entities that are yet to depart  
off (default) | on

Number of found entities that are yet to depart.

### Dependencies

This check box appears if the **Extract found entities** check box in the **Main** tab is selected.

**Programmatic Use****Block Parameter:** NumberEntitiesInBlock**Type:** character vector**Values:** 'on' | 'off'**Default:** 'off'**Average wait, w** — Outputs the average wait time

off (default) | on

Sum of the wait times for entities departing the block divided by their total number. Wait time is the duration between the Entity Find block entry and exit of an entity. For more information, see “Interpret SimEvents Models Using Statistical Analysis”.

**Dependencies**

This check box appears if the **Extract found entities** check box in the **Main** tab is selected.

**Programmatic Use****Block Parameter:** AverageWait**Type:** character vector**Values:** 'on' | 'off'**Default:** 'off'**Average queue length, l** — Outputs the average length of the entity queue

off (default) | on

Accumulated time-weighted average queue size. The block computes this value by:

- 1 Multiplying the size of the queue by its duration to calculate time-weighted queue size
- 2 Summing up all time-weighted queue sizes and averaging them over total time

For more information, see “Interpret SimEvents Models Using Statistical Analysis”.

**Dependencies**

This check box appears if the **Extract found entities** check box in the **Main** tab is selected.

**Programmatic Use****Block Parameter:** AverageStoreSize**Type:** character vector**Values:** 'on' | 'off'**Default:** 'off'**Number of entities extracted, ex** — Number of entities extracted from this block

off (default) | on

Outputs the number of extracted entities.

**Dependencies**

This check box appears if the **Extract found entities** check box in the **Main** tab is selected.

**Programmatic Use**

**Block Parameter:** NumEntitiesExtracted

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

## **Version History**

**Introduced in R2018b**

### **See Also**

Resource Releaser | Resource Pool | Resource Acquirer

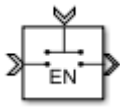
### **Topics**

“Model Using Resources”

“SimEvents Common Design Patterns”

# Entity Gate

Gate entities



**Libraries:**  
SimEvents

## Description

The Entity Gate block controls when pending entities can advance in the model.

The **Operating mode** parameter specifies how the pending entities advance through the gate.

- **Enable gate** — opens and allows entities to advance whenever the control port receives an anonymous entity with a positive value, and closes whenever it has zero or a negative value. For more information, see “Use Queue Event Actions to Model a Storage Tank”.
- **Release gate** — allows one pending entity to advance for each anonymous entity or message that arrives on the control port. At all other times, the entity input port of the block is unavailable.
- **Selection gate** — allows entities to advance whenever the anonymous entity value from the control port matches the attributes of the pending entities.

Use the Entity Gate block to control the flow of entities on the entity path. Use the Entity Output Switch block to select an output port for the departure of an entity among multiple entity output ports. For more information, see “Route Vehicles Using an Entity Output Switch Block”.

## Ports

### Input

**Port\_1** — Input entity  
scalar | vector | matrix

Input entity that carries scalar, bus, or vector data to enter the gate.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | string | fixed point

**control** — Incoming control entity  
scalar

Input control port to accept the entity that determines the state of the gate.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | fixed point

### Output

**Port\_1** — Exiting entity  
scalar

Output entity port for entities leaving the gate.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `enumerated` | `bus` | `string` | `fixed point`

## Parameters

**Operating mode** — Select the mode of operation

`Enable gate` (default) | `Release gate` | `Selection gate`

Select the mode of operation of this gate. With the **Operating mode** parameter set to:

- `Enable gate`, this block represents a gate that opens whenever the control port receives an anonymous entity with a positive value, and closes whenever it has zero or a negative value. By definition, an open gate permits entity arrivals as long as the entities are able to immediately advance to the next block, while a closed gate forbids entity arrivals. The anonymous entity that is received at the control port has a numerical value of type `double`. Since the gate receives an anonymous entity with a positive value and opens, an enabled gate remains open until it receives an entity with zero or a negative value and closes.
- `Release gate`, this block permits the arrival of one pending entity for each anonymous entity or message that arrives on the control port. At all other times, the entity input port of the block is unavailable. By definition, the opening of the gate permits one pending entity to arrive if the entity is able to immediately advance to the next block.
- `Selection gate`, this block permits the arrival of pending entities whenever the anonymous entity value from the control port matches the attributes of the pending entities. Otherwise it prevents the arrival of pending entities.

### Programmatic Use

**Block Parameter:** `OperatingMode`

**Type:** character vector

**Values:** `'Enable gate'` | `'Release gate'` | `'Selection gate'`

**Default:** `'Enable gate'`

**Matching attribute** — Specify name of the attribute that matches the value from the control port

`Attribute1` (default) | character vector

Name of the attribute to match the value from the control port.

### Dependencies

This parameter is visible when you set **Operating mode** to `Selection Gate`.

### Programmatic Use

**Block Parameter:** `MatchingAttributeName`

**Type:** character vector

**Values:** `'Attribute1'` | character vector

**Default:** `'Attribute1'`

**Initial value from the control port** — Specify initial value from the control port to match the matching attribute

`NaN` (default) | scalar



Specify the initial value to match the matching attribute which opens the gate.

#### Dependencies

This parameter is visible when you set **Operating mode** to Selection Gate.

#### Programmatic Use

**Block Parameter:** InitialValueOfMatchingAttribute

**Type:** character vector

**Values:** 'NaN' | scalar

**Default:** 'NaN'

**Open gate at simulation start** — Opening the gate at the start  
off (default) | on

Select this option to open the gate at the start of the simulation.

off

Gate is closed at the start of the simulation.

on

Gate is open at the start of the simulation.

#### Programmatic Use

**Block Parameter:** OpenGateAtSimulationStar

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

## Version History

Introduced in R2016a

### See Also

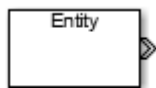
Entity Queue | Entity Multicast | Entity Server | Composite Entity Creator | Composite Entity Splitter  
| Entity Input Switch | Entity Output Switch | Discrete Event Chart | Entity Replicator

### Topics

“SimEvents Common Design Patterns”

# Entity Generator

Generate Entities



**Libraries:**  
SimEvents

## Description

The Entity Generator block generates entities. Entities are discrete items of interest that you can define in a discrete-event simulation. An entity can carry scalar, bus, or vector data. The meaning of an entity depends on the model. Entity can represent customers in a queuing system, data packets from a remote controller to an actuator, or any discrete item you define.

By default the block entity generation method is **Time-based**. In this method, the block generates entities using intergeneration times specified by the **Period**, from an input signal or statistical distribution. See “Entities in a SimEvents Model”, for more information about creating time-based and randomized entities.

The block also creates event-based entities. Choose **Event-based** as the **Generation Method** for an external event to specify the entity intergeneration time. For an example, see “Generate Entities When Events Occur”.

To customize actions when the entity is generated or it exits the block, in the **Event actions** tab, in the **Generate** action, or **Exit** action field, enter MATLAB code. For more information, see “Events and Event Actions”.

## Ports

### Input

**Port\_1** — Input to trigger entity generation upon arrival of events

scalar | vector | matrix

### Dependencies

To enable this port, click the **Entity generation** tab and select **Event-based** for the **Generation method**.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | fixed point

**Port\_2** — Input for the signal that determines the intergeneration time value for the next entity generation

scalar | vector | matrix

### Dependencies

To enable this port, click the **Entity generation** tab and select **Time-based** for the **Generation method** and **Signal** port for the **Time source**.

Data Types: double

## Output

**Port\_1** — Output generated entity

scalar | vector | matrix

Output port for the generated entities departing the generator.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | fixed point

**Port\_d** — Number of entities that have departed the block

scalar

Number of entities that have departed the block.

### Dependencies

To enable this port, click the **Statistics** tab and select the box labeled **Number of entities departed, d**.

Data Types: double

**Port\_pe** — Pending entity in block

off (default) | on

Outputs 1 for a pending entity, and 0 otherwise.

### Dependencies

To enable this port, select the **Statistics > Pending entity in block, pe**.

Data Types: double

**Port\_w** — Average intergeneration time

off (default) | on

Outputs the average time between generation of entities.

### Dependencies

To enable this port, select the **Statistics > Average intergeneration time, w**.

Data Types: double

## Parameters

**Generation method** — Select the method of entity generation

Time-based (default) | Event-based

Choose the entity generation method. Choose Time-based to generate entities using intergeneration times from an input signal or statistical distribution. Choose Event-based for an external event to determine the entity intergeneration time.

### Programmatic Use

**Block Parameter:** GenerationMethod

**Type:** character vector

**Values:** 'Time-based' | 'Event-based'  
**Default:** 'Time-based'

**Time source** — Select the source of the intergeneration time  
 Dialog (default) | Signal port | MATLAB action

Specify the source for entity intergeneration time.

- Select **Dialog** to specify a fixed period between entity generations.
- Select **Signal port** to generate entities based on an input signal.
- Select **MATLAB action** to define a MATLAB Script that defines the intergeneration time represented by *dt*.

For more information about specifying intergeneration times for entities, see “Specify Intergeneration Times for Entities”.

#### Dependencies

This parameter is visible when the **Generation method** is set to **Time-based**.

#### Programmatic Use

**Block Parameter:** TimeSource

**Type:** character vector

**Values:** 'Dialog' | 'Signal port' | 'MATLAB action'

**Default:** 'Dialog'

**Period** — Define the period between the generation of entities  
 1 (default) | scalar

Specify the time between entity intergeneration. For instance, if the **Period** is 1, the block waits 1 simulation time in between entity generations. See, “Specify Intergeneration Times for Entities” for more information.

**Tunable:** Yes

#### Dependencies

This parameter is visible when **Generation method** is set to **Time-based**.

#### Programmatic Use

**Block Parameter:** Period

**Type:** character vector

**Values:** '1' | scalar

**Default:** '1'

**Intergeneration time action** — Specify the time between entity generations  
 $dt = rand(1,1)$  (default) | MATLAB code

Use MATLAB code to specify service time. *dt* specifies the time between entity generations. You can manually specify *dt* or use **Insert pattern** button to generate entities with a repeating sequence or from a distribution. The block uses this parameter every time it is ready for entity generation. For an example, see “Specify Intergeneration Times for Entities”.

**Dependencies**

This parameter is visible when **Service time source** is set to MATLAB action.

**Programmatic Use**

**Block Parameter:** IntergenerationTimeAction

**Type:** character vector

**Values:** MATLAB code

**Default:** 'dt = rand(1,1);'

**Generate entity at simulation start** — Generate an entity at the start of the simulation  
on (default) | off

Generates entity at the start of the simulation.

**Programmatic Use**

**Block Parameter:** GenerateEntityAtSimulationStart






**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'on'

**Entity type** — Choose the type of entity to generate  
Structured (default) | Anonymous | Bus object

Choose the type of entity to generate.

- The Anonymous type has one data value associated with it.
- The Structured type (default) includes name and initial value attributes that you can specify.
  - You can attach more than one attribute to an entity by clicking .
  - You can delete attributes by clicking .
  - You can change the order of the attributes by clicking  and .
  - You can convert a Structured type entity to a bus object by clicking .
- The Bus object type lets you generate bus objects as entities.

Click **Launch Type Editor** to open the Type Editor to generate bus objects. A bus object can be an element of another bus object which can be used to create hierarchy in the data that is attached to an entity.

For more information, see “Entities in a SimEvents Model”.

**Programmatic Use**

**Block Parameter:** EntityType

**Type:** character vector

**Values:** 'Structured' | 'Anonymous' | 'Bus object'

**Default:** 'Structured'

**Entity priority** — Specify the priority of the generated entity  
300 (default) | scalar

Determines the priority of the generated entity. The lower the value the higher the priority For more information, see “Working with Entity Attributes and Entity Priorities”.

**Programmatic Use****Block Parameter:** EntityPriority**Type:** character vector**Values:** '300' | scalar**Default:** '300'**Entity type name** — Specify the name of the generated entity

Entity (default) | character vector

Determines the name of the generated entity.

**Dependencies**

This parameter is visible when **Entity type** is set to Bus object or Structured.

**Programmatic Use****Block Parameter:** EntityTypeName**Type:** character vector**Values:** 'Entity' | character vector**Default:** 'Entity'**Data initial value** — Specify the initial value of anonymous entity data

0 (default) | scalar | vector | matrix

Set the anonymous entity data initial value. This value cannot be of type int64 or fixed-point.

**Dependencies**

This parameter is visible when you set **Entity type** to Anonymous.

**Programmatic Use****Block Parameter:** DataInitialValue**Type:** character vector**Values:** '0' | scalar | vector | matrix**Default:** '0'**Attribute Name** — Define the name of the generated entity attribute

Attribute1 (default) | character vector

Define entity attribute name.

---

**Note** When done, you can export the structured entity type as a bus object, with the name **Entity type name**, to the base workspace. Export the bus object when using the MATLAB Discrete-Event System and Discrete Event Chart blocks.

---

**Dependencies**

This parameter is visible when **Entity type** is set to Structured.

**Programmatic Use****Block Parameter:** AttributeName**Type:** character vector**Values:** 'Attribute1' | character vector**Default:** 'Attribute1'**Attribute Initial Value** — Define the generated entity attribute initial value

1 (default) | scalar

Specify the entity attribute initial value. This parameter is visible when **Entity type** is set to Structured. This value can not be of type fixed-point.

**Programmatic Use****Block Parameter:** AttributeInitialValue**Type:** character vector**Values:** 1 | scalar**Default:** '1'**Event actions** — Specify the behavior of the entity on certain events

Generate (default) | Exit

Define the behavior in the **Event action** parameter. The Generate action is called when an entity is generated and the Exit action is called just before an entity exits the block.

**Programmatic Use****Block Parameter:** GenerateAction, ExitAction**Type:** character vector**Values:** MATLAB code**Default:** ''**Number of entities departed, d** — Outputs the number of entities that have departed the block

off (default) | on

Number of entities that have departed the block.

**Programmatic Use****Block Parameter:** NumberEntitiesDeparted**Type:** character vector**Values:** 'on' | 'off'**Default:** 'off'**Pending entity present in block, pe** — Pending entities

off (default) | on

Indicates whether an entity that is yet to depart is present in the block. The value is 1 for a pending entity, and 0 otherwise. This block can have at most one pending entity because its storage capacity is one. If there is an existing pending entity, the block does not generate another entity until the pending entity departs the block.

**Programmatic Use****Block Parameter:** PendingEntityInBlock**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

**Average intergeneration time, w** — Average time between generation of entities

off (default) | on

Outputs the average time between generation of entities. **Average intergeneration time, w** is the ratio of the total generation time to the total number of generated entities.

**Programmatic Use**

**Block Parameter:** AverageIntergenerationTime

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

## Version History

Introduced in R2016a

### See Also

Entity Queue | Entity Multicast | Entity Gate | Composite Entity Creator | Composite Entity Splitter | Entity Input Switch | Entity Output Switch | Entity Server | Discrete Event Chart | Multicast Receive Queue | Entity Multicast

### Topics

“Generate Multiple Entities at Time Zero”

“Specify Intergeneration Times for Entities”

“Generate Entities When Events Occur”

“SimEvents Common Design Patterns”

“Entities in a SimEvents Model”



# Entity Input Switch

Switch input entities



**Libraries:**  
SimEvents

## Description

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.

---

**Note** If you want to merge message or entity paths and generate code for your component interface, use the Simulink Message Merge block. Message Merge block's behavior is the same as the Entity Output Switch block with **Active port selection** parameter set to All.

---

For an example, see “Generate Entities When Events Occur”.

## Combine Entity 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.

### Select Arrival Path

The Entity Input Switch block allows arrival of entities at its ports. The selected entity input port can change during the simulation.

You can also select the criterion for switching between input ports.

- Select `Round robin` to select ports in a round robin fashion.
- Select `From control port` to let the control port determine the selected port.
- Select `Equiprobable` to let the block randomly select any port with equal probability.

## Ports

### Input

#### **Port\_1** — Input entity

scalar | vector | matrix

Input entity port for entities entering the block.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `enumerated` | `bus` | `fixed point`

#### **Port\_2** — Input entity

scalar | vector | matrix

Input entity port for entities entering the block.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `enumerated` | `bus` | `fixed point`

#### **control** — Incoming control entity

scalar

Input control port for the incoming control entity that determines the input port for the entities arriving at the block.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

### Output

#### **Port\_1** — Output entity

scalar | vector | matrix

Output entity port for entities exiting the block.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `enumerated` | `bus` | `fixed point`

## Parameters

**Number of input ports** — Specify the number of input ports  
2 (default) | scalar

Determines how many entity input ports the block has.

### Programmatic Use

**Block Parameter:** `NumberInputPorts`

**Type:** character vector

**Values:** '2' | scalar

**Default:** '2'

**Active port selection** — Specify the active input port  
All (default) | `Switch`

Specify the active input port to allow arrival of entities at all ports or one port at a time. Select `All` to combine entity paths and allow arrival of entities at all ports. Select `Switch` to allow arrival of an entity at only one port at a time.

### Programmatic Use

**Block Parameter:** `ActivePortSelection`

**Type:** character vector

**Values:** 'All' | 'Switch'

**Default:** 'All'

**Switching criterion** — Specify input port switch criterion  
`Round robin` (default) | `From control port` | `Equiprobable`

Select the criterion for switching between input ports.

- Select `Round robin` to select ports in a round robin fashion. Set the initial port in the **Initial port selection**.
- Select `From control port` to let the control port determine the selected port. A control port will appear to input an anonymous entity carrying data with a value greater than 0, and smaller than or equal to the number of input ports to determine the active port.
- Select `Equiprobable` to let the block randomly select any port with equal probability. Set the **Seed** to generate a random number and to determine the active input port.

### Programmatic Use

**Block Parameter:** `SwitchingCriterion`

**Type:** character vector

**Values:** 'Round robin' | 'From control port' | 'Equiprobable'

**Default:** 'Round robin'

**Initial port selection** — Specify the initial input port for entity entry  
1 (default) | scalar

Specify initially which port allows arrival of an entity.

**Programmatic Use**

**Block Parameter:** InitialPortSelection

**Type:** character vector

**Values:** '1' | scalar

**Default:** '1'

**Seed** — Specify the seed for the random number generator to determine the input port  
23453 (default) | scalar

Specify the seed for the random number generator to determine the input port.

**Dependencies**

This parameter is visible when **Switching criterion** is set to Equiprobable.

**Programmatic Use**

**Block Parameter:** Seed

**Type:** character vector

**Values:** '23453' | scalar

**Default:** '23453'

## Version History

Introduced in R2016a

### See Also

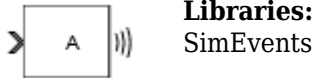
Entity Queue | Entity Gate | Composite Entity Creator | Composite Entity Splitter | Entity Output Switch | Multicast Receive Queue | Entity Multicast | Entity Replicator | Entity Terminator

### Topics

“SimEvents Common Design Patterns”

# Entity Multicast

Send multicast entities



## Description

The Entity Multicast block broadcasts entities. An entity that arrives at the block is cloned into copies and sent with a **Multicast tag**. Each Entity Queue block with **Entity arrival source** specified as **Multicast** with the same **Multicast tag** receives copies. For more information, see “Overview of Queues and Servers in Discrete-Event Simulation”.

## Ports

### Input

**Port\_1** — Input entity  
scalar | vector | matrix

Input entity port for entities entering the block.

---

**Note** Event actions are not supported with string entity data type

---

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | string | fixed point

### Output

**send** — Broadcast entities  
scalar | vector | matrix

Output broadcasted entities.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | string | fixed point

## Parameters

**Multicast tag** — Specify the tag with which to broadcast the entities  
A (default) | character vector

Specify the tag with which to broadcast the entities. For example if the default tag A is used, each Entity Queue block with **Entity arrival source** specified as **Multicast** and the **Multicast tag** is set as A receives a copy.

**Programmatic Use**

**Block Parameter:** MulticastTag

**Type:** character vector

**Values:** 'A' | character vector

**Default:** 'A'

## **Version History**

**Introduced in R2016a**

### **See Also**

Entity Generator | Multicast Receive Queue

# Entity Output Switch

Output entities



**Libraries:**  
SimEvents

## Description

The Entity Output Switch block allows you to select an output port for the departure of an entity among multiple entity output ports. The selected port can change during the simulation and you can choose the criterion for switching between output ports.

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. For an example, see “Route Vehicles Using an Entity Output Switch Block”.

- 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.

Alternatively, you can use the Entity Gate block to control the flow of entities on a specific entity path. For more information, see “Using Entity Priority to Sequence Departures”.

## Ports

### Input

**Port\_1** — Input entity

scalar | vector | matrix

Input entity port for entities entering the block.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `enumerated` | `bus` | `fixed point`

**control** — Incoming control entity

scalar

Input control port for the incoming control entity that determines the output port for the entities departing from the block.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

## Output

### **Port\_1** — Output entity

`scalar` | `vector` | `matrix`

Output entity port for entities exiting the block.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `enumerated` | `bus` | `fixed point`

### **Port\_2** — Output entity

`scalar` | `vector` | `matrix`

Output entity port for entities exiting the block.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `enumerated` | `bus` | `fixed point`

## Parameters

### **Number of output ports** — Specify the number of output ports

`2` (default) | `scalar`

Number of output ports for entity departure.

#### **Programmatic Use**

**Block Parameter:** `NumberOutputPorts`

**Type:** `character vector`

**Values:** `'2'` | `scalar`

**Default:** `'2'`

### **Switching criterion** — Choose the switching criterion

`First port that is not blocked` (default) | `Round robin` | `From control port` | `From attribute` | `Equiprobable`

Choose the criterion for switching between output ports.

- To output the entity to the first unblocked port, select `First port that is not blocked`.

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.

- To output entities in a round robin fashion among the output ports, select `Round robin`.
- Select `From control port` to let the control port determine the selected port for entity departure. A control port will appear to input an anonymous entity carrying data with a value greater than 0 and smaller than or equal to the number of output ports to determine the active port.



- To specify an attribute that determines the output port, select `From attribute`. The attribute value is greater than 0 and smaller than or equal to the number of output ports to determine the active 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 randomly select an output port for entity departure, select `Equiprobable`. Set the **Seed** to generate a random number and to determine the active output port.

---

**Note** The block rounds a double precision value to the nearest integer less than or equal to its value as port selection. For instance, the value 0.3 is rounded of to 0 which is not a valid value for port selection.

---

#### Programmatic Use

**Block Parameter:** `SwitchingCriterion`

**Type:** character vector

**Values:** 'First port that is not blocked' | 'Round robin' | 'From control port' | 'From attribute' | 'Equiprobable'

**Default:** 'First port that is not blocked'

**Initial port selection** — Specify the output port at the start of the simulation

1 (default) | scalar

Select the initial port for the entity departure.

#### Dependencies

This parameter is visible when **Switching criterion** is set to `Round robin` or `From control port`.

#### Programmatic Use

**Block Parameter:** `InitialPortSelection`

**Type:** character vector

**Values:** '1' | scalar

**Default:** '1'

**Switch Attribute name** — Specify the attribute that determines the output port

`Attribute1` (default) | character vector

Specify the attribute name used to switch the output port.

#### Dependencies

This parameter is visible when **Switching criterion** is set to `From attribute`.

#### Programmatic Use

**Block Parameter:** `SwitchAttributeName`

**Type:** character vector

**Values:** 'Attribute1' | character vector

**Default:** 'Attribute1'

**Seed** — Specify the seed  
34567 (default) | scalar

Specify the seed for the random number generator to determine the output port.

**Dependencies**

This parameter is visible when **Switching criterion** is set to Equiprobable.

**Programmatic Use**

**Block Parameter:** Seed

**Type:** character vector

**Values:** '34567' | scalar

**Default:** '34567'

## **Version History**

**Introduced in R2016a**

### **See Also**

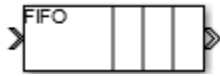
Entity Queue | Entity Gate | Composite Entity Creator | Composite Entity Splitter | Entity Input Switch | Multicast Receive Queue | Entity Multicast | Entity Replicator | Entity Terminator

### **Topics**

“SimEvents Common Design Patterns”

# Queue, Entity Queue

Enqueue messages and entities



**Libraries:**  
 Simulink / Messages & Events  
 SimEvents

## Description

This block stores entities or messages in a queue, based on the order of arrival or priority. Each element at the head of the queue departs when the downstream block is ready to accept it. The Queue block and the Entity Queue block are the same blocks with different default values for the **Overwrite the oldest element if queue is full** check box.

You can specify the capacity of the queue, and the policy when the queue is full. The block supports three different message or queue sorting policies, first-in-first out (FIFO), last-in-first out (LIFO), and priority. The priority queue can be used only when the **Overwrite the oldest element if queue is full** check box is cleared.

## Ports

### Input

**Port\_1** — Input entity or message

scalar | vector | matrix

Input entity or message that carries scalar, bus, or vector data to enter the queue.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | fixed point

### Output

**Port\_1** — Output entity or message

scalar | vector | matrix

Output port that allows entities or messages at the head of the queue to depart when a downstream block is ready to accept them.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | fixed point

**Port\_d** — Number of entities that have departed the block

scalar

Number of entities that have departed the block.

### Dependencies

To enable this port, select **Overwrite the oldest element if queue is full** check box, and click the **Statistics** tab and select the box labeled **Number of entities departed, d**.

Data Types: double

**Port\_n** — Number of entities that have not yet departed the block  
scalar

Number of entities that have not yet departed the block.

#### **Dependencies**

To enable this port, select **Overwrite the oldest element if queue is full** check box, and click the **Statistics** tab and select the box labeled **Number of entities in block, n**.

Data Types: double

**Port\_w** — Average wait time for entities in the block  
scalar

Average wait time for entities in the block.

#### **Dependencies**

To enable this port, select **Overwrite the oldest element if queue is full** check box, and click the **Statistics** tab and select the box labeled **Average wait, w**.

Data Types: double

**Port\_l** — Average length of the entity queue  
scalar

**Port\_l** outputs the average length of the entity queue.

#### **Dependencies**

To enable this port, select **Overwrite the oldest element if queue is full** check box, and click the **Statistics** tab and select the box labeled **Average queue length, l**.

Data Types: double

**Port\_ex** — Number of entities extracted  
scalar

Number of entities that are pulled out of this block.

#### **Dependencies**

To enable this port, select **Overwrite the oldest element if queue is full** check box, and click the **Statistics** tab and select the box labeled **Number of entities extracted, ex**.

Data Types: double

## **Parameters**

**Overwrite the oldest element if queue is full** — Specify queue overwriting policy  
on (default for Simulink) | off (default for SimEvents)

Select this check box to choose between two queue overwriting policies.

- If you select the check box, an incoming message overwrites the oldest if the queue is full.

This mode represents a simple message buffer that you can use to generate asynchronous communication between Simulink components and production code.

- If you clear the check box, the block does not accept new messages if the queue is full.

In this mode, you can manipulate entity data using event actions and visualize statistics.

To customize actions when entities or messages enter, exit, or are blocked, enter MATLAB code in the **Entry action**, **Exit action**, or **Blocked action** field of the **Event actions** tab. For more information, see “Events and Event Actions”.

For an example, see “Manage Entities Using Event Actions”.

#### **Programmatic Use**

**Block Parameter:** QueueOverwriting

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'on' (for Simulink) and 'off' (for SimEvents)

**Capacity** — Specify the capacity of the queue

25 (default) | scalar

Specify the capacity of the queue.

#### **Programmatic Use**

**Block Parameter:** Capacity

**Type:** character vector

**Values:** '25' | scalar

**Default:** '25'

**Queue type** — Choose the queue type

FIFO (default) | LIFO | Priority

Choose the queue type.

- FIFO — first-in-first-out
- LIFO — last-in-first-out
- Priority — store elements in order of priority, see “Serve High-Priority Customers by Sorting Entities Based on Priority”. Priority can be selected when you clear the **Overwrite the oldest element if queue is full** check box.

---

**Note** Priority queue does not support fixed point data type.

---

#### **Programmatic Use**

**Block Parameter:** QueueType

**Type:** character vector

**Values:** 'FIFO' | 'LIFO' | 'Priority'

**Default:** 'FIFO'

**Multicast tag** — Specify the tag when accepting entities broadcast via multicast sources

A (default) | character vector

Specify the tag when accepting entities broadcast via multicast sources. The Entity Multicast block requires SimEvents license.

**Dependencies**

This parameter is visible when you clear the **Overwrite the oldest element if queue is full** check box, and set **Entity arrival source** to Multicast.

**Programmatic Use**

**Block Parameter:** MulticastTag

**Type:** character vector

**Values:** 'A' | character vector

**Default:** 'A'

**Priority source** — Specify which attribute of the entity determines its priority

PriorityAttribute (default) | character vector

Specify which attribute of the entity determines its priority.

**Dependencies**

This parameter is visible when you clear the **Overwrite the oldest element if queue is full** check box, and set **Queue type** to Priority.

**Programmatic Use**

**Block Parameter:** PrioritySource

**Type:** character vector

**Values:** 'PriorityAttribute' | character vector

**Default:** 'PriorityAttribute'

**Sorting direction** — Choose the direction of sorting based on priority

Ascending (default) | Descending

Choose the direction of sorting based on priority.

- Ascending — Elements with smaller priority values appear in front of the queue.
- Descending — Elements with greater priority values appear in front of the queue.

**Dependencies**

This parameter is visible when you clear the **Overwrite the oldest element if queue is full** check box, and set **Queue type** to Priority.

**Programmatic Use**

**Block Parameter:** SortingDirection

**Type:** character vector

**Values:** 'Ascending' | 'Descending'

**Default:** 'Ascending'

**Entity arrival source** — Choose the source of arrival for the entities

Input port (default) | Multicast

Choose the source of arrival for the entities.

- **Input port** — Input port is source of messages or entities.
- **Multicast** — Entity Multicast block is source of entities. The Entity Multicast block requires SimEvents license.

#### Dependencies

This parameter is visible when you clear the **Overwrite the oldest element if queue is full** check box, and set **Queue type** to Priority.

#### Programmatic Use

**Block Parameter:** EntityArrivalSource

**Type:** character vector

**Values:** 'Input port' | 'Multicast'

**Default:** 'Input port'

**Event action** — Specify the behavior of the entity in certain events

Entry (default) | Exit | Blocked

Specify the behavior of the entity in certain events. Define the behavior in the **Event action** parameter. The **Entry** and the **Exit** actions are called just after the entity entry and just before entity exit. The **Blocked** action is called after an entity is blocked. For more information, see “Events and Event Actions”.

---

**Note** If an event action changes an entity, related block behavior such as resorting a priority queue, and rescheduling of any events, will occur after the event action has fully finished and returned.

---



---

**Note** Event actions do not support fixed point data type.

---

#### Dependencies

Event actions are visible when you clear the **Overwrite the oldest element if queue is full** check box.

#### Programmatic Use

**Block Parameter:** EntryAction, ExitAction, BlockedAction

**Type:** character vector

**Values:** MATLAB code

**Default:** ''

**Number of entities departed, d** — Outputs the number of entities that have departed the block  
off (default) | on

Number of entities that have departed the block.

#### Dependencies

**Number of entities departed, d** is visible when you clear the **Overwrite the oldest element if queue is full** check box.

**Programmatic Use****Block Parameter:** NumberEntitiesDeparted**Type:** character vector**Values:** 'on' | 'off'**Default:** 'off'

**Number of entities in block, n** — Outputs the number of entities present in the block, that are yet to depart

off (default) | on

Number of entities present in the block, but which are yet to depart.

**Dependencies**

**Number of entities in block, n** is visible when you clear the **Overwrite the oldest element if queue is full** check box.

**Programmatic Use****Block Parameter:** NumberEntitiesInBlock**Type:** character vector**Values:** 'on' | 'off'**Default:** 'off'

**Average wait, w** — Outputs the average wait time

off (default) | on

Sum of the wait times for entities departing the block divided by their total number. Wait time is the duration between the Entity Queue block entry and exit of an entity. For more information, see “Interpret SimEvents Models Using Statistical Analysis”.

**Dependencies**

**Average wait, w** is visible when you clear the **Overwrite the oldest element if queue is full** check box.

**Programmatic Use****Block Parameter:** AverageWait**Type:** character vector**Values:** 'on' | 'off'**Default:** 'off'

**Average queue length, l** — Outputs the average length of the entity queue

off (default) | on

Accumulated time-weighted average queue size. The block computes this value by:

- 1 Multiplying the size of the queue by its duration to calculate time-weighted queue size
- 2 Summing up all time-weighted queue sizes and averaging them over total time

For more information, see “Interpret SimEvents Models Using Statistical Analysis”.



**Dependencies**

**Average queue length, l** is visible when you clear the **Overwrite the oldest element if queue is full** check box.

**Programmatic Use**

**Block Parameter:** AverageQueueLength

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

**Number of entities extracted, ex** — Number of entities extracted from this block

off (default) | on

Outputs the number of extracted entities which are pulled out from this block by the Entity Find block. The Entity Find block requires a SimEvents license. If the extracted entity is the first entity in the queue, the next entity is set as the pending entity to leave the block. If an entity attribute defines the priority in a priority queue and it is modified by the Entity Find block, the queue is sorted again. When an entity is extracted, **Number of entities departed, d**, **Number of entities in block, n**, **Average wait, w**, and **Average queue length, l** statistics are updated accordingly. For more information about finding and extracting entities, see “Find and Extract Entities in SimEvents Models”.

**Dependencies**

**Number of entities extracted, ex** is visible when you clear the **Overwrite the oldest element if queue is full** check box.

**Programmatic Use**

**Block Parameter:** NumEntitiesExtracted

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

**Block Characteristics**

<b>Data Types</b>	Boolean   bus   double   enumerated   fixed point   integer   single   string
<b>Direct Feedthrough</b>	no
<b>Multidimensional Signals</b>	yes
<b>Variable-Size Signals</b>	no
<b>Zero-Crossing Detection</b>	no

**Version History**

Introduced in R2016a

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

Code generation is not supported for event actions and statistics.

### **See Also**

Entity Generator | Entity Server | Multicast Receive Queue | Entity Multicast | Message Send | Message Receive

### **Topics**

“Overview of Queues and Servers in Discrete-Event Simulation”  
“SimEvents Common Design Patterns”

# Entity Replicator

Replicate entities



**Libraries:**  
SimEvents

## Description

The Entity Replicator block duplicates entities. It outputs replica entities and can also output the original entity. The block provides an output port for the original entity.

If the original entity departs the block before the replicas, then its replicas are destroyed. Selecting the **Hold original entity until all replicas depart** check box ensures that the replicas depart the block before the original entity.

An original entity can be extracted from this block by the Entity Find block. If an original entity waiting in the Entity Replicator block is extracted, all the replicas are destroyed. Only the original entities can be extracted because the replicas can not acquire resources in this block. For more information about finding and extracting entities, see “Find and Extract Entities in SimEvents Models”.

## Ports

### Input

**Port\_1** — Input entity

scalar | vector | matrix

Input entity port for entities entering the block.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | string | fixed point

### Output

**Port\_1** — Output replica entity

scalar | vector | matrix

Output entity port for replica entities exiting the block.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | string | fixed point

**Port\_2** — Output the entity that is replicated

scalar | vector | matrix

Output entity port for original entities exiting the block.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | string | fixed point

## Parameters

**Replicas depart from** — Choose how replicas depart  
Separate output ports (default) | Single output port

Choose if the replicas depart from separate output ports or a single output port.

- Separate output ports — Outputs one replica entity from each output port
- Single output port — Outputs all replicas from a single output port

### Programmatic Use

**Block Parameter:** ReplicasDepartFrom

**Type:** character vector

**Values:** 'Separate output ports' | 'Single output port'

**Default:** 'Separate output ports'

**Replication amount source** — Specify source of replica number  
Dialog (default) | Attribute

Specify the source of replica number.

- Select Dialog to specify the number of replicas in the dialog box.
- Select Attribute to select an attribute that specifies the number of replicas.

### Dependencies

This parameter is visible when you set **Replicas depart from** to Single output port.

### Programmatic Use

**Block Parameter:** ReplicationAmountSource

**Type:** character vector

**Values:** 'Dialog' | 'Attribute'

**Default:** 'Dialog'

**Number of replicas** — Specify the number of replicas  
1 (default) | scalar

Specify the number of replicas. If you select Single output port, all replicas depart from this output port. If you select Separate output ports, each replica has its own port.

### Dependencies

This parameter is visible when you set **Replicas depart from** to Separate output ports or Single output port and **Replication amount source** to Dialog.

### Programmatic Use

**Block Parameter:** NumberReplicas

**Type:** character vector

**Values:** '1' | scalar

**Default:** '1'

**Replicate attribute name** — Specify the attribute that determines the number of replicas  
ReplicateAttribute (default) | character vector

## Dependencies

Specify the attribute that determines the number of replicas. This parameter is visible when you set **Replicas depart from** to `Single` output port and **Replication amount source** to `Attribute`.

### Programmatic Use

**Block Parameter:** `ReplicateAttributeName`

**Type:** character vector

**Values:** `'ReplicateAttribute'` | character vector

**Default:** `'ReplicateAttribute'`

**Hold original entity until all replicas depart** — Choose how replicas depart  
`off` (default) | `on`

Select this check box to hold the original entity until all the replicas have departed. The block first attempts to send all the replicas before it sends out the original entity. Selecting this check box prevents destruction of replica entities when they do not depart the block before the original entity.

### Programmatic Use

**Block Parameter:** `HoldOriginalEntityUntilAllReplicasDepart`

**Type:** character vector

**Values:** `'on'` | `'off'`

**Default:** `'off'`

## Version History

Introduced in R2016a

## See Also

Entity Generator | Entity Server

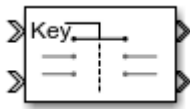
## Topics

“Enable a Gate for a Time Interval”

“SimEvents Common Design Patterns”

## Entity Selector

Select entities



**Libraries:**  
SimEvents

### Description

The Entity Selector block selects entities from multiple streams of ready-to-leave entities, and matches them to a reference entity. You can use this block for modeling scenarios which involve items to be matched based on input. For instance, you can model a facility that produces parts which are matched to the corresponding order.

The block first accepts a reference entity with its **Key entity attribute name**. Then the block selects a matching entity with **Matching entity attribute name(s)** from each of the other input ports accepting incoming entities. The match is based on the equality of the specified attribute values. When a match is found across all the entity streams, the set of matching entities and the key entity become ready to depart.

The Entity Store block can be used as a temporary container for entities to be selected by the Entity Selector block. For an example, see “Match Entities Based on Attributes”.

### Ports

#### Input

**Key** — Incoming reference entity

scalar | vector | matrix

Input entity port for reference entities entering the block.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | fixed point

**Port\_1** — Incoming entity

scalar | vector | matrix

Input entity port for matching entities entering the block.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | fixed point

#### Output

**Port\_1** — Exiting reference entity

scalar | vector | matrix

Output entity port for reference entities exiting the block.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `enumerated` | `bus` | `fixed point`

**Port\_2** — Exiting matched entity

`scalar` | `vector` | `matrix`

Output entity port for matching entities exiting the selector.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `enumerated` | `bus` | `fixed point`

**Port\_d** — Number of entities that have departed the block

`scalar`

Number of entities that have departed the block.

**Dependencies**

To enable this port, click **Statistics** and select the **Number of entities departed, d** check box.

Data Types: `double`

**Port\_n** — Number of entities that have not yet departed the block

`scalar`

Number of entities that have not yet departed the block.

**Dependencies**

To enable this block, click **Statistics** and select the **Number of entities in block, n** check box.

Data Types: `double`

**Port\_ex** — Number of entities extracted

`scalar`

Number of entities that are pulled out of this block.

**Dependencies**

To enable this port, click the **Statistics** tab and select the box labeled **Number of entities extracted, ex**.

Data Types: `double`

## Parameters

**Number of matching streams** — Determine how many input entity streams the block accepts

1 (default) | `numerical`

Specify the number of entity streams to be matched to the reference entity. The block can accept at most 8 matching streams.

**Programmatic Use**

**Block Parameter:** `mNumberOfStreams`

**Type:** `character vector`

**Values:** '1' | scalar

**Default:** '1'

**Key entity attribute Name** — Specify name of reference attribute

{ 'ID' } (default) | character vector

Name of the reference attribute that is used to evaluate a match.

**Programmatic Use**

**Block Parameter:** mKeyEntityAttributeName

**Type:** character vector

**Values:** '{ 'ID' }' | character vector

**Default:** '{ 'ID' }'

**Matching entity attribute name(s)** — Specify name of selected attribute names

{ 'Type' } (default) | character vector

Name of the matching entity attribute that is used to evaluate a match. You can specify one attribute name, or an array of attribute names that are compared with the key value to evaluate a match.

**Programmatic Use**

**Block Parameter:** mEntityAttributeName

**Type:** character vector

**Values:** '{ 'Type' }' | character vector

**Default:** '{ 'Type' }'

**Number of entities source** — Source to determine the number of entities to be matched

Dialog (default) | Attribute

Source that determines the number of entities to be selected from each stream.

**Programmatic Use**

**Block Parameter:** mMatchingNumberSource

**Type:** character vector

**Values:** 'Dialog' | 'Attribute'

**Default:** 'Dialog'

**Number of entities to select** — Number of entities to select from each of the incoming entity streams

1 (default) | scalar

Number of entities to be selected from each matching stream. You can specify 0, a positive integer, or an array of positive integers to determine the number of entities to select from each matching entity stream.

**Dependencies**

To enable this parameter, set the **Number of entities source** parameter to Dialog.

**Programmatic Use**

**Block Parameter:** mNumberOfMatches

**Type:** character vector



**Values:** '1' | scalar

**Default:** '1'

**Key attribute for number of entities** — Key attribute name that determines the number of entities to be matched

Name (default) | character vector

The name of the key attribute that determines the number of entities to be selected from each matching stream. The attribute value can be an integer or an array of integers of size equal to the number of incoming entity streams.

#### Dependencies

To enable this parameter, set the **Number of entities source** parameter to Attribute.

#### Programmatic Use

**Block Parameter:** mNumberOfMatchesAttribute

**Type:** character vector

**Values:** 'Name' | character vector

**Default:** 'Name'

**Number of entities departed, d** — Outputs the number of entities that have departed the block

off (default) | on

Number of entities that have exited the block.

#### Programmatic Use

**Block Parameter:** mNumEntitiesDepOpt

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

**Number of entities in block, n** — Outputs the number of entities present in the block, which have yet to depart

off (default) | on

Number of entities present in the block, which have yet to depart.

#### Programmatic Use

**Block Parameter:** mNumEntitiesInBlockOpt

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

**Number of entities extracted, ex** — Outputs the number of entities extracted from the block

off (default) | on

Outputs the number of extracted entities which are pulled out from this block by the Entity Find block. If a key entity is extracted, the Entity Selector block waits until all the matching entities arrive at the block. Then, the matching entities depart from the corresponding output port. If a matching entity is extracted, the block looks for another matching entity. When an entity is extracted, **Number of entities departed, d**, and **Number of entities in block, n** statistics are updated accordingly.

**Programmatic Use**

**Block Parameter:** mNumEntitiesExtracted0pt

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

## **Version History**

**Introduced in R2018a**

### **See Also**

Entity Queue | Entity Store | Entity Multicast | Entity Server | Composite Entity Creator | Composite Entity Splitter | Entity Input Switch | Entity Output Switch | Discrete Event Chart | Entity Replicator

### **Topics**

“SimEvents Common Design Patterns”

“Match Entities Based on Attributes”

# Entity Server

Serve entities



**Libraries:**  
SimEvents

## Description

The Entity Server block serves entities as they arrive. In a discrete-event simulation, a server stores entities for a length of time, called service time, and then attempts to output the entity. During the service period, the block is said to be serving the entity that it stores. The block can serve multiple entities simultaneously and output each entity through the output port, unless the port is blocked. When the block permits preemption, an entity in the server can depart early through a second port.

To customize actions when entities enter, complete service, exit, and are blocked or preempted by the block, enter MATLAB code in the **Entry** action, **Service complete** action, **Exit** action, **Blocked** action, or **Preempt** action field of the **Event actions** tab. For more information, see “Events and Event Actions”.

## Ports

### Input

**Port\_1** — Input entity  
scalar | vector | matrix

Input entity that carries scalar, bus, or vector data to enter the server.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | string | fixed point

**Port\_2** — Input signal port for service time source  
scalar

Input signal port to determine service time.

### Dependencies

This port is visible when **Service time source** is set to **Signal** port.

Data Types: double

### Output

**Port\_1** — Output entity  
scalar | vector | matrix

Output entity port for entities exiting the server.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | string | fixed point

**Port\_d** — Number of entities that have departed the block  
scalar

Number of entities that have departed the block.

**Dependencies**

To enable this port, click the **Statistics** tab and select the box labeled **Number of entities departed, d**.

Data Types: double

**Port\_n** — Number of entities that have not yet departed the block  
scalar

Number of entities that have not yet departed the block.

**Dependencies**

To enable this port, click the **Statistics** tab and select the box labeled **Number of entities in block, n**.

Data Types: double

**Port\_pe** — Pending entity in block  
off (default) | on

Selecting this check box outputs the value 1 for a pending entity in the block, and 0 otherwise.

**Dependencies**

To enable this port, select the **Statistics > Pending entity in block, pe**.

Data Types: double

**Port\_np** — Number of pending entities  
off (default) | on

Selecting this check box outputs the number of pending entities in the block.

**Dependencies**

To enable this port, select the **Statistics > Number of pending entities, np**.

Data Types: double

**Port\_w** — Average wait time for entities in the block  
scalar

Average wait time for entities in the block.

**Dependencies**

To enable this port, click the **Statistics** tab and select the box labeled **Average wait, w**.

Data Types: double

**Port\_util** — Outputs the average time the server is occupied  
scalar

Average time the server is occupied.

#### Dependencies

To enable this port, click the **Statistics** tab and select the box labeled **Utilization, util**.

Data Types: double

**Port\_p** — Outputs the number of preempted entities

scalar

Number of preempted entities.

#### Dependencies

To enable this port, first click the **Preemption** tab, then click **Statistics** tab and select the box labeled **Number of entities preempted, p**.

Data Types: double

**Port\_ex** — Number of entities extracted

scalar

Number of entities that are pulled out of this block.

#### Dependencies

To enable this port, click the **Statistics** tab and select the box labeled **Number of entities extracted, ex**.

Data Types: double

## Parameters

**Capacity** — Specify the capacity of the server

1 (default) | scalar

Specify the number of entities the block can serve simultaneously.

#### Programmatic Use

**Block Parameter:** Capacity

**Type:** character vector

**Values:** '1' | scalar

**Default:** '1'

**Service time source** — Choose the source to specify the service time

Dialog (default) | Signal port | Attribute | MATLAB action

Determine the source that specifies the service time.

You can select:

- 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.

#### **Programmatic Use**

**Block Parameter:** ServiceTimeSource

**Type:** character vector

**Values:** 'Dialog' | 'Signal port' | 'Attribute' | 'MATLAB action'

**Default:** 'Dialog'

**Service time attribute name** — Specify service time source attribute name

ServiceTime (default) | character vector

Determine the name of the attribute that is used as the service time value.

#### **Dependencies**

This parameter is visible when **Service time source** is set to Attribute.

#### **Programmatic Use**

**Block Parameter:** ServiceTimeAttributeName

**Type:** character vector

**Values:** 'ServiceTime' | character vector

**Default:** 'ServiceTime'

**Service time value** — Specify the value of the service time

1 (default) | scalar

**Tunable:** Yes

#### **Dependencies**

This parameter is visible when **Service time source** is set to Dialog.

#### **Programmatic Use**

**Block Parameter:** ServiceTimeValue

**Type:** character vector

**Values:** '1.0' | scalar

**Default:** '1.0'

**Service time action** — Specify service time

*dt* = rand(1,1) (default) | MATLAB code

Use MATLAB code to specify service time. *dt* specifies the service time. You can manually specify *dt* or use **Insert pattern** button to specify service time from a repeating sequence or from a distribution. For an example, see “Count Simultaneous Departures from a Server”.

## Dependencies

This parameter is visible when **Service time source** is set to MATLAB action.

### Programmatic Use

**Block Parameter:** ServiceTimeAction

**Type:** character vector

**Values:** MATLAB code

**Default:** 'dt = rand(1,1);'

**Event action** — Specify the behavior of the entity in certain events  
Entry (default) | Service complete | Exit | Blocked | Preempt

Specify the behavior of the entity in certain events. Define the behavior in the **Event action** parameter. The **Entry** and the **Exit** actions are called just after the entity entry and just before entity exit. The **Service complete** action is called after the completion of the entity service. The **Blocked** action is called after an entity is blocked. The **Preempt** is called after the preemption.

---

**Note** If an event action changes an entity, related block behavior such as resorting a priority queue, and rescheduling of any events, will occur after the event action has fully finished and returned.

---

### Programmatic Use

**Block Parameter:** EntryAction, ServiceCompleteAction, ExitAction, BlockedAction, PreemptAction

**Type:** character vector

**Values:** MATLAB code

**Default:** ''

**Permit preemption based on attribute** — Enable preemption of entities  
off (default) | on

Select this check box if you want to allow preemption of entities. Preemption is the replacement of an entity in a server block by an entity that satisfies certain criteria. Selecting this check box enables these parameters:

- **Sorting attribute name** in the **Preemption** tab
- **Sorting direction** in the **Preemption** tab
- **Write residual time to attribute** in the **Preemption** tab
- **Number of entities preempted, p** in the **Statistics** tab

For an example, see “Task Preemption in a Multitasking Processor”.

### Programmatic Use

**Block Parameter:** PermitPreemptionBasedOnAttribute

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

**Sorting attribute name** — Specify the name of the attribute used to determine the priority  
entity | entitySys.priority | character vector

Specify the name of the attribute used to determine the priority.

**Dependencies**

This parameter is visible when the **Permit preemption based on attribute** box is selected.

**Programmatic Use**

**Block Parameter:** `SortingAttributeName`

**Type:** character vector

**Values:** 'entity' | 'entitySys.priority' | character vector

**Default:** 'entity'

**Sorting direction** — Choose the direction of sorting the entities

`Ascending` (default) | `Descending`

Specify if the entities are sorted in ascending or descending order.

- `ascending` — Sorting entities with smaller key values to have a higher priority
- `descending` — Sorting entities with greater key values to have a higher priority

**Dependencies**

This parameter is visible when the **Permit preemption based on attribute** box is selected.

**Programmatic Use**

**Block Parameter:** `SortingDirection`

**Type:** character vector

**Values:** 'Ascending' | 'Descending'

**Default:** 'Ascending'

**Write residual time to attribute** — Save the residual service time from a preempted entity to an attribute

`off` (default) | `on`

**Dependencies**

This parameter is visible when the **Permit preemption based on attribute** box is selected.

**Programmatic Use**

**Block Parameter:** `WriteResidualTimeToAttribute`

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

**Residual time attribute name** — Specify the name of the attribute to contain the residual service time of a preempted entity

`ResidualTime` (default) | character vector

**Dependencies**

This parameter is visible when the **Write residual time to attribute** box is selected.



**Programmatic Use****Block Parameter:** ResidualTimeAttributeName**Type:** character vector**Values:** 'ResidualTime' | character vector**Default:** 'ResidualTime'

**Number of entities departed, d** — Outputs the number of entities that have departed the block  
off (default) | on

Number of entities that have departed the block.

**Programmatic Use****Block Parameter:** NumberEntitiesDeparted**Type:** character vector**Values:** 'on' | 'off'**Default:** 'off'

**Number of entities in block, n** — Outputs the number of entities present in the block, that are yet to depart  
off (default) | on

Number of entities present in the block that are yet to depart.

**Programmatic Use****Block Parameter:** NumberEntitiesInBlock**Type:** character vector**Values:** 'on' | 'off'**Default:** 'off'

**Pending entity in block, pe** — Pending entities  
off (default) | on

Indicates whether an entity that is yet to depart is present in the block. The value is 1 for a pending entity, and 0 otherwise.

**Programmatic Use****Block Parameter:** PendingEntityPresentInBlock**Type:** character vector**Values:** 'on' | 'off'**Default:** 'off'

**Number of pending entities, np** — Number of Pending entities  
off (default) | on

Outputs the number of pending entities the block has served that are yet to depart.

**Programmatic Use****Block Parameter:** NumberEntitiesPending**Type:** character vector**Values:** 'on' | 'off'**Default:** 'off'

**Average wait, w** — Outputs the average wait time

off (default) | on

Sum of the wait times for entities departed the block divided by their total number. Wait time is the duration between the Entity Server block entry and exit of an entity. For more information, see “Interpret SimEvents Models Using Statistical Analysis”.

**Programmatic Use**

**Block Parameter:** AverageWait

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

**Utilization, util** — Outputs the average time the server is occupied

off (default) | on

Average time the server is occupied. The block calculates this time as the ratio of the total wait time for entities to the server capacity multiplied by the total simulation time.

Wait time is the duration between the Entity Server block entry and exit of an entity. Total wait time is the sum of the wait times for entities departed the block.

**Programmatic Use**

**Block Parameter:** Utilization

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

**Number of entities preempted, p** — Outputs the number of preempted entities

off (default) | on

Outputs the number of preempted entities. Preemption is the replacement of an entity in a server block by an entity that satisfies certain criteria.

**Dependencies**

This check box appears if the **Permit preemption based on attribute** check box is selected.

**Programmatic Use**

**Block Parameter:** NumberEntitiesPreempted

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

**Number of entities extracted, ex** — Number of entities extracted from this block

off (default) | on

Outputs the number of extracted entities which are pulled out from this block by the Entity Find block. If an entity is being served during the extraction, the service is terminated. If an attribute that defines the service time and it is modified by the Entity Find block, service time does not change. When an entity is extracted, **Number of entities departed, d**, **Number of entities in block, n**,

**Average wait, w**, and **Utilization, util** statistics are updated accordingly. For more information about finding and extracting entities, see “Find and Extract Entities in SimEvents Models”.

**Programmatic Use**

**Block Parameter:** NumEntitiesExtracted

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

## Version History

Introduced in R2016a

### See Also

Entity Generator | Entity Multicast | Entity Gate | Composite Entity Creator | Composite Entity Splitter | Entity Input Switch | Entity Output Switch | Entity Queue | Discrete Event Chart | Multicast Receive Queue | Entity Multicast | Entity Replicator | Entity Terminator | MATLAB Discrete Event System | Resource Acquirer | Resource Releaser | Resource Pool

### Topics

“Count Simultaneous Departures from a Server”

“Model Server Failure”

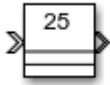
“Overview of Queues and Servers in Discrete-Event Simulation”

“Storage with Servers”

“SimEvents Common Design Patterns”

# Entity Store

Store entities



**Libraries:**  
SimEvents

## Description

The Entity Store block serves as a container or bin to store unordered entities. The entities are ready to leave the block immediately provided they are accepted by the next block. The Entity Store block attempts to forward an entity immediately upon its arrival. If the attempt fails, the block puts the entity in a pending state. The entity can then leave when the next block can start accepting it.

To customize actions when entities enter, exit, or are blocked, enter MATLAB code in the **Entry action**, **Exit action**, or **Blocked action** field of the **Event actions** tab.

## Ports

### Input

**Port\_1** — Input entity  
scalar | vector | matrix

Input entity port for entities entering the storage.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | fixed point

### Output

**Port\_1** — Output entity  
scalar | vector | matrix

Output entity port for entities exiting the storage.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | fixed point

**Port\_d** — Number of entities that have departed the block  
scalar

Number of entities that have departed the block.

### Dependencies

To enable this port, click the **Statistics** tab and select the box labeled **Number of entities departed, d**.

Data Types: double

**Port\_n** — Number of entities that have not yet departed the block  
scalar

Number of entities that have not yet departed the block.

#### Dependencies

To enable this port, click the **Statistics** tab and select the box labeled **Number of entities in block, n**.

Data Types: double

**Port\_w** — Average wait time for entities in the block  
scalar

Average wait time for entities in the block.

#### Dependencies

To enable this port, click the **Statistics** tab and select the box labeled **Average wait, w**.

Data Types: double

**Port\_l** — Average store size  
scalar

Average size of the store.

#### Dependencies

To enable this port, click the **Statistics** tab and select the box labeled **Average store size, l**.

Data Types: double

**Port\_ex** — Number of entities extracted  
scalar

Number of entities that are pulled out of this block.

#### Dependencies

To enable this port, click the **Statistics** tab and select the box labeled **Number of entities extracted, ex**.

Data Types: double

## Parameters

**Capacity** — Specify the capacity of the storage  
25 (default) | scalar

Specify the maximum number of entities contained in the storage.

#### Programmatic Use

**Block Parameter:** Capacity

**Type:** character vector

**Values:** '25' | scalar

**Default:** '25'

**Number of entities departed, d** — Outputs the number of entities that have departed the block  
off (default) | on

Selecting this check box outputs the number of entities that have exited the block.

**Programmatic Use**

**Block Parameter:** NumberEntitiesDeparted

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

**Number of entities in block, n** — Outputs the number of entities present in the block, which have yet to depart  
off (default) | on

Selecting this check box outputs the number of entities present in the block, which have yet to depart.

**Programmatic Use**

**Block Parameter:** NumberEntitiesInBlock

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

**Average wait, w** — Outputs the average wait time  
off (default) | on

Sum of the wait times for entities departing the block divided by their total number. Wait time is the duration between the Entity Store block entry and exit of an entity. For more information, see “Interpret SimEvents Models Using Statistical Analysis”.

**Programmatic Use**

**Block Parameter:** AverageWait

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

**Average store size, l** — Outputs the average store size  
off (default) | on

Accumulated time-weighted average store size. Store size is the number of entities stored in the block. The block computes average store size by:

- 1 Multiplying the store size by its duration to calculate time-weighted store size
- 2 Summing up all time-weighted store sizes and averaging them over total time

For more information, see “Interpret SimEvents Models Using Statistical Analysis”.

**Programmatic Use****Block Parameter:** AverageStoreSize**Type:** character vector**Values:** 'on' | 'off'**Default:** 'off'

**Number of entities extracted, ex** — Number of entities extracted from this block  
off (default) | on

Outputs the number of extracted entities which are pulled out from this block by the Entity Find block. When an entity is extracted, **Number of entities departed, d**, **Number of entities in block, n**, **Average wait, w**, and **Average store size, l** statistics are updated accordingly. For more information about finding and extracting entities, see “Find and Extract Entities in SimEvents Models”.

**Programmatic Use****Block Parameter:** NumEntitiesExtracted**Type:** character vector**Values:** 'on' | 'off'**Default:** 'off'

## Version History

**Introduced in R2018a**

### See Also

Entity Queue | Entity Multicast | Entity Server | Composite Entity Creator | Composite Entity Splitter | Entity Input Switch | Entity Output Switch | Discrete Event Chart | Entity Replicator | Entity Selector

### Topics

“SimEvents Common Design Patterns”

“Match Entities Based on Attributes”

# Entity Terminator

Terminate entities



**Libraries:**  
SimEvents

## Description

The Entity Terminator block accepts and destroys entities. Use this block to represent the entity departure from the model.

To customize actions when entities enter, use MATLAB code in the **Entry** action field of the **Event actions** tab. See “Events and Event Actions”, for more information.

## Ports

### Input

**Port\_1** — Input entity  
scalar | vector | matrix

Input entity port for entities entering the terminator.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | fixed point

### Output

**Port\_a** — Number of entities arrived  
scalar

Number of entities that have arrived at the block.

### Dependencies

To enable this port, click the **Statistics** tab and select the box labeled **Number of entities arrived, a**.

Data Types: double

## Parameters

**Event action** — Specify the behavior of the entity in certain events  
Entry

Specify the behavior of the entity in certain events. Define the behavior in the **Event action** parameter. The **Entry** action is called just after the entity entry to the block.

### Programmatic Use

**Block Parameter:** EntryAction



**Type:** character vector

**Values:** MATLAB code

**Default:** ''

**Number of entities arrived, a** — Outputs the number of entities that have arrived at the block  
off (default) | on

Number of entities that have arrived at the block.

**Programmatic Use**

**Block Parameter:** NumberEntitiesArrived

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

## Version History

Introduced in R2016a

### See Also

Entity Generator | Entity Queue | Entity Server

### Topics

“SimEvents Common Design Patterns”

# Hit Crossing

Detect crossing point



## Libraries:

Simulink / Discontinuities  
 Simulink / Messages & Events  
 HDL Coder / Discontinuities  
 SimEvents

## Description

The Hit Crossing block detects when the input reaches the **Hit crossing offset** parameter value in the direction specified by the **Hit crossing direction** property.

You can configure the block to output a 1 or 0 signal, a message, or a function-call event. See “Output” on page 2-82 for more information.

## Ports

### Input

**Port\_1** — Input signal

scalar | vector

Input signal that the block detects when it reaches the offset in the specified direction.

Data Types: double

### Output

**Port\_1** — Output signal

scalar | vector | message | function-call event

Output indicating if the input signal crossed the hit offset. This port is visible only when you select the **Show output port** parameter check box.

### Signal Output

If you select the **Show output port** check box and set the **Output type** parameter to Signal, the block output indicates when the crossing occurs.

- If the input signal is exactly the value of the offset value after the hit crossing is detected in the specified direction, the block continues to output a value of 1.
- If the input signals at two adjacent points brackets the offset value, the block outputs a value of 1 at the second time step.
- If the **Show output port** check box is *not* selected, the block ensures that the simulation finds the crossing point but does not generate output.
- If the initial signal is equal to the offset value, the block outputs 1 only if the **Hit crossing direction** property is set to either.

- If Boolean logic signals are enabled, then the output is a Boolean.

### Message Output

The Hit Crossing block can also output a message when the **Output type** parameter is set to Message.

- If the input signal crosses the offset value in the specified direction, the block outputs a message.
- If the input signal reaches the offset value in the specified direction and remains there, block outputs one message at the hit time and one message when the signal leaves the offset value.
- If the initial input signal is equal to the offset value, the block outputs a message with Crossing Type value None only if the **Hit crossing direction** is set to either.

The message output signal is a struct with four fields.

---

**Note** If the message output signal crosses model reference boundaries or is used as an input to a Stateflow chart, you need to create a bus object for the message. See “Tips”.

---

### Function-Call Output

The Hit Crossing block can also output a function-call event when the **Output type** parameter is set to Function-Call.

- Each time the input signal crosses the offset value in the specified direction, the block outputs a single function-call event.
- The function-call event can be sent to the function-call input port of a function-call subsystem or function-call model.
- The output is equivalent to the output of a Function-Call Generator block at each time step with the **Number of iterations** parameter of that block set to 1.

### CrossingType — Direction of zero-crossing

None | NegativeToPositive | NegativeToZero | ZeroToPositive | PositiveToNegative | PositiveToZero | ZeroToNegative

This field shows the direction in which the signal crosses the **Hit crossing offset** value. Negative, Zero, and Positive are defined relative to the offset value. The data type is `slHitCrossingType` which is an enumerated data type. See “Use Enumerated Data in Simulink Models” for more information. For example, if `HitCrossingOffset` is set to 2, a rising signal crossing this offset value would be recorded as a `NegativeToPositive` hit crossing.

---

**Note** A hit crossing is recorded based on the **Hit crossing direction** setting. In other words, if you set **Hit crossing direction** to detect a falling hit crossing, a `NegativeToPositive` hit is not recorded.

---



---

**Note** In a SimEvents block, if the Crossing Type of an entity is a `NegativeToPositive` hitcrossing then `entity.CrossingType == slHitCrossingType.NegativeToPositive` returns logical 1 (true).

---

If the signal reaches the `HitCrossingOffset` value and holds it, a single `NegativeToZero` or `PositiveToZero`, depending on the direction, hit is registered at the time of the hit crossing.

Data Types: `s1HitCrossingType`

**Index** — Index of the input signal at which the hit crossing event occurs  
nonnegative integer

For  $n$  signals being passed to the Hit Crossing block, this field denotes which signal had a hit crossing event. For a matrix input, this field follows MATLAB linear indexing. See “Array Indexing”.

Data Types: `uint32`

**Time** — Time of hit crossing event  
real, finite

Time  $T$  of the hit crossing event.

Data Types: `double`

**Offset** — Hit crossing value for detection  
0 (default) | real values

Hit crossing offset value as specified by the “Hit crossing offset” parameter.

Data Types: `double`

Data Types: `double` | `Boolean` | `struct`

## Parameters

**Hit crossing offset** — Hit crossing value for detection

0 (default) | real values

Specify the value the block detects when the input crosses in the direction specified by **Hit crossing direction**.

### Programmatic Use

**Block Parameter:** `HitCrossingOffset`

**Type:** character vector

**Values:** real values

**Default:** '0'

**Hit crossing direction** — Input signal direction to hit crossing

`either` (default) | `falling` | `rising`

Direction from which the input signal approaches the hit crossing offset for a crossing to be detected.

When set to `either`, the block serves as an *almost equal* block, useful in working around limitations in finite mathematics and computer precision. Used for these reasons, this block might be more convenient than adding logic to your model to detect this condition.

When the **Hit crossing direction** property is set to `either` and the model uses a fixed-step solver, the block has the following behavior. If the output signal is 1, the block sets the output signal to 0 at the next time step, unless the input signal equals the offset value.

**Programmatic Use****Block Parameter:** HitCrossingDirection**Type:** character vector**Values:** 'either' | 'rising' | 'falling'**Default:** 'either'**Show output port** — Display an output port

off (default) | on

If selected, create an output port on the block icon.

**Programmatic Use****Block Parameter:** ShowOutputPort**Type:** character vector**Values:** 'off' | 'on'**Default:** 'on'**Output type** — Choose signal, message, or function-call output

Signal (default for Simulink) | Message (default for SimEvents) | Function-Call

When **Output type** is set to Signal, the output signal is set to one whenever the input signal crosses the **Hit crossing offset** value in the **Hit crossing direction** and is zero at other times.When the **Output type** is set to Message, the output signal becomes a message.When **Output type** is set to Function-Call, the output signal becomes a function-call event.**Programmatic Use****Block Parameter:** HitCrossingOutputType**Type:** character vector**Values:** 'Signal' | 'Message' | 'Function-Call'**Default:** 'Signal'**Enable zero-crossing detection** — Enable zero-crossing detection

on (default) | off

Select to enable zero-crossing detection. For more information, see “Zero-Crossing Detection”.

**Programmatic Use****Parameter:** ZeroCross**Type:** character vector, string**Values:** 'on' | 'off'**Default:** 'on'**Block Characteristics**

<b>Data Types</b>	double
<b>Direct Feedthrough</b>	yes
<b>Multidimensional Signals</b>	no

<b>Variable-Size Signals</b>	no
<b>Zero-Crossing Detection</b>	yes

## Tips

If the Hit Crossing block is configured to output a message and the output signal:

- Crosses into or out of a referenced model
- Is fed to the input of a Stateflow chart

then you need to create a bus object for the message signal. In the MATLAB Command Window, run `Simulink.createHitCrossMessage` to check for and, if needed, create a hit crossing message bus object in the base workspace.

Set the data type of the corresponding port to Bus: `HitCrossMessage`.

## Version History

Introduced in R2018a

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

Not recommended for production code.

Does not support non-floating data type for ert targets.

### HDL Code Generation

Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

HDL Coder™ provides additional configuration options that affect HDL implementation and synthesized logic.

### HDL Architecture

This block has one default HDL architecture.

### HDL Block Properties

<b>ConstrainedOutputPipeline</b>	Number of registers to place at the outputs by moving existing delays within your design. Distributed pipelining does not redistribute these registers. The default is 0. For more details, see “ConstrainedOutputPipeline” (HDL Coder).
<b>InputPipeline</b>	Number of input pipeline stages to insert in the generated code. Distributed pipelining and constrained output pipelining can move these registers. The default is 0. For more details, see “InputPipeline” (HDL Coder).

<b>OutputPipeline</b>	Number of output pipeline stages to insert in the generated code. Distributed pipelining and constrained output pipelining can move these registers. The default is 0. For more details, see “OutputPipeline” (HDL Coder).
-----------------------	--

**Restriction**

The Hit crossing direction must be `rising` or `falling`.

HDL code generation is not supported when the `Output Type` is set to `Message`.

**See Also**

“Zero-Crossing Detection” | “Implement logic signals as Boolean data (vs. double)”

# MATLAB Discrete-Event System

MATLAB discrete-event system



**Libraries:**  
SimEvents

## Description

The MATLAB Discrete-Event System block allows you to create and author custom discrete-event systems. With this block, you can author an event-driven entity-flow system using MATLAB, and use it in your Simulink model. For more information about implementing `matlab.DiscreteEventSystem` class with MATLAB Discrete-Event System block, see “Create Custom Blocks Using MATLAB Discrete-Event System Block”.

Below, there are six examples to help you learn how to create custom blocks using the MATLAB Discrete-Event System block.

- 1 “Delay Entities with a Custom Entity Storage Block”
- 2 “Create a Custom Entity Storage Block with Iteration Event”
- 3 “Custom Entity Storage Block with Multiple Timer Events”
- 4 “Custom Entity Generator Block with Signal Input and Signal Output”
- 5 “Build a Custom Block with Multiple Storages”
- 6 “Create a Custom Resource Acquirer Block”

## Parameters

**System object name** — Specify the full name of the System object  
character vector

Specify the full name of the user-defined discrete-event System object class without the file extension. This entry is case sensitive. The class name must exist on the MATLAB path.

You can specify a discrete-event System object name in one of these ways:

- Enter the name in the text box.
- Click the list arrow attached to the text box. If valid System objects exist in the current folder, the names appear in the list. Select a System object from this list.
- Browse to a folder that contains a valid discrete-event System object. If the folder is not on your MATLAB path, the software prompts you to add it.

If you need to create a discrete-event System object, you can create one from a template by clicking **New**.

After you save the SimEvents System object, the name appears in the **System object name** text box.



Use the full name of the user-defined discrete-event System object class name. The block does not accept a MATLAB variable that you have assigned to a discrete-event System object class name.

**Programmatic Use**

**Block Parameter:** System

**Type:** character vector

**Values:** '<Enter System Class Name>' | character vector

**Default:** '<Enter System Class Name>'

**New** — Create a SimEvents System object from a template

SimEvents System object

Click this button to create a SimEvents System object from a template.

Select one of these options.

- Basic

Starts MATLAB Editor and displays a template for a simple discrete-event System object using the fewest System object methods.

After you save the SimEvents System object, you can enter the name in the **System object name** text box.

**Simulate using** — Specify the simulation mode

Code generation (default) | Interpreted execution

Specify the simulation mode as Code generation or Interpreted execution .

- Interpreted execution

This mode simulates the System object based on the interpreted MATLAB language with debuggers enabled.

- Code generation

This code generation mode reduces simulation time of SimEvents models. On the first model run, the MATLAB Discrete-Event System block simulates and generates code using only MATLAB functions supported for code generation. If the System object code and the block parameters do not change, subsequent model runs do not regenerate the code. MATLAB Discrete-Event System blocks also support code reuse for models that have multiple MATLAB Discrete-Event System blocks using the same System object source file. For more information, see Generate Code for MATLAB Discrete-Event System Blocks.

**Programmatic Use**

**Block Parameter:** SimulateUsing

**Type:** character vector

**Values:** 'Code generation' | Interpreted execution

**Default:** 'Code generation'

## Version History

Introduced in R2016a

## **See Also**

`matlab.DiscreteEventSystem` | `matlab.System` | Discrete Event Chart

## **Topics**

- “Create Custom Blocks Using MATLAB Discrete-Event System Block”
- “Call Simulink Function from a MATLAB Discrete-Event System Block”
- “Generate Code for MATLAB Discrete-Event System Blocks”
- “SimEvents Common Design Patterns”

# Receive, Message Receive

Receive messages



## Libraries:

Simulink / Messages & Events  
SimEvents

## Description

The Receive block extracts data from received messages and writes them to the output signal port. If there are no new messages when the block executes, the block uses the **Value source when queue is empty** value. Receive and Message Receive blocks are identical blocks.

- Select `Hold last value` to hold data read from the last message.
- Select `Use initial value` to write default data.

## Ports

### Input

**Port\_1** — Input message

scalar | vector | matrix

The input port for the message.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | fixed point

### Output

**Port\_S** — Show whether a message was received

scalar

Outputs 1 if the block receives a message successfully, and 0 otherwise.

### Dependencies

To enable this port, select the check box labeled **Show receive status**.

Data Types: double

**Port\_1** — Output signal

scalar | vector | matrix

Output port for the signal.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | fixed point

## Parameters

**Use internal queue** — Select to use an internal queue  
on (default for SimEvents) | off (default for Simulink)

Select this check box if you use an internal queue to receive messages.

### Programmatic Use

**Block Parameter:** UseInternalQueue

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'on'

**Overwrite the oldest element if queue is full** — Specify queue overwriting policy  
on (default for Simulink) | off (default for SimEvents)

Select this check box to choose between two queue overwriting policies.

- If you select the check box, an incoming message overwrites the oldest if the queue is full.
- If you clear the check box, the block does not accept new messages if the queue is full.

### Dependencies

This parameter is visible when you select the box labeled **Use internal queue**.

### Programmatic Use

**Block Parameter:** QueueOverwriting

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'on' (for Simulink) and 'off' (for SimEvents)

**Queue length** — Specify the length of the message queue  
16 (default) | scalar

Specify message queue capacity. The queue length can be specified as a positive integer between 1 and  $2^{26}-1$  (both included).

### Dependencies

This parameter is visible when you select the box labeled **Use internal queue**.

### Programmatic Use

**Block Parameter:** QueueLength

**Type:** character vector

**Values:** '16' | scalar

**Default:** '16'

**Queue type** — Specify message queue sorting policy  
FIFO (default) | LIFO | Priority

The block supports three message sorting policies:

- First-in-first-out (FIFO) — The oldest message in the storage departs first.
- Last-in-first-out (LIFO) — The newest message in the storage departs first.
- Priority — Messages or entities are sorted based on their priority.

The priority queue can be used only when the **Overwrite the oldest element if queue is full** check box is cleared.

---

**Note** Priority queue accepts only non-bus scalar and it does not support fixed point data type.

---

### Dependencies

This parameter is visible when you select the box labeled **Use internal queue**.

#### Programmatic Use

**Block Parameter:** QueueType

**Type:** character vector

**Values:** 'FIFO' | 'LIFO' | 'Priority'

**Default:** 'FIFO'

**Priority order** — Specify message queue priority

Ascending (default) | Descending

Choose the direction of sorting messages based on priority.

- Ascending — Messages with smaller priority values appear in front of the queue.
- Descending — Messages with greater priority values appear in front of the queue.

### Dependencies

This parameter is visible when you select the box labeled **Use internal queue** and **Queue type > Priority**.

#### Programmatic Use

**Block Parameter:** PriorityOrder

**Type:** character vector

**Values:** 'Ascending' | 'Descending'

**Default:** 'Ascending'

**Show receive status** — Show whether a message was received

off (default) | on

Select this check box to show whether a message was received. If this check box is selected the block outputs 1 if it receives a message successfully, and 0 otherwise.

#### Programmatic Use

**Block Parameter:** ShowQueueStatus

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

**Initial value** — Set initial data value

[] (unspecified) (default) | scalar | vector | matrix

Enter an initial data value for the queue before the arrival of the first message. The default value [ ] (unspecified) is treated as 0 with data type double.

To use this block to initialize a nonvirtual bus signal, specify the initial value as a MATLAB structure. For more information about initializing nonvirtual bus signals using structures, see “Specify Initial Conditions for Bus Elements”.

#### Programmatic Use

**Block Parameter:** InitialValue

**Type:** character vector

**Values:** ' [] ' | scalar

**Default:** ' [] '

**Value source when queue is empty** — Value source for empty queue

Hold last value (default) | Use initial value

Specify the value to receive when received message queue is empty.

- Hold last value (default) — Holds data read from the last message.  
Initially, if the block tries to receive a message and fails, it outputs the initial value.
- Use initial value — Writes default data.

#### Dependencies

This parameter is visible when you select the box labeled **Use internal queue**.

#### Programmatic Use

**Block Parameter:** ValueSourceWhenQueueIsEmpty

**Type:** character vector

**Values:** 'Hold last value' | 'Use initial value'

**Default:** 'Hold last value'

**Sample time (-1 for inherited)** — Specify the time interval between samples

-1 (default) | scalar

To inherit the sample time, set this parameter to -1. See “Specify Sample Time” for more information.

#### Programmatic Use

**Block Parameter:** SampleTime

**Type:** character vector

**Values:** ' -1 ' | scalar

**Default:** ' -1 '

## Block Characteristics

<b>Data Types</b>	Boolean   bus   double   enumerated   fixed point   integer   single   string
<b>Direct Feedthrough</b>	no

<b>Multidimensional Signals</b>	yes
<b>Variable-Size Signals</b>	no
<b>Zero-Crossing Detection</b>	no

## Version History

Introduced in R2016a

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

Message Send

### Topics

“Simulink Messages Overview”

“SimEvents Common Design Patterns”

## Send, Message Send

Create and send message



### Libraries:

Simulink / Messages & Events  
SimEvents

### Description

The Send block reads the value of an input signal, and sends a message that carries this value. In message-based communication, a message is a discrete-item of interest that carry data of any type that Simulink supports. Send and Message Send blocks are identical blocks.

### Ports

#### Input

**Port\_Enable** — External enable signal  
scalar

Input port to enable the block to send a message. For any input value that is greater than 0 send is enabled. For any value less than or equal to 0, the send is disabled.

#### Dependencies

To enable this port, select the box labeled **Show enable port**.

Data Types: double

**Port\_1** — Input signal  
scalar | vector | matrix

This block accepts inputs of any type that Simulink supports, including enumerated types and converts the input signal to a message. For more information, see “Data Types Supported by Simulink”.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | fixed point

#### Output

**Port\_1** — Output message  
scalar | vector | matrix

The block outputs a message with constant priority 20.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | fixed point



## Parameters

**Show enable port** — Display the enable port  
off (default) | on

Select this check box to display enable port. For any input value that is greater than 0 send is enabled. For any value less than or equal to 0, the send is disabled.

### Programmatic Use

**Block Parameter:** ShowEnablePort

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

## Block Characteristics

<b>Data Types</b>	Boolean   bus   double   enumerated   fixed point   integer   single   string
<b>Direct Feedthrough</b>	no
<b>Multidimensional Signals</b>	yes
<b>Variable-Size Signals</b>	no
<b>Zero-Crossing Detection</b>	no

## Version History

Introduced in R2016a

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

Message Receive

### Topics

“Simulink Messages Overview”

“SimEvents Common Design Patterns”

“Generate Entities When Events Occur”

“Enable a Gate for a Time Interval”

## Entity Transport Delay

Introduce a delay in propagation of a SimEvents message



**Libraries:**  
Simulink / Continuous  
SimEvents

### Description

The Entity Transport Delay block delays an entity for a period of time, named *transport delay*. The first input is the entity that is transported from point A to point B on a constant-length moving surface whose speed changes over time. The value from the second input is the *instantaneous delay*. The speed of the surface is the reciprocal of instantaneous delay. The product of instantaneous delay and the speed is 1.

The block calculates the implemented transport delay by the distance-speed-time relationship, where the surface length (distance) is equal to the integral of the variable surface speed over the duration of the transport delay (time). For more information about this calculation, see Variable Transport Delay.

The Entity Transport Delay block connects SimEvents to Simulink using the input from a Simulink signal and computing the transport delay as a continuous process, and applying this delay to an entity in a discrete-event process. For an example that uses the Entity Transport Delay block, see “Modeling Cyber-Physical Systems”.

### Ports

#### Input

**Port\_1** — Input entity  
SimEvents entity

SimEvents entity or message. For more information on entities in SimEvents, see “Entities in a SimEvents Model”.

**$t_i$**  — Instantaneous delay  
scalar | vector | matrix

Instantaneous delay in the transport of the SimEvents entity.  $t_i$  is the reciprocal of the speed of the entity. For more information on the calculation of instantaneous delay, see “Variable Transport Delay”.

Data Types: single | double | int8 | int16 | int32 | uint8 | uint16 | uint32 | bus

#### Output

**Port\_1** — Delayed entity  
SimEvents entity

SimEvents entity with the instantaneous delay  $t_i$  applied to it.

**n** — Number of delayed entities  
real scalar

Secondary output signal of the block, which displays the number of entities processed in a time step.

#### Dependencies

To enable this port, enable the **Output number of entities in block, n** parameter

Data Types: double

## Parameters

**Capacity** — Specify the capacity of the block

inf (default) | scalar

Specify capacity to accept entities to be delayed.

#### Programmatic Use

**Block Parameter:** Capacity

**Type:** character vector

**Values:** 'inf' | real scalar

**Default:** 'inf'

**Output number of entities in block, n** — Outputs the number of delayed entities present in the block

off (default) | on

Number of entities present in the block that are being delayed.

#### Programmatic Use

**Block Parameter:** ShowNumberEntitiesInBlock

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

## Block Characteristics

<b>Data Types</b>	Boolean   bus   double   enumerated   fixed point   integer   single
<b>Direct Feedthrough</b>	no
<b>Multidimensional Signals</b>	yes
<b>Variable-Size Signals</b>	no
<b>Zero-Crossing Detection</b>	yes

## Version History

Introduced in R2019b

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **See Also**

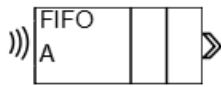
[Receive](#) | [Send](#) | [Transport Delay](#) | [Variable Transport Delay](#)

### **Topics**

“Establish Message Send and Receive Interfaces Between Software Components”

# Multicast Receive Queue

Receive multicast entities



**Libraries:**  
SimEvents

## Description

The Multicast Receive Queue block is identical to an Entity Queue block with the **Entity arrival source** parameter set to **Multicast**. For information about parameter descriptions, see the documentation for the Entity Queue on page 2-51 block.

You specify a **Multicast tag** for receiving entities. Then the block receives entities with a matching **Multicast tag** broadcast by the Entity Multicast block. See “Overview of Queues and Servers in Discrete-Event Simulation”, for more information about using multicast mode to broadcast entities.

## Ports

### Input

**receive** — Receive broadcasted entity  
scalar | vector | matrix

Input for received entities.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | string | fixed point

### Output

**Port\_1** — Output entity  
scalar | vector | matrix

Output entity at the head of the queue to depart when a downstream block is ready to accept it.

---

**Note** Event actions are not supported with string entity data type

---

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | string | fixed point

## Version History

Introduced in R2016a

### See Also

Entity Generator | Entity Server | Entity Multicast | Entity Store | Entity Selector

**Topics**

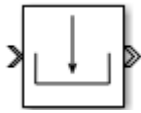
“Overview of Queues and Servers in Discrete-Event Simulation”

“Storage with Queues”

“SimEvents Common Design Patterns”

## Resource Acquirer

Acquire entity resources



**Libraries:**  
SimEvents

### Description

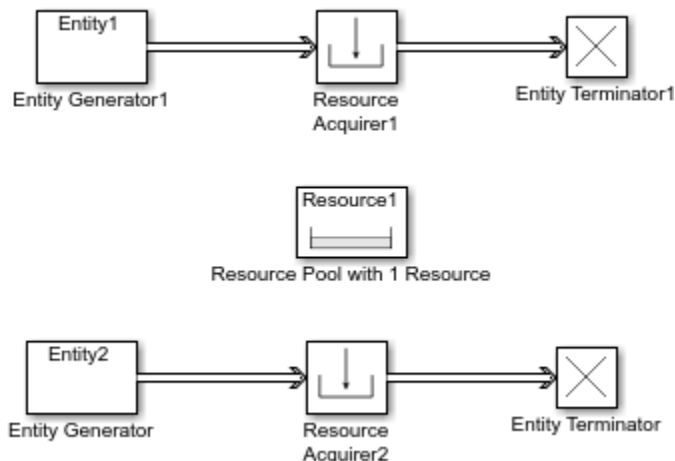
The Resource Acquirer block acquires for and assigns them to input entities. You can specify which resource amounts and type the block acquires.

An entity does not depart the Resource Acquirer block until the entity acquires all of the requested resources. For example, if an entity requests 5 resources and there are 2 resources available in a Resource Pool block, then the entity waits until all the requested resources are available before exiting. Similarly, if an entity requests 2 resources from one type and 3 resources from another type, the entity waits until all of the resources from both types are available.

Initialize a Resource Pool block with a specified amount of available resources. Then:

- Use one or more Resource Acquirer blocks to reserve those resources.

The priority order of Resource Acquirer blocks is determined at the beginning of a simulation and cannot be customized. The entity in the higher priority Resource Acquirer block always acquires the resource first.



For example, suppose only 1 resource is available in the Resource Pool block and Resource Acquirer1 is higher priority than Resource Acquirer2. If Entity1 and Entity2 want to acquire the resource at the same time, Entity1 always acquires the resource. Even if the resource becomes available again and there are two entities Entity1 and Entity2 waiting in the ResourceAcquirer1 and ResourceAcquirer2, Entity1 again acquires the resource.

- Use a Resource Releaser block to return resources to the Resource Pool block for future use.

The visibility of the resources is determined by the **Resource visibility** parameter of the Resource Pool blocks in the model.

To customize actions when entities enter, exit, or are blocked, enter MATLAB code in the Entry action, Exit action, or Blocked action field of the **Event actions** tab.

### Available Resources




Use the **Available Resources** controls to:

- Select the resources from the resources defined in all the Resource Pool blocks in the model.
- Add the resources to the **Selected Resources** table, where you can configure resource acquisition details.

The list displays all the available resources in the model. (If there are no resources, the **Available Attributes** list is empty.)

If the resource list is long, you can type the resource name in the text box to filter the list.

Use the buttons in the **Available Resources** section to help build the resources table. The buttons perform these actions.

Button	Action
	Refresh the <b>Available Resources</b> list. The list updates with any upstream model changes you make while the block dialog box is open.
	Add the selected resources to the <b>Selected Resources</b> table.
	Move the selected resource from the <b>Selected Resources</b> table to the <b>Available Resources</b> list.  <b>Note</b> If the selected resource is one you added manually, this button appears dimmed.

The message area below the available resources list displays additional messages about the resources, as they apply.

Message	Meaning
> Resource already selected	You have already added the resource to the <b>Selected Resources</b> table. You cannot add the resource to the table again.

### Selected Resources

Use the controls under **Selected Resources** to build and manage the list of resources to attach to the entity. Each resource appears as a row in a table.






Using these controls, you can:

- Add a resource manually.



- Modify a resource that you added to the table from the **Available Resources** list to attach to the entity.

The buttons under **Selected Resources** perform these actions:

Button	Action	Notes
	Add a template resource to the table.	Rename the resource and specify its properties.
	Add a copy of the selected resource to the table to use as the basis of a new resource.	Rename the copy. Two resources cannot have the same name.
	Remove the selected resource from the <b>Selected Resources</b> table.	When you delete a resource this way, no confirmation appears and you cannot undo the operation.
	Move up the selected resource in order in the <b>Selected Resources</b> table.	N/A
	Move the selected resource down in order in the <b>Selected Resources</b> table.	N/A

**Note** If you delete a row and apply the change, the deletion can affect signal output ports corresponding to other attributes. For example, if the block has a signal output port **A2** and you delete the attribute with a port marked **A1**, the block relabels **A2** as **A1**. Verify that any signal that connects to the relabeled port is still connected as you expect.

Property	Specify	Use
<b>Name</b>	The name of the resource. Each resource must have a unique name.	Double-click the existing name, and then type the new name.
<b>Amount Source</b>	Whether the resource amount, that an entity requests, comes from the dialog box or an attribute.	Select <b>Dialog</b> or <b>Attribute</b> . If you select <b>Attribute</b> , the source of the resource amount comes from the attribute of the entity. This option allows each entity to acquire varying amounts of resources. For more information, see "Set Resource Amount with Attributes"

Property	Specify	Use
<b>Amount</b>	The value to assign to the resource (when the resource comes from the dialog box).	<p>Double-click the value, and then type the value you want to assign.</p> <p>This value is the number of resources acquired per entity. For example, if <b>Amount</b> is 3, each entity that arrives at the Resource Acquire block must wait to acquire 3 resources before departing the block.</p> <p>Granularity of the resources to be acquired matches the granularity of the resources in the Resource Pool block.</p>

## Ports

### Input

#### **Port\_1** — Input entity

scalar | vector | matrix

Input entity port for entities entering the block.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | fixed point

### Output

#### **Port\_1** — Output entity

scalar | vector | matrix

Output entity port for entities exiting the block.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | fixed point

#### **Port\_d** — Number of entities that have departed the block

scalar

Number of entities that have departed the block.

### Dependencies

To enable this port, click the **Statistics** tab and select the box labeled **Number of entities departed, d**.

Data Types: double

#### **Port\_n** — Number of entities that have not yet departed the block

scalar

Number of entities that have not yet departed the block.

#### Dependencies

To enable this port, click the **Statistics** tab and select the box labeled **Number of entities in block, n**.

Data Types: double

**Port\_w** — Average wait time for entities in the block  
scalar

Average wait time for entities in the block.

#### Dependencies

To enable this port, click the **Statistics** tab and select the box labeled **Average wait, w**.

Data Types: double

**Port\_ex** — Number of entities extracted  
scalar

Number of entities that are pulled out of this block.

#### Dependencies

To enable this port, click the **Statistics** tab and select the box labeled **Number of entities extracted, ex**.

Data Types: double

## Parameters

**Maximum number of waiting entities** — Maximum number of entities that can wait for a resource  
1 (default) | scalar

Enter the maximum number of entities that can wait for a resource.

#### Programmatic Use

**Block Parameter:** NumberWaitingEntities

**Type:** character vector

**Values:** '1' | scalar

**Default:** '1'

**Event actions** — Specify the behavior of the entity on certain events

Entry (default) | Exit | Blocked

Define the behavior in the **Event actions**. For example, the **Entry** action is called when an entity enters the block.

---

**Note** If an event action changes an entity, related block behavior such as resorting a priority queue, and rescheduling of any events, will occur after the event action has fully finished and returned.

---

**Programmatic Use****Block Parameter:** EntryAction, ExitAction, BlockedAction**Type:** character vector**Values:** MATLAB code**Default:** ''

**Number of entities departed, d** — Outputs the number of entities that have departed the block  
off (default) | on

Selecting this check box outputs the number of entities that have exited the block.

**Programmatic Use****Block Parameter:** NumberEntitiesDeparted**Type:** character vector**Values:** 'on' | 'off'**Default:** 'off'

**Number of entities in block, n** — Outputs the number of entities present in the block, which have yet to depart  
off (default) | on

Selecting this check box outputs the number of entities present in the block, which have yet to depart.

**Programmatic Use****Block Parameter:** NumberEntitiesInBlock**Type:** character vector**Values:** 'on' | 'off'**Default:** 'off'

**Average wait, w** — Outputs the average wait time  
off (default) | on

Sum of the wait times for entities departing the block divided by their total number. Wait time is the duration between the entity's entry into and exit from the Resource Acquirer block. For more information, see "Interpret SimEvents Models Using Statistical Analysis".

**Programmatic Use****Block Parameter:** AverageWait**Type:** character vector**Values:** 'on' | 'off'**Default:** 'off'

**Number of entities extracted, ex** — Number of entities extracted from this block  
off (default) | on

Outputs the number of extracted entities which are pulled out from this block by the Entity Find block. When an entity is extracted, its resource acquisition from the Resource Pool block is canceled and **Number of entities departed, d**, **Number of entities in block, n**, and **Average wait, w** statistics are updated accordingly. For more information about finding and extracting entities, see "Find and Extract Entities in SimEvents Models".

**Programmatic Use**

**Block Parameter:** NumEntitiesExtracted

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

## **Version History**

**Introduced in R2016a**

### **See Also**

Entity Generator | Resource Releaser | Resource Pool

### **Topics**

“Model Using Resources”

“SimEvents Common Design Patterns”

## Resource Pool

Pool entity resources



### Libraries:

SimEvents / Entity Management  
SimEvents

## Description

The Resource Pool block defines resources that entities can use during model simulation. Use the Resource Acquirer and Resource Releaser blocks to work with these resources.

Initialize the block with the specified amount of available resources. Then:

- Use one or more Resource Acquirer blocks to reserve those resources.
- Use a Resource Releaser block to return resources back to this block for future use.

You can determine the visibility of available resources in a model hierarchy. You can choose `Global` or `Scoped` resources in the pool.

- `Global` — Resources can be referenced from anywhere in a model hierarchy.
- `Scoped` — Resources are locally visible and can be referenced only from the subsystem that contains the Resource Pool block and all the subsystems inside.

## Ports

### Input

**Port\_1** — Change resource amount

scalar | vector | matrix

Input entity port for changing resource amount. The input cannot be a negative value.

### Dependencies

To enable this port, select the `Change amount through control port` as the **Resource amount source**.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `fixed point`

### Output

**Port\_u** — Outputs the number of resources that are in use

scalar

Number of resources that are in use.

### Dependencies

To enable this port, click the **Statistics** tab and select the box labeled **Amount in use, u**.

Data Types: double

**Port\_util** — Outputs the average time the pool is utilized  
scalar

Average wait time the pool is utilized.

**Dependencies**

To enable this port, click the **Statistics** tab and select the box labeled **Average utilization, util**.

Data Types: double

**Port\_avail** — Outputs the number of resources that are available  
scalar

Number of resources that are available.

**Dependencies**

To enable this port, click the **Statistics** tab and select the box labeled **Amount available, avail**.

Data Types: double

## Parameters

**Resource name** — Name for the resources in the pool  
Resource1 (default) | character vector

Enter name of entity resource.

**Programmatic Use**

**Block Parameter:** ResourceName

**Type:** character vector

**Values:** 'Resource1' | character vector

**Default:** 'Resource1'

**Resource granularity** — Select granularity of resource use  
Discrete unit (default) | Fractional amount

Select granularity of resource use.

- Discrete unit — Use whole number increment.
- Fractional amount — Use fractional increment.

**Programmatic Use**

**Block Parameter:** ResourceGranularity

**Type:** character vector

**Values:** 'Discrete unit' | 'Fractional amount'

**Default:** 'Discrete unit'

**Reusable upon release** — Specify if the resource is reusable upon release  
off (default) | on

- Select this check box to allow this resource to return to the resource pool upon release. An example of such a resource is a table in a restaurant, which is available for reuse when a customer leaves.
- Clear this check box to prevent this resource from returning to the resource pool upon release. In this case, when the resource is released, it is no longer available in the resource pool. An example of such a resource is food in a restaurant, which is not reusable upon consumption.

**Programmatic Use****Block Parameter:** ReusableUponRelease**Type:** character vector**Values:** 'on' | 'off'**Default:** 'off'**Resource amount source** — Select resource amount source  
Dialog (default) | Change amount through control port

Select resource amount source.

- Dialog
- Change amount through control port

Select this option to enable an input entity port and a variable capacity resource. The payload of the arriving message increments the existing number of resources for the block. For example, if the resource pool has five resources, and a message with a payload of three arrives at the input port, the block has eight resources available. The number of resources cannot decrement.

**Programmatic Use****Block Parameter:** ResourceAmountSource**Type:** character vector**Values:** 'Dialog' | 'Change amount through control port'**Default:** 'Dialog'**Resource amount** — Set the amount of resource  
10 (default) | scalar

Enter amount of resource.

**Dependencies**Select the Dialog to enable the **Resource amount source**.**Programmatic Use****Block Parameter:** ResourceAmount**Type:** character vector**Values:** '10' | scalar**Default:** '10'**Initial resource amount** — Enter initial amount of resource  
10 (default) | scalar

Enter initial amount of resource.



## Dependencies

Select the `Change amount through control port` to enable the **Resource amount source**.

### Programmatic Use

**Block Parameter:** `InitialResourceAmount`

**Type:** character vector

**Values:** '10' | scalar

**Default:** '10'

**Resource visibility** — Select the availability of the resources

Global (default) | Scoped

Choose the behavior of the resources acquired from this pool as `Global` or `Scoped`.

When `Global` is selected:

- Resource pool names must be unique in the model.
- All resources have global scope and they can be referenced from anywhere in a model hierarchy.
- An entity carrying a resource acquired from this block, must explicitly relinquish the resource.
- When an entity is destroyed, the resources that are associated with it are returned to the pool, if the **Reusable upon release** check box is selected.

When `Scoped` is selected:

- Resources are locally visible and can be referenced only from the subsystem that contains the Resource Pool block and all the subsystems inside.
- Resource pool names must be unique within the model hierarchy where the Resource Pool block is visible.
- An entity carrying the resource acquired from this block, must explicitly relinquish the resource.
- When an entity leaves the scope, any resources that are local to that scope and not already released, are automatically released. If **Reusable upon release** check box is selected, they are returned to the pool.

### Programmatic Use

**Block Parameter:** `InitialResourceAmount`

**Type:** character vector

**Values:** '10' | scalar

**Default:** '10'

**Amount in use, u** — Number of resources that are in use

off (default) | on

Outputs the amount of resources that the block has acquired and has not yet released. For example, if the resource pool has 10 resources, and the entity acquires all of them, this port shows 10. When the block releases the resources, this port shows 0.

### Programmatic Use

**Block Parameter:** `AmountInUse`

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

**Average utilization, util** — Outputs the average resource utilization  
off (default) | on

Outputs the average resource utilization.

**Programmatic Use**

**Block Parameter:** AverageUtilization

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

**Amount available, avail** — Outputs the amount of resources available  
off (default) | on

Outputs the amount of resources available.

**Programmatic Use**

**Block Parameter:** AmountAvailable

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

## Version History

Introduced in R2016a

### See Also

Entity Generator | Resource Acquirer | Resource Releaser

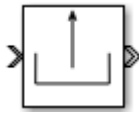
### Topics

“Model Using Resources”

“SimEvents Common Design Patterns”

# Resource Releaser

Release entity resources



**Libraries:**  
SimEvents

## Description

The Resource Releaser block releases resources when an entity enters the block. The block accepts one entity and the entity departs the block immediately provided it is accepted by the next block or extracted by the Entity Find block.

You can specify that the block release certain resource types or release all resources.

### Available Resources




Use the **Available Resources** controls to:

- Select the resources from the resources defined in all the Resource Pool blocks in the model.
- Add the resources to the Selected Resources table, where you can modify them.

The list displays all the resources in the model. (If there are no resources, the **Available Resources** list is empty).

If the resource list is long, you can type the resource name in the text box to filter the list.

Use the buttons in the **Available Resources** section to help build the resources table. The buttons perform these actions.

Button	Action
	Refresh the <b>Available Resources</b> list. The list updates with any upstream model changes you make while the block dialog box is open.
	Add the selected resources to the <b>Selected Resources</b> table.
	Move the selected resource from the <b>Selected Resources</b> table to the <b>Available Resources</b> list.  If the selected resource is one you added manually, this button appears dimmed.

The message area below the available resources list displays additional messages about the resources, as they apply.

Message	Meaning
> Resource already selected	You have already added the resource to the <b>Selected Resources</b> table. You cannot add the resource to the table again.






### Selected Resources

Use the controls under **Selected Resources** to build and manage the list of resources to release. Each resource appears as a row in a table.

Using these controls, you can:

- Add a resource manually.
- Modify a resource that you added to the table from the **Available Resources** list to release.
- Choose the amount of resources to be released by setting the **Amount Source** parameter to **Dialog** or **Attribute**.
  - **Dialog** — Specify the amount of resources to be released under the **Amount** column.
  - **Attribute** — Specify the name of the attribute that defines the amount of resources to be released.

The buttons under **Selected Resources** perform these actions.

Button	Action	Notes
	Add a template resource to the table.	Rename the resource and specify its properties.
	Add a copy of the selected resource to the table to use as the basis of a new resource.	Rename the copy. Two resources cannot have the same name.
	Remove the selected resource from the <b>Selected Resources</b> table.	When you delete a resource this way, no confirmation appears and you cannot undo the operation.
	Move up the selected resource in order in the <b>Selected Resources</b> table.	N/A
	Move the selected resource down in order in the <b>Selected Resources</b> table.	N/A

**Note** If you delete a row and apply the change, the deletion can affect signal output ports corresponding to other attributes. For example, if the block has a signal output port **A2** and you delete the attribute with a port marked **A1**, the block relabels **A2** as **A1**. Verify that any signal that connects to the relabeled port is still connected as you expect.

Property	Specification	Usage
<b>Name</b>	The name of the resource. Each resource must have a unique name.	Double-click the existing name, and then type the new name.
<b>Amount Source</b>	Whether the resource amount, that an entity requests, comes from the dialog box or an attribute.	Select <b>Dialog</b> or <b>Attribute</b> . If you select <b>Attribute</b> , the source of the resource amount comes from the attribute of the entity. This option allows each entity to acquire varying amounts of resources. For more information, see “Set Resource Amount with Attributes”.
<b>Amount</b>	The value to assign to the resource (when the resource comes from the dialog box).	<p>Double-click the value, and then type the value you want to assign.</p> <p>This value is the number of resources released per entity. For example, if <b>Amount</b> is three, each entity that arrives at the Resource Releaser block must wait to release 3 resources before departing the block.</p> <p>Granularity of the resources to be released matches the granularity of the resources in the Resource Pool block.</p>

## Ports

### Input

#### **Port\_1** — Input entity

scalar | vector | matrix

Input entity port for entities entering the block.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | fixed point

### Output

#### **Port\_1** — Output entity

scalar | vector | matrix

Output entity port for entities exiting the block.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | enumerated | bus | fixed point

**Port\_d** — Number of entities that have departed the block  
scalar

Number of entities that have departed the block.

**Dependencies**

To enable this port, click the **Statistics** tab and select the box labeled **Number of entities departed, d**.

Data Types: double

**Port\_n** — Number of entities that have not yet departed the block  
scalar

Number of entities that have not yet departed the block.

**Dependencies**

To enable this port, click the **Statistics** tab and select the box labeled **Number of entities in block, n**.

Data Types: double

**Port\_ex** — Number of entities extracted  
scalar

Number of entities that are pulled out of this block.

**Dependencies**

To enable this port, click the **Statistics** tab and select the box labeled **Number of entities extracted, ex**.

Data Types: double

**Parameters**

**Resource to release** — Select the resources to release  
All (default) | Selected

Select the resources to release.

- All

Release the use of all resources for a passing entity.

- Selected

Release selected resources. Selecting this option enables the **Available Resources** table.

**Programmatic Use**

**Block Parameter:** ResourceToRelease

**Type:** character vector

**Values:** 'All' | 'Selected'

**Default:** 'All'

**Event actions** — Specify the behavior of the entity on certain events

Entry (default) | Exit | Blocked

Define the behavior in the **Event actions**. For example, the **Entry** action is called when an entity enters the block.

**Programmatic Use**

**Block Parameter:** EntryAction, ExitAction, BlockedAction

**Type:** character vector

**Values:** MATLAB code

**Default:** ''

**Number of entities departed, d** — Outputs the number of entities that have departed the block

off (default) | on

Selecting this check box outputs the number of entities that have exited the block.

**Programmatic Use**

**Block Parameter:** NumberEntitiesDeparted

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

**Number of entities in block, n** — Outputs the number of entities present in the block, which have yet to depart

off (default) | on

Selecting this check box outputs the number of entities present in the block, which have yet to depart.

**Programmatic Use**

**Block Parameter:** NumberEntitiesInBlock

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

**Number of entities extracted, ex** — Number of entities extracted from this block

off (default) | on

Number of extracted entities that are pulled out from this block by the Entity Find block. For more information about finding and extracting entities, see “Find and Extract Entities in SimEvents Models”.

**Programmatic Use**

**Block Parameter:** NumEntitiesExtracted

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'off'

## **Version History**

**Introduced in R2016a**

### **See Also**

Entity Generator | Resource Acquirer | Resource Pool

### **Topics**

“Model Using Resources”

“SimEvents Common Design Patterns”



# Sequence Viewer

Display messages, events, states, transitions, and functions between blocks during simulation



## Libraries:

Simulink / Messages & Events  
 Simulink Test  
 SimEvents  
 Stateflow

## Description

The Sequence Viewer block displays messages, events, states, transitions, and functions between certain blocks during simulation. The blocks that you can display are called lifeline blocks and include:

- Subsystems
- Referenced models
- Blocks that contain messages, such as Stateflow charts
- Blocks that call functions or generate events, such as Function Caller, Function-Call Generator, and MATLAB Function blocks
- Blocks that contain functions, such as Function-Call Subsystem and Simulink Function blocks

To see states, transitions, and events for lifeline blocks in a referenced model, you must have a Sequence Viewer block in the referenced model. Without a Sequence Viewer block in the referenced model, you can see only messages and functions for lifeline blocks in the referenced model.

---

**Note** The Sequence Viewer block does not display function calls generated by MATLAB Function blocks and S-functions.

---

## Parameters

**Time Precision for Variable Step** — Digits for time increment precision

3 (default) | scalar

Number of digits for time increment precision. When using a variable step solver, change this parameter to adjust the time precision for the sequence viewer. By default the block supports 3 digits of precision.

Suppose the block displays two events that occur at times 0.1215 and 0.1219. Displaying these two events precisely requires 4 digits of precision. If the precision is 3, then the block displays two events at time 0.121.

### Programmatic Use

**Block Parameter:** VariableStepTimePrecision

**Type:** string scalar or character vector

**Values:** "3" | scalar

**Default:** "3"

**History** — Maximum number of previous events to display  
5000 (default) | scalar

Total number of events before the last event to display.

For example, if **History** is 5 and there are 10 events in your simulation, then the block displays 6 events, including the last event and the five events prior the last event. Earlier events are not displayed. The time ruler is greyed to indicate the time between the beginning of the simulation and the time of the first displayed event.

Each send, receive, drop, or function call event is counted as one event, even if they occur at the same simulation time.

#### Programmatic Use

**Block Parameter:** History

**Type:** string scalar or character vector

**Values:** "1000" | scalar

**Default:** "1000"

## Block Characteristics

<b>Data Types</b>	Boolean   bus   double   enumerated   fixed point   integer   single
<b>Direct Feedthrough</b>	no
<b>Multidimensional Signals</b>	yes
<b>Variable-Size Signals</b>	no
<b>Zero-Crossing Detection</b>	no

## Version History

Introduced in R2015b

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

This block can be used for visualizing message transitions during simulation, but is not included in the generated code.

### HDL Code Generation

Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

This block displays messages during simulation when used in subsystems that generate HDL code, but it is not included in the hardware implementation.

## **See Also**

### **Tools**

**Sequence Viewer**

### **Topics**

“Use the Sequence Viewer to Visualize Messages, Events, and Entities”

# SimEvents Debugger

Debug SimEvents models



**Libraries:**  
SimEvents

## Description

The SimEvents Debugger block enables the debugger for your SimEvents model. Using this block, you can:

- Inspect entities and their attribute values in storage blocks
- Set breakpoints on blocks and events
- Watch entities

---

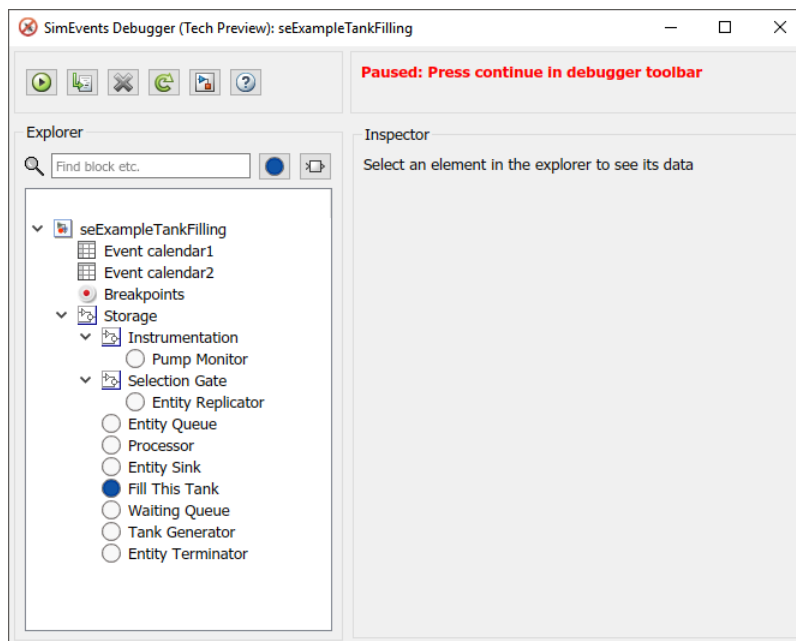
**Note** The SimEvents debugger is a preliminary version.


---

To start debugging your model:

- 1 From the SimEvents Library, add the SimEvents Debugger block into your SimEvents model.
- 2 In the Simulink Editor, click the **Step Forward** button.



The debugger interface appears.



- Click the **Continue simulation** button () to begin the simulation in the debugger. When the simulation completes, the debugger interface closes.
- To explore their data and behavior, the model tree displays in the left pane. Select the elements in the tree.
- When done, on the Simulink editor, click the **Stop** button to stop the simulation.

## Inspect Entities

To inspect entities in the debugger:

- To step to the next time step and inspect entities, in the Simulink editor, click the **Step Forward** button again. This action skips over all events at  $t_{now}$ .
- To step to the next event and inspect entities, in the debugger, click .
- To set a breakpoint:
  - At a particular time, use the Simulation Stepper to set breakpoints.
  - At an event on the event calendar, in the debugger, in the left pane, click an event calendar.
  - At every event, in the debugger, in the left pane, select the event calendar. In the Event Calendar Events pane, select the **Break before event execution** check box.
  - When an entity enters a block, in the debugger, select the block. At the bottom of the Inspector pane, select the **Break upon entry** check box.
  - When an entity leaves a block, in the debugger, select the block. At the bottom of the Inspector pane, select the **Break prior to entity exit** check box.
- To go to a breakpoint, in the debugger, click the **Continue** button ()

---

**Note** When you stop the debugger at a breakpoint, the Simulink editor and the MATLAB Command Window appear unresponsive. However, you can inspect entities, set new breakpoints, and continue the simulation from the debugger window.

---

- To watch entities, in the left tree, click the block. In the Inspector pane, select the check box of the entity you want to watch.

## Parameters

**Enable debugger** — Enable the debugger  
on (default) | off

Select this check box to enable the debugger for your model.

## Version History

Introduced in R2016a

## See Also

Entity Generator | Entity Queue

**Topics**

“Debug SimEvents Models”

“Debug Simulations in the Simulink Editor”

“Visualization and Animation for Debugging”

# Configuration Parameters

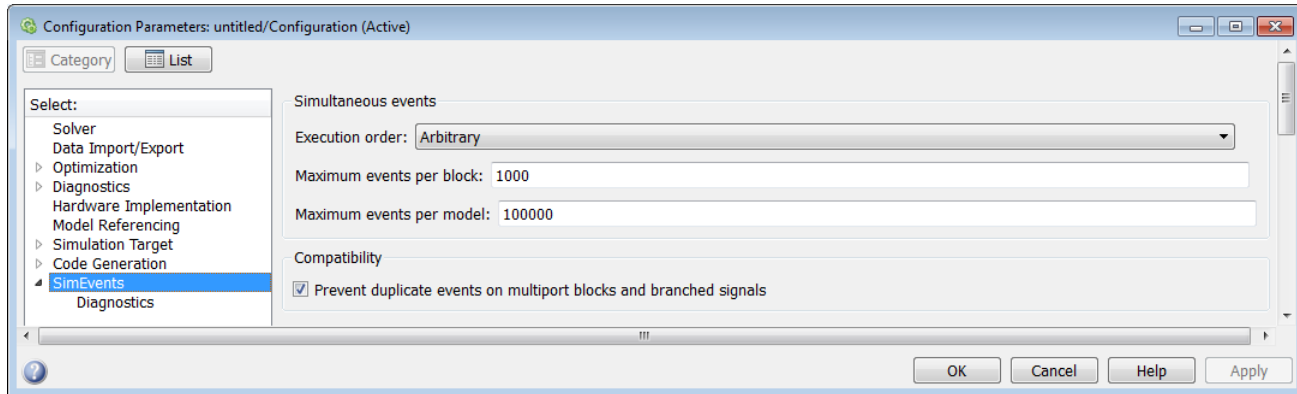
---

- “SimEvents Pane” on page 3-2
- “SimEvents Diagnostics Pane” on page 3-6

## SimEvents Pane

**Note** These configuration parameters are obsolete. They are available only for SimEvents releases prior to R2016a.

---



### In this section...

“SimEvents Pane Overview” on page 3-2

“Execution order” on page 3-2

“Seed for event randomization” on page 3-3

“Maximum events per block” on page 3-4

“Maximum events per model” on page 3-4

“Prevent duplicate events on multiport blocks and branched signals” on page 3-5

## SimEvents Pane Overview

**Note** These configuration parameters are obsolete. They are available only for SimEvents releases prior to R2016a.

---

Configure modelwide parameters related to discrete-event simulation and the logging of events and entities.

### Configuration

This pane appears only if your model contains a SimEvents block.

### Execution order

**Note** These configuration parameters are obsolete. They are available only for SimEvents releases prior to R2016a.

---



Select an algorithm for determining the sequence for processing simultaneous events having equal priorities.

### Settings

**Default:** Arbitrary

#### Arbitrary

Causes the simulation to use an internal algorithm to determine the sequence for processing simultaneous events having equal priorities.

#### Randomized

Causes the simulation to assign equal probability to all possible execution sequences of simultaneous events having equal numerical priorities.

### Tip

The processing sequence might be different from the sequence in which the events were scheduled on the event calendar.

### Dependency

Selecting Randomized enables **Seed for event randomization**.

### Command-Line Information

**Parameter:** propIdentEvents

**Type:** double

**Value:** 0 | 1

**Default:** 0

## Seed for event randomization

---

**Note** These configuration parameters are obsolete. They are available only for SimEvents releases prior to R2016a.

---

Initialize the random number generator for event processing.

### Settings

**Default:** 123456789

**Minimum:** 0

**Maximum:**  $2^{31} - 1$

This is a number that initializes the random number generator used to determine the sequence for processing simultaneous events having equal priorities.

### Tips

- For a given value of this parameter, the output of the random number generator is repeatable.

- To avoid unexpected correlations, make the value of this parameter distinct from all other seed parameters in the model (for example, the **Initial seed** parameter in the Event-Based Random Number block).

### Dependency

This parameter is enabled by **Execution order**.

### Command-Line Information

**Parameter:** propIdentEventSeed

**Type:** string

**Value:**

**Default:** '123456789'

## Maximum events per block

---

**Note** These configuration parameters are obsolete. They are available only for SimEvents releases prior to R2016a.

---

Limit the number of entity generation, service completion, subsystem execution, and function-call events that each SimEvents block performs at each fixed time instant.

### Settings

**Default:** 1000

**Minimum:** 2

**Maximum:**  $2^{31}-1$

### Command-Line Information

**Parameter:** propMaxDesBlkSimulEvents

**Type:** string

**Value:**

**Default:** '1000'

## Maximum events per model

---

**Note** These configuration parameters are obsolete. They are available only for SimEvents releases prior to R2016a.

---

Limit the total number of events scheduled via the event calendar at each fixed time instant. This is the maximum number of events per discrete-event system in a model.

### Settings

**Default:** 100000

**Minimum:** 2

**Maximum:**  $2^{31}-1$

**Command-Line Information**

**Parameter:** propMaxDesMdlSimulEvents

**Type:** string

**Value:**

**Default:** '100000'

## Prevent duplicate events on multiport blocks and branched signals

---

**Note** These configuration parameters are obsolete. They are available only for SimEvents releases prior to R2016a.

---

Prevent multifiring behavior on multiport blocks or branched signals that results in duplication of events. Multifiring behavior, an implicit result of the way that the software executes particular block configurations, occurs when the software executes a block more than once in response to a single, discrete event in the simulation.

### Settings

**Default:** On

On

Enable **Prevent duplicate events on multiport blocks and branched signals** parameter to prevent multifiring behavior.

Off

Allow multifiring behavior on multiport blocks or branched signals.

### Command-Line Information

**Parameter:** propPreventDuplicateEvents

**Type:** integer or boolean

**Value:**

**Default:** '1' for integer, 'True' for boolean

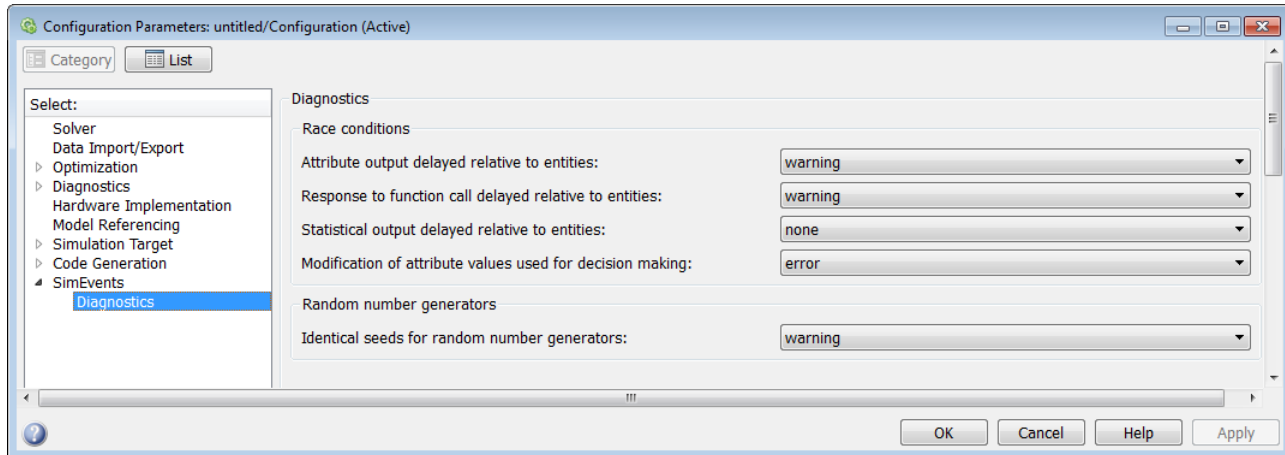
### See Also

### More About

- “SimEvents Diagnostics Pane” on page 3-6

## SimEvents Diagnostics Pane

**Note** These configuration parameters are obsolete. They are available only for SimEvents releases prior to R2016a.



### In this section...

“Diagnostics Pane Overview” on page 3-6

“Attribute output delayed relative to entities” on page 3-7

“Response to function call delayed relative to entities” on page 3-8

“Statistical output delayed relative to entities” on page 3-9

“Modification of attribute values used for decision making” on page 3-10

“Identical seeds for random number generators” on page 3-11

## Diagnostics Pane Overview

**Note** These configuration parameters are obsolete. They are available only for SimEvents releases prior to R2016a.

Specify what diagnostic action the application should take, if any, when it detects situations that might cause problems or unexpected results in the simulation.

### Configuration

This pane appears only if your model contains a SimEvents block.

### Tips

- The options are typically to do nothing or to display a warning or an error message.
- A warning does not terminate a simulation, but an error does.

## Attribute output delayed relative to entities

**Note** These configuration parameters are obsolete. They are available only for SimEvents releases prior to R2016a.

Select the diagnostic action to take if the application detects a situation in which a Get Attribute block updates a signal during entity advancement, but a subsequent block responds to the signal update after the entity has arrived. The application's processing sequence might cause the latter block to process the entity using outdated signal values.

### Settings

**Default:** error

none

The application does not check for this situation.

warning

When the application detects this situation, it displays a warning.

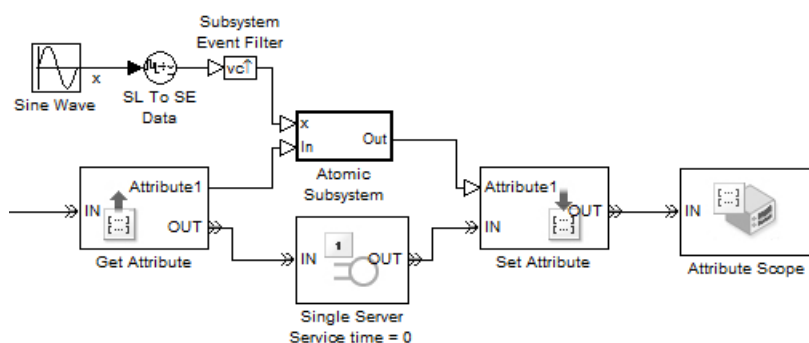
error

When the application detects this situation, it terminates the simulation and displays an error message.

### Tip

A Single Server block whose **Service time** parameter is 0 can address the problem by storing the entity while the latter block responds to the signal update.

### Example 3.1. Example of Solution



Alternatively, you might be able to address the problem by using an attribute directly instead of by using the signal output of a Get Attribute block.

### Command-Line Information

**Parameter:** propDiagAttribOutput

**Type:** double

**Value:** 0 | 1 | 2

**Default:** 2

### Recommended Settings

Application	Setting
Debugging	warning or error
Efficiency	none

### Response to function call delayed relative to entities

---

**Note** These configuration parameters are obsolete. They are available only for SimEvents releases prior to R2016a.

---

Select the diagnostic action to take if the application detects a situation in which a block issues a function call during entity advancement, but subsequent blocks respond to the function call and its consequences after the entity has arrived. The application's processing sequence might cause subsequent blocks to process the entity using outdated values of a signal whose update is a consequence of the function call.

#### Settings

**Default:** error

none

The application does not check for this situation.

warning

When the application detects this situation, it displays a warning.

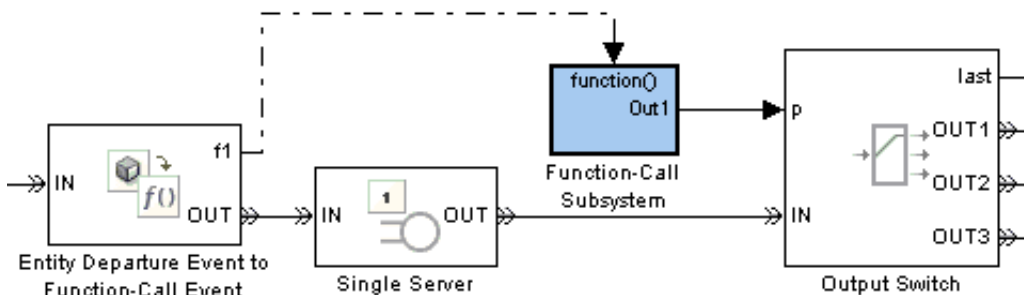
error

When the application detects this situation, it terminates the simulation and displays an error message.

#### Tip

A Single Server block whose **Service time** parameter is 0 can address the problem by storing the entity while subsequent blocks respond to the function call and its consequences.

#### Example 3.2. Example of Solution



**Command-Line Information****Parameter:** propDiagFcnCallOutput**Type:** double**Value:** 0 | 1 | 2**Default:** 2**Recommended Settings**

Application	Setting
Debugging	warning or error
Efficiency	none

**Statistical output delayed relative to entities**


---

**Note** These configuration parameters are obsolete. They are available only for SimEvents releases prior to R2016a.

---

Select the diagnostic action to take if the application detects a situation in which a block updates a statistical output signal during entity advancement, but a subsequent block responds to the signal update after the entity has arrived. The application's processing sequence might cause the latter block to process the entity using outdated signal values.

**Settings****Default:** error

none

The application does not check for this situation.

warning

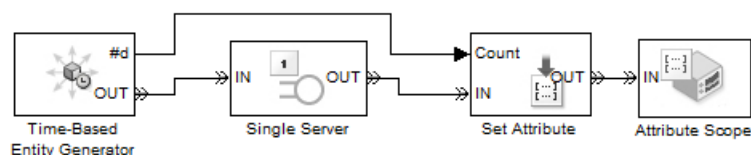
When the application detects this situation, it displays a warning.

error

When the application detects this situation, it terminates the simulation and displays an error message.

**Tip**

A Single Server block whose **Service time** parameter is 0 can address the problem by storing the entity while the latter block responds to the signal update.

**Example 3.3. Example of Solution**

#### Command-Line Information

**Parameter:** propDiagStatOutput

**Type:** double

**Value:** 0 | 1 | 2

**Default:** 1

#### Recommended Settings

Application	Setting
Debugging	warning or error
Efficiency	none

### Modification of attribute values used for decision making

---

**Note** These configuration parameters are obsolete. They are available only for SimEvents releases prior to R2016a.

---

Select the diagnostic action to take if the application detects certain situations in which a block modifies an attribute that a subsequent block uses to determine its availability. In some of these cases, internal queries among blocks might result in a decision based on information that changes when the entity actually advances.

#### Settings

**Default:** error

none

The application does not check for this situation.

warning

When the application detects this situation, it displays a warning.

error

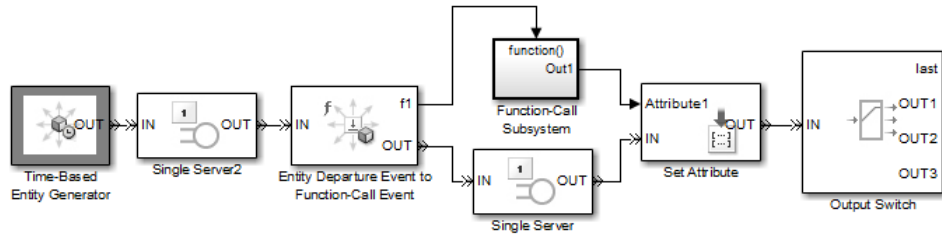
When the application detects this situation, it terminates the simulation and displays an error message.

#### Tip

A Single Server block whose **Service time** parameter is 0 can address the problem by storing the entity while the latter block responds to the signal update.



### Example 3.4. Example of Solution



#### Command-Line Information

**Parameter:** propDiagChangeAttrib

**Type:** double

**Value:** 0 | 1 | 2

**Default:** 2

#### Recommended Settings

Application	Setting
Debugging	warning or error
Efficiency	none

### Identical seeds for random number generators

**Note** These configuration parameters are obsolete. They are available only for SimEvents releases prior to R2016a.

Select the diagnostic action to take if the application detects that multiple random number generators use the same seed value, which might cause correlations among random processes.

#### Settings

**Default:** warning

none

The application does not check for this situation.

warning

When the application detects this situation, it displays a warning.

error

When the application detects this situation, it terminates the simulation and displays an error message.

### Tips

- If you set the parameter to `warning`, the warning message contains hyperlinks labeled “Randomize” and “Randomize All” that can help you address the problem.
- The `se_randomize_seeds` function provides a programmatic way to address the problem.
- Set the parameter to `none` if duplicate seeds are intentional in your model.

### Command-Line Information

**Parameter:** `propRNGIdenticalSeeds`

**Type:** `double`

**Value:** `0 | 1 | 2`

**Default:** `1`

### Recommended Settings

Application	Setting
Debugging	warning or error
Efficiency	none

### See Also

### More About

- “SimEvents Pane” on page 3-2

# Upgrade Advisor Checks

---

## SimEvents Upgrade Advisor Checks

---

**Note** These checks are obsolete. They are available only for SimEvents releases prior to R2016a.

---

### In this section...

“Checks Overview” on page 4-2

“Check for implicit event duplication caused by SimEvents blocks” on page 4-2

## Checks Overview

---

**Note** These checks are obsolete. They are available only for SimEvents releases prior to R2016a.

---

Use SimEvents Upgrade Advisor checks to identify backward-compatibility issues in your model.

### Check for implicit event duplication caused by SimEvents blocks

---

**Note** These checks are obsolete. They are available only for SimEvents releases prior to R2016a.

---

Check configuration parameters of model for status of **Prevent duplicate events on multiport blocks and branched signals** option.

#### Description

This Upgrade Advisor check verifies if you have selected the **Prevent duplicate events on multiport blocks and branched signals** check box in the Configuration Parameters dialog box of your model.

When you run a model created in a version of SimEvents prior to R2012a, the model might exhibit a behavior called multifiring that leads to duplication of events in the simulation. This event duplication behavior is implicit in models with certain configurations and results from the way the software executes the blocks of such configurations. Implicit event duplication is resolved in R2012a with the addition of the configuration parameter **Prevent duplicate events on multiport blocks and branched signals**.

Available with SimEvents.

#### Results and Recommended Actions

Condition	Recommended Action
SimEvents > <b>Prevent duplicate events on multiport blocks and branched signals</b> check box is not selected.	In the Configuration Parameters dialog box of your model, select the SimEvents > <b>Prevent duplicate events on multiport blocks and branched signals</b> check box.

An alternative to the recommended action in the preceding table is to use the **Modify Settings** button in the **Action** section of the Upgrade Advisor results pane. If you click **Modify Settings**, the software directly enables **Prevent duplicate events on multiport blocks and branched signals**.

---

**Note** The configuration parameter **Prevent duplicate events on multiport blocks and branched signals** is not compatible with blocks from versions of SimEvents prior to 4.0 (R2011b). The Upgrade Advisor provides the recommended action (if any) for the check, “Check for implicit event duplication caused by SimEvents blocks” on page 4-2.

---

## See Also

### More About

- “Consult the Upgrade Advisor”

