

# ANYLOGIC™

Enterprise Library Reference Guide



© 1992-2005 XJ Technologies Company Ltd.

[www.xjtek.com](http://www.xjtek.com)

Copyright © 1992-2005 XJ Technologies. All rights reserved.

**XJ Technologies Company Ltd**

AnyLogic@xjtek.com

<http://www.xjtek.com/products/anylogic>

# Contents

<b>1. GENERAL PRINCIPLES.....</b>	<b>5</b>
1.1 CONSTRUCTING A FLOWCHART FROM ACTIVE OBJECTS.....	5
1.2 ENTITY FLOW RULES.....	12
1.3 DYNAMIC PARAMETERS.....	19
<b>2. ENTITIES .....</b>	<b>21</b>
2.1 GENERIC CLASS ENTITY .....	21
2.2 SUBCLASSES OF ENTITY DEFINED BY THE USER.....	23
2.3 ANIMATING ENTITIES .....	24
<b>3. ACTIVE OBJECTS.....</b>	<b>27</b>
3.1 ENTITY FLOW.....	27
3.1.1 <i>Source</i> .....	27
3.1.2 <i>Sink</i> .....	30
3.1.3 <i>Enter</i> .....	31
3.1.4 <i>Exit</i> .....	32
3.1.5 <i>Hold</i> .....	34
3.1.6 <i>Split</i> .....	35
3.1.7 <i>Combine</i> .....	37
3.1.8 <i>Assembler</i> .....	39
3.1.9 <i>SelectInputPriority</i> .....	42
3.1.10 <i>SelectOutput</i> .....	43
3.1.11 <i>Queue</i> .....	45
3.1.12 <i>MatchQ</i> .....	49
3.1.13 <i>RestrictedAreaStart</i> .....	53
3.1.14 <i>RestrictedAreaEnd</i> .....	54
3.2 WORKING WITH ENTITY CONTENTS.....	55

3.2.1	<i>BatchQ</i> .....	55
3.2.2	<i>Unbatch</i> .....	58
3.2.3	<i>Pickup</i> .....	59
3.2.4	<i>Dropoff</i> .....	61
3.3	PROCESSING.....	63
3.3.1	<i>Schedule</i> .....	63
3.3.2	<i>Delay</i> .....	64
3.3.3	<i>Server</i> .....	67
3.4	WORKING WITH RESOURCES .....	69
3.4.1	<i>Resource</i> .....	69
3.4.2	<i>SeizeQ</i> .....	72
3.4.3	<i>Release</i> .....	75
3.4.4	<i>ProcessQ</i> .....	77
3.5	TRANSPORTATION.....	80
3.5.1	<i>Conveyor</i> .....	80
3.5.2	<i>Lane</i> .....	83
3.5.3	<i>Node</i> .....	85
3.5.4	<i>Segment</i> .....	90
3.6	ANIMATING ACTIVE OBJECTS .....	92
3.7	ADVANCED TRANSPORTATION.....	96
3.7.1	<i>Network and resources</i> .....	97
3.7.2	<i>Movement</i> .....	103
3.7.3	<i>Working with network resources</i> .....	109

# 1. General principles

AnyLogic™ Enterprise Library provides a higher-level interface for fast creation of discrete event models in the style of flowcharts. Flowcharts are a widely accepted graphical representation of systems of many important domains: manufacturing, logistics, workflow, service, business processes, computer networks, telecommunications, etc. Traditional flowchart frameworks, however, suffer from the lack of scalability, are hardly reusable, and are very inflexible when it comes to complex algorithms or large-scale real life models.

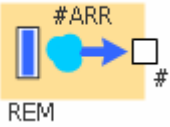
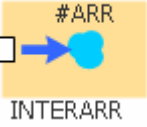
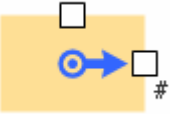
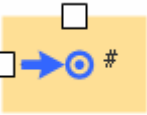

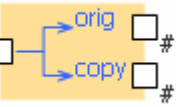
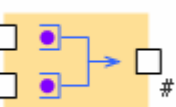
Based on a solid object-oriented framework and Java™, and armed with statecharts and hybrid discrete/continuous simulation engine, AnyLogic™ is designed to make complex problems amenable to simulation modeling. The Enterprise Library contains active objects traditional to flowcharts: sources and sinks, queues, delays, conveyors, etc. You can use this library to rapidly create models and animations in a drag-and-drop manner, just like in any other flowchart tool. At the same time, you are able to extend the standard functionality by inserting Java™ code, create your own classes of objects, use replication constructs, add state-based logic, add continuous behavior, etc. In other words, your model will always be scalable, flexible and extensible.

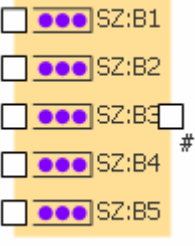

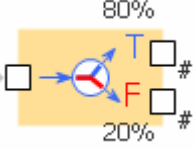
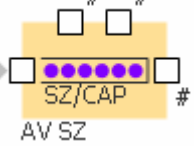
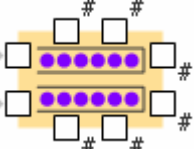


The way the Enterprise Library objects are implemented is open to the user. You may look into the implementation for better understanding of how they work or to construct your own custom object. XJ Technologies welcomes your suggestions about the contents, functionality, animation, and performance of the library. We're happy to do anything to make this important add-on even better.

## 1.1 Constructing a flowchart from active objects

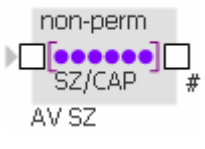
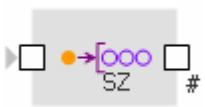

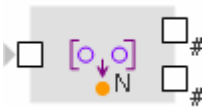
AnyLogic™ active object classes contained in the Enterprise Library are the building blocks you will use to construct your flowcharts. A message class Entity defined in the library will be your base class for entities, resource units and transporters. As usual, you have objects that generate entities, control entity flow, process entities, work with resources, and transport entities. In this reference, they are described in the six categories:

## Entity flow

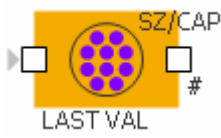
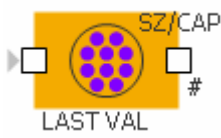
Icon	Name	Description
	Source	Generates entities.
	Sink	Disposes incoming entities.
	Enter	Inserts entities created elsewhere into the flowchart
	Exit	Accepts incoming entities.
	Hold	Blocks/unblocks the entity flow.
	Split	Creates the specified number of new entities (copies) of the entity.
	Combine	Waits for two entities, then produces a new entity from them.

	Assembler	Combines entities according specified bill.
	SelectInputPriority	Merges entities coming from two directions on a priority basis.
	SelectOutput	Forwards the entity to one of the output ports depending on the condition.
	Queue	Stores entities in the specified order.
	MatchQ	Finds a match between two entities from different inputs, then outputs them.
	RestrictedAreaStart	Limits number of entities in a part of flowchart between corresponding area start and area end blocks.
	RestrictedAreaEnd	Ends an area started with RestrictedAreaStart block.

## Working with entity contents

Icon	Name	Description
	BatchQ	Accumulates entities, then outputs them contained in a new entity.
	Unbatch	Extracts all entities contained in the incoming entity and outputs them.
	Pickup	Adds the selected entities to the contents of the incoming entity.
	Dropoff	Extracts the selected entities from the contents of the incoming entity.


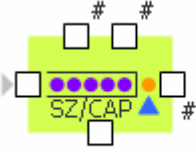
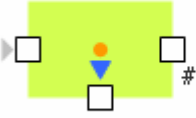
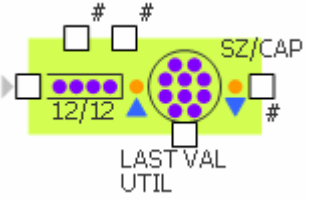
### Processing

Icon	Name	Description
	Delay	Delays entities by the specified delay time.
	Server	Delays entities until they receive the specified amount of service time.

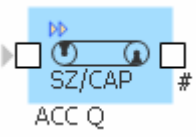
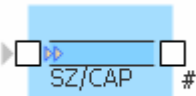
### Working with resources

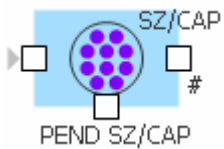
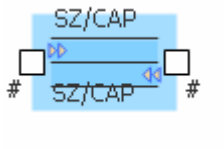
Icon	Name	Description
------	------	-------------




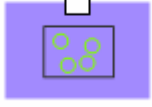



	Resource	Provides units that are seized and released by entities.
	SeizeQ	Seizes the number of units of the specified resource required by the entity.
	Release	Releases resource units previously seized by the entity.
	ProcessQ	Seizes resource units for the entity, delays it, and releases the seized units.

## Transportation

Icon	Name	Description
	Conveyor	Moves entities at a certain speed, preserving order and space between them.
	Lane	Moves entities with individual (and possibly different) speeds.

	Node	A source, destination and/or routing point of the transportation network.
	Segment	A network segment that provides bi-directional movement of transporters.

### Advanced transportation

Icon	Name	Description
	Network	Object that defines network.
	NetworkResource	Describes resources located in the network.
	NetworkEnter	Adds an entity to the specified location in the network.
	NetworkExit	Removes an entity from the network.
	NetworkMoveTo	Moves an entity from its current location to new location.

	NetworkMoveTo WithQ	Moves an entity to new location with an escort. The escort consists from resource units.
	NetworkCallQ	Calls a set of resource units of type staff to specified location.
	NetworkFree	Frees a set of previously seized units of type staff.
	NetworkFetchQ	Fetches portable resource unit by specified staff units.
	NetworkReturn	Returns portable resource unit by staff.
	NetworkSeize	Seizes resource unit of type static.
	NetworkRelease	Releases static resource.

## 1.2 Entity flow rules

When constructing a flowchart, it is important to understand how active objects exchange entities. Active objects of the Enterprise Library follow certain rules when they pass entities one to another. Entities enter and exit objects via ports. A port may work in only one direction: input or output (in this case, we do not consider passing resource units and transporters in the network). An input port may only be connected to an output port.

### 1.2.1.1 Entity passing protocol

When the entity is passed, the objects follow the specific protocol:

1. When an object intends to output an entity, it sends a notification to all connected objects.
2. If the object wants to receive an entity, it sends an entity request to all the ports that have entities. Actually, if an input port is connected to multiple output ports, it can accept an entity from any of them.
3. The entity is passed on the first request reception. If the request arrives by the time there is no entity at the sending object, `null` is passed.

Therefore the entity can never exit or enter an object without its prior agreement. This protocol is implemented on top of standard AnyLogic™ ports by defining two subclasses of the generic AnyLogic™ class `port`: `EntityInPort`, `EntityOutPort` and `EntityOutPortQueue`. There are two other protocols for the exchange of resource units between the `Resource` object and `SeizeQ` and `Release` objects, and for the exchange of transporters between `Node` and `Segment`, but they are not important for an Enterprise Library user, so we've left them out of this consideration.

The important property of the entity exchange protocol is that an object may sometimes be unable to output an entity because of inability of the other objects to accept it. There are only several objects that would allow an entity to stay and wait until it can be passed out: `Queue`, `Conveyor` and `Lane` (and objects that encapsulate them). All other objects will report an error if an entity spends a non-zero time waiting at the output. (This however does not mean the entities cannot be buffered there: it is OK for multiple entities to be passed to an output as long as they all exit in zero time.) Therefore, you should organize your entity

flow diagram in such a way that entities are always able to exit whenever they are not allowed to stay by adding a buffering object, or increasing a capacity of the existing object. A special case is Source object, which would optionally buffer the entities at its output—this is for simplicity of model creation. We recommend, however, to switch this option off later on, so you are always sure where your entities spend time.

As long as you connect outputs to inputs, arbitrary configurations are allowed within an AnyLogic™ flowchart: you can connect multiple output ports to a single input port, a single output port to multiple input ports, create active object classes and use relay ports to build hierarchical models, use replicated objects, etc.—see Figure 1. It is important to understand how M:1 and 1:M connections will behave in case there are several alternative possibilities. In AnyLogic™ if an input port can accept an entity from multiple output ports, it will do it in round-robin manner requesting the outputs (that have entity to give) one by one in a loop. If an output port is connected to multiple inputs and intends to give out an entity, it will broadcast the notification to everyone and then pass the entity on the first request. In case several objects were ready and sent their requests, the order in which they arrive is arbitrary, so the entity will be passed to a randomly chosen object.

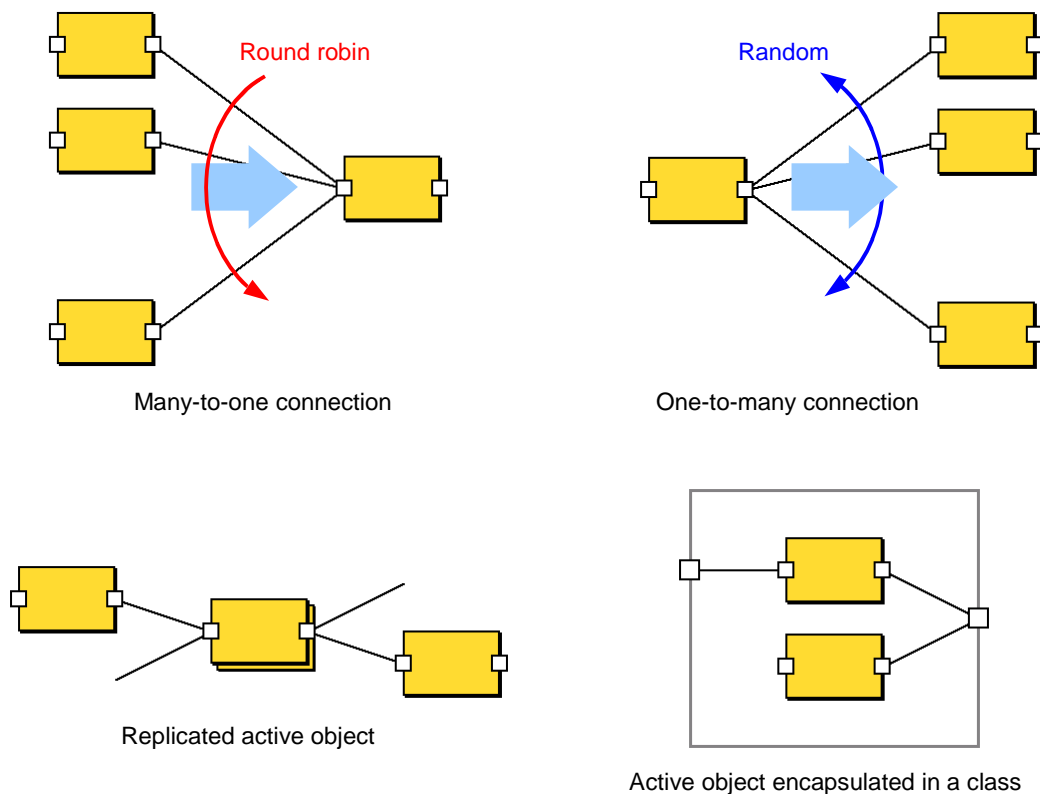


Figure 1 Entity flow and valid connections of active objects

The arbitrary connections allowed in AnyLogic™ Enterprise Library have an important consequence. It is important to understand how entities are passed in case there are several alternative possibilities. As long as an object may be accepting an entity from multiple sources, (which, in turn may be connected to multiple recipients), it can never know for sure which entity will come or even that an entity will come at all until it actually arrives. Symmetrically, an object can never know that another object will accept an entity until it actually requests it.

At some point during the model execution, you may wish to block the input of an object so that it stops accepting entities. Every Enterprise Library object has a function `block()` and `unblock()` for each input port. By naming those functions you may model working schedules of objects, change entity routing, or otherwise impose additional rules in the entity flow.

Let's examine how the entity passing protocol works in the model shown in Figure 2. Entities are passed via the output ports `outA`, `outB`, `outC1` and `outC2` to the `inK` and `inL` input ports of `objectK` and `objectL` correspondingly.

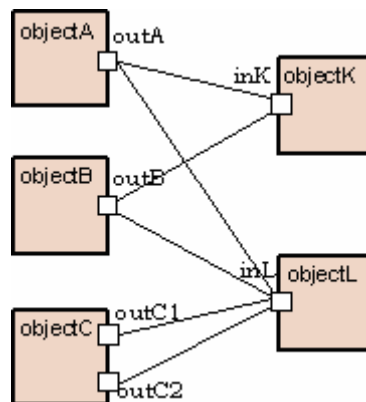


Figure 2.

Figure 3 shows the method call sequence. At the initial time entities are already pending at the `outA` and `outC1` ports.

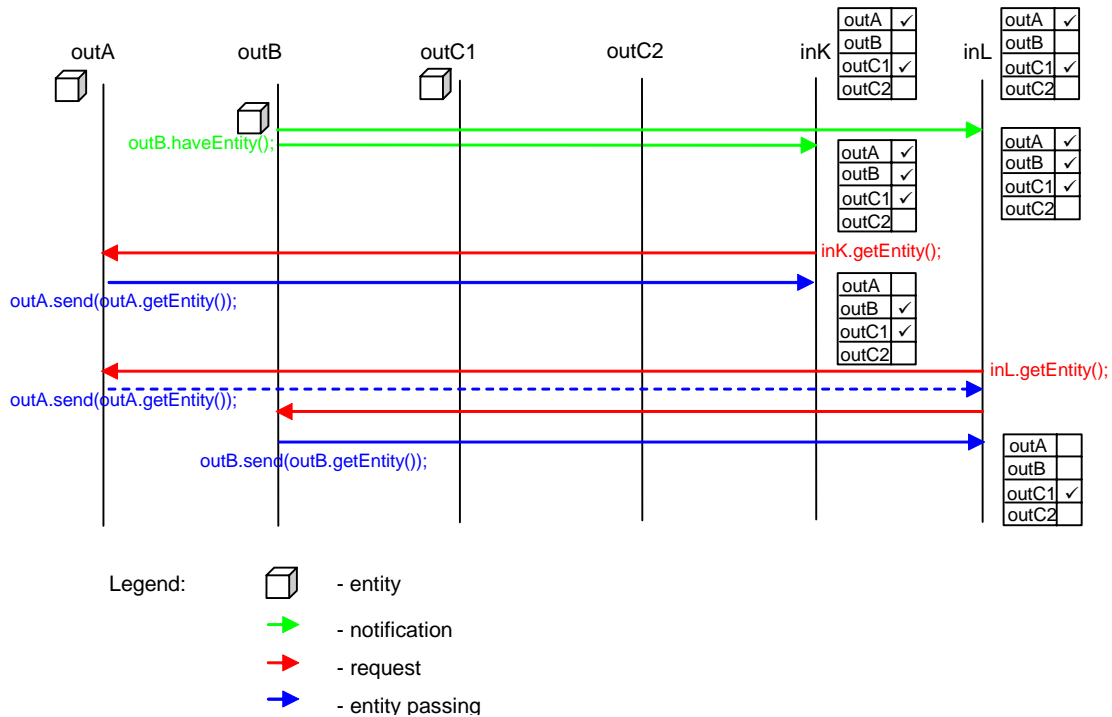


Figure 3. Entity passing protocol

The objectB passes entity at the outB port for output. The outB port sends the notification to all connected inputs (namely, inK and inL ports) by calling the `haveEntity()` method.

The notification is received at all the connected inputs. Each input port stores a table of the connected outputs. Ports ready to output entities are marked in this table with special flags. When a notification from the outB port is received at an input port, the corresponding flag in its table is set up.

If more than one entity is currently pending at the object, the `haveMoreEntities()` method is called. It schedules the successive notification sending at the same model time instant.

After some time objectK requests an entity by calling the `getEntity()` method of its inK port. All the connected outputs that have entities to output are requested in a round-robin manner. It is implemented by sending request messages to the marked outputs from the port table one by one in a loop.

First, the request is received at the outA port. The entity for output is returned by the `getEntity()` method of the outA port. Since the outA port has an entity to output, it is

passed to the requestor object. When the entity arrives at the `inK` port, the entity request procedure is finished and the entity requestor method `getEntity()` returns the received entity to the object.

Then the `inL` port requests an entity. First, the request is sent to the first marked port in the `inL` outputs table, namely to the `outA` port. But the request arrives by the time there is no entity at the `outA` port (the object has already output an entity to `objectK` on its request), so `null` is passed to the entity requestor. Since no entity was received from the `outA` port, the request is sent to the next marked port in the `inL` outputs table, namely to the `outB` port. The `outB` port has an entity to output, so it is passed to the `inL` port.

In case several objects were ready and sent their requests, the order in which they arrive is arbitrary, so the entity will be passed to a randomly chosen object.

In the case all the requested outputs already had no entities to output, the request initiator method `getEntity()` returns `null`.

### 1.2.1.2 Enterprise Library port classes

This section gives the detailed description of AnyLogic™ Enterprise Library port classes. It may be helpful for creating your own port classes with customized behavior. AnyLogic™ has two predefined port classes: `Port` for a port without a queue, and `PortQueuing` for a port with a queue. If you want to customize the default behavior of ports you need to define your own port class, derived from one of these base classes.

#### EntityInPort class

The input ports in the Enterprise Library are represented by the instances of the `EntityInPort` port class.

The methods of the `EntityInPort` class are listed in the table below.

Method	Description
--------	-------------



<code>void haveEntity()</code>	The method is called on notification arrival. You can override this method in the port instance of your object to specify custom actions to be performed – e.g., request an entity.
<code>Entity getEntity()</code>	The method requests an entity from connected outputs. The method returns the received entity. If no entities were received, <code>null</code> is returned.
<code>boolean hasEntity()</code>	The method checks if any connected output has an entity. If so, <code>true</code> is returned; otherwise <code>false</code> is returned.
<code>void processAuxiliary(EntityMsg fm )</code>	AnyLogic™ Enterprise Library enables you to send auxiliary messages between ports in both directions. You can process the received auxiliary message in a custom way by redefining this method in the port instance.
<code>ActiveObject getConnectedObject()</code>	The method returns first connected object, if any are known.

### EntityOutPort class

The output ports in Enterprise Library are represented by the instances of the `EntityOutPort` port class.

The methods of the `EntityOutPort` class are listed in the table below:

Method	Description
<code>void haveEntity()</code>	The method notifies connected input ports that this object has an entity to send via this port.
<code>void haveMoreEntities()</code>	The method schedules sending a notification to the connected input ports that this object has more entities to send via this port.

<code>abstract Entity getEntity()</code>	The method extracts the entity for output. The method is abstract and is overridden in the port instances of the Enterprise Library objects, since the entity obtaining logic may depend on the object work logic.
<code>int getCount()</code>	The method returns the number of entities exited through the port.
<code>int size()</code>	The method returns the number of pending entities.
<code>void resetStats()</code>	The method resets the collected statistics on exited entities.
<code>boolean isEmpty()</code>	The method checks if there are any entities pending in the port (the method returns <code>false</code> if there are any pending entities and <code>true</code> otherwise).
<code>void processAuxiliary( EntityMsg fm )</code>	AnyLogic™ Enterprise Library enables you to send auxiliary messages between ports in both directions. You can process the received auxiliary message in a custom way by redefining this method in the port instance.

### EntityOutPortQueue class

The `EntityOutPortQueue` port class, derived from `EntityOutPort`, is used to model output ports with queues. This port is commonly used in entity processing objects. If more than 1000 entities are pending, the port signals an error.

The methods of the `EntityOutPortQueue` class are listed in the table below:

Method	Description
<code>void take()</code>	The method places the entity passed as the method parameter in the port queue.
<code>Entity getEntity()</code>	The method extracts the first entity from the port queue.

<code>boolean isEmpty()</code>	The method checks if there are any entities pending in the queue (the method returns <code>false</code> if there are any pending entities and <code>true</code> otherwise).
<code>void setCanWait(boolean cw)</code>	The method allows or forbids the entity waiting at the output port queue by passing <code>true</code> or <code>false</code> correspondingly. (However, multiple entities can be passed to an output as long as they all are passed further in zero time).
<code>int size()</code>	The method returns the number of pending entities.

### 1.3 Dynamic parameters

When entities (including resource units and transporters) move within your flowchart, you often need to execute actions on them at particular moments, e.g., on entering or exiting an object, on picking up another entity, on releasing resource unit, on reaching the exit of a conveyor, etc. These actions may change some data within the entity, or say something to other objects, modify animation, and so on. Also, sometimes an entity carries information that is needed in the object to find out how to handle the entity. For example, you may wish the `Delay` object to delay an entity proportionally to a value of its custom field, or entities arriving at `Lane` object will be assigned different speeds depending on another custom information.

AnyLogic™ is an extremely flexible tool and allows you to do all that work by means of so-called “dynamic parameter evaluation.” There are two main types of active object parameters in AnyLogic™: simple and dynamic. Simple parameters work as constants within the object, whereas dynamic parameters invoke execution of a certain code each time they are accessed. Moreover, that code may be left undefined when the active object class is being developed, and defined later when the object is instantiated. That’s why such parameters are called dynamic.

Almost all objects of the Enterprise Library have parameters whose type starts with the word `code`. That means you can write a sequence of Java™ statements (in case of `code`) or Java™ expression of type `T` (in case of `code<T>`) in their value field. The code you write there will be executed each time the parameter is accessed within the object. Normally this happens on

particular events of the entity's lifetime within the object, and you need to know how to refer to the entity and maybe some other related variables. A list of such public variables of each object is given in the description of the library objects, and in most cases the current entity may be referred to simply as `entity`.

Please keep in mind that the code or expression you provide as a dynamic parameter value belongs to two objects at the same time: the encapsulated object whose parameter is being defined, and the object that encapsulates it (i.e. on whose structure diagram it is instantiated). Therefore, there may be conflicts when these two objects have variables or functions with identical names. The conflicts are resolved in the following way. Whenever you access a variable or call a function within a dynamic parameter, AnyLogic™ first interprets it as belonging to the encapsulated object, and then, if the name is not found, to the current object. For example, if your class `MyClass` encapsulates the object `queue` of type `Queue`, then the expression `size()` occurring, say, in the `onExit` code of the `queue`, would refer to the size of the `queue`. If, however, you have the same function `size()` in `MyClass` and wish to refer to that one in the dynamic parameter, you must write `MyClass.this.size()`.

## 2. Entities

### 2.1 Generic class Entity

Entity class is a fundamental class for all message objects that travel between the active objects of the Enterprise Library. An entity may represent an entity in its conventional meaning (product, order, customer, data packet), a resource unit (operator, machine, critical section), or a transporter (train, bus, ship, forklift truck).

Entities as “conventional” entities are generated at Source objects, then flow through the system being modeled, get processed, serviced, transported, compete for resources, accumulate cost, and then exit the system. Resource units are generated at Resource objects, are seized by entities, released and returned to the Resource object. Transporters are generated just like normal entities at Source objects, then are passed to Node objects and used to transport other entities between the nodes along segments. An object of class Entity may be used in any of these three roles, or even change the role if desired.

An entity may contain other entities and so on to any desired depth. The contained entities are stored in the field `contents` of type `Vector`. The resource units seized by the entity are stored in the `Vector resources`.

#### Variables

Type	Name	Default value	Description
int	priority	0	The entity priority used by the Queue objects, the larger the higher.
Vector	contents		The contents of the entity (other entities it carries)
Vector	resources		The resource units that the entity possesses.
Node	destination		Final destination of the entity being transported within the network.

ShapeRect	location		Location of the entity in a network*.
Vector	seizedStaff		Vector of the seized network resources of type staff*.
Vector	seizedPortable		Vector of seized network resources of type portable*.
Vector	seizedStatic		Vector of seized network resources of type static*.
Entity	lastResource		Reference to last seized or released static resource*.

Variables marked with \* should be used while working with ‘advanced transportation’ blocks.

### Functions

Return type	Name	Description
void	enableRotation( boolean en )	Enables or disables rotation of the entity animation.
ShapeBase	getAnimation()	Returns the animation of the entity.
void	setAnimation( ShapeBase sb )	Sets the animation of the entity to sb.
double	getX()	Returns the X position of the entity animation.
double	getY()	Returns the Y position of the entity animation.
void	setX( double x )	Sets the X position of the entity animation to x.
void	setY( double y )	Sets the Y position of the entity animation to y.
Color	getColor()	Returns the fill color of the entity default animation (the rectangle).
void	setColor( Color c )	Sets the fill color of the entity default animation (the rectangle).

void	<code>updateAnimation()</code>	Redraws the entity default animation and the animations of the contained entities in an arranged manner.
double	<code>getInOperatingTime()</code>	Time which includes any time that resource unit moves to (from) an entity, moves with entity and is being used by an entity at node*.
double	<code>getInIdleTime()</code>	Accumulated time of the resource unit being idle*.
int	<code>getCurrentState()</code>	Returns current state of the resource unit*.
int	<code>getResourceType()</code>	Returns type of the resource unit*.
double	<code>getUtilization()</code>	Returns utilization for the resource unit*.

Functions marker with \* return meaningful values only if the entity is used as a resource unit in the network (see Advanced transportation for details).

### Comments

Please remember that whenever you access a Vector element (contents or resources) by calling `get(i)`, the returned object is of class `Object` and needs to be casted to `Entity` or further down.

The X and Y positions of the entity animation are given with respect to its current context.

The function `updateAnimation()` is frequently overridden in subclasses. This saves you type downcasting when you refresh the animation.

## 2.2 Subclasses of Entity defined by the user

You frequently need to have custom data fields, methods or animation in entities. To achieve this, you need to declare a new message class that inherits from `Entity` and add the fields and methods there. Then when you generate your own class, you need to write new

`MyEntity()` instead of `new Entity()`. And whenever you access its custom functionality, you need to downcast it: `((MyEntity)entity).myField`.

It is an essential advantage of AnyLogic™ that you are able to use arbitrary Java™ classes as entities, resources or transporters: this gives you unlimited possibilities of making your models more close to real life and interoperable with other enterprise software.

## 2.3 Animating entities

A basic class `Entity` already provides simple entity animation. By default, an entity is animated as a small rectangle of randomly chosen color of green/blue spectrum range. The color can be changed by calling `setColor()`. This is useful when, e.g., you wish to mark the entities coming from a particular source with a particular color. You can also retrieve the color as `getColor()`, which may serve as a primitive identification of entities.

In case an entity contains other entities, they can be animated in an arranged manner. You, however, need to call the function `updateAnimation()` after you change the contents.

If you wish to associate a particular shape with an entity, you may simply draw this shape on the animation diagram at position (0,0) and call (e.g. right after entity creation):

```
entity.setAnimation( animation.shape );
```

You may even draw several shapes, then add them to a pivot group and use pivot as entity animation. Rotation of the entity animation may be allowed or prohibited by the function `enableRotation( boolean enable )`.

However, if you do it for several entities, then they will all have identical animations. If you then change, say, the color or other graphical parameters of the shape, it will change for all entities at once. This may be undesirable, as you may wish to distinguish entities graphically, e.g., set different shape size, rotation or color reflecting different stages of the entity lifetime. To make entities animations really independent, you need to create a different shape for each entity. Also, you will need to remember somehow the shape at each entity to be able to refer to it in the future.

The best solution in this case is to a) define your own entity class and b) use AnyLogic™ pivot with enabled “Custom shape template” as its animation. Follow these steps:



1. Draw the shape or shapes that will graphically represent the entities of the class on the animation diagram of, e.g., the Root object.
2. Draw a pivot there and add all these shapes to the pivot group.
3. Check the “Custom shape template” checkbox on the Graphics pane of the pivot properties. Give it the name `ShapeMy`. It is also good style to capitalize the first letter of the pivot name, as it will be now the name of a class.
4. Create a message class `EntityMy` in the project tree and make `Entity` its base class. This will be your new entity class.
5. In the Additional class code section of the Code pane of `EntityMy` write

```
Root._Group.ShapeMy shape = ((Root)Engine.getRoot()).animation.new ShapeMy();
```

6. In the Constructor code section write

```
shape.setup();
setAnimation( shape );
```

7. In the active object where you generate entities, write `new EntityMy()` instead of `new Entity()`.

This will give each entity of the class `EntityMy` individual animations. Now you can fully control the visual representation of these entities at runtime. Use the field `shape` of the entity to access the entity animation. E.g., you need to make an `oval` (one of the shapes you’ve added to the `ShapeMy` pivot group) invisible when the entity enters a `Delay` object. Then in the `onEnter` parameter of the `Delay`, you write:

```
((EntityMy)entity).shape.oval.setVisible( false );
```

You may further develop the class `EntityMy` to achieve more complex functionality with respect to animation. For example, you may add an image shape to `ShapeMy` and define a function `setPicture( int n )` that will change the active image. If you need to animate the entity contents, you may declare another field of `EntityMy` and a function:

```
Group shapeContents = new Group();

public void updateAnimation() {
    shapeContents.removeAll();
    for( int i=0; i<contents.size(); i++ ) {
        ShapeBase sb = ((Entity)contents.get(i)).getAnimation();
        shapeContents.add( sb );
    }
}
```

```
sb.setPosX( -i*6 );  
sb.setPosY( 0 );  
}  
}
```

Then each time you add or remove entities to/from the entity contents, you may call its function `updateAnimation()` to visually reflect the changes. Please look at the examples for further reference.

## 3. Active Objects

### 3.1 Entity flow

#### 3.1.1 Source

Generates entities. Is usually used as a starting point of the entity flow, or as a generator of resource units, transporters, etc. Can produce entities of arbitrary subclass of the generic Entity class with arbitrary interarrival times. Generation can be based either on distribution law or user can specify a table with timestamps. Maximum number of arrivals as well as the number of entities in each arrival can be defined. In case the generated entities are unable to exit, they are optionally buffered at the output port. When generation is based on distribution law the next interarrival time is calculated after each arrival; therefore it can be made, e.g., probabilistic, deterministic, dependent on external data, etc.



Figure 4 Source object

#### Variables

Type	Name	Description
Entity	entity	The current entity.

#### Functions

Return type	Name	Description
int	getArrivals()	Returns the number of the current arrival, starting with 0.

void	reset()	Causes re-evaluation of the current interarrival time.
------	---------	--

### Parameters

Type	Name	Default value	Description
code	onExit		Code executed when the entity exits the object.
Class	newEntity	Entity.class	Type of entity to generate.
	generationType	distribution	How that source will generate entities – based on distribution law or using arrival list ('arrivalList').
double	firstArrivalTime	0	The absolute time of the first entity arrival. Makes sense only if generation is distribution based. For 'arrivalList' first entity arrival is first timestamp in the list.
code<double>	interarrivalTime	exponential( 1)	Expression used to evaluate the delay time for each entity in distribution based mode.
code<int>	entitiesPerArrival	1	Expression used to evaluate the number of entities for each arrival. If 'arrivalList' is used then this parameter is invisible and number of entities for arrival is equal to corresponding value in lookup table.
LookupTable	arrivalList	null	If generationType is 'arrivalList' then lookup table with times and number of entities for each arrival should be specified here.

int	period	aperiodic	If 'arrivalList' is used then that parameter is period in model time units for arrival list. Parameter is not available for generation based on distribution law.
int	arrivalsMax	infinity	The maximum number of arrivals.
boolean	canWaitAtOutput	true	If true, entities may be buffered at the output when unable to exit the object (not more than 1000), otherwise runtime error will be signaled.

### Comments

To generate entities of a particular subclass `MyClass` of the generic class `Entity`, select `MyClass.class` in drop down list of the `newEntity` parameter.

To generate `n` entities at the beginning of the model execution, set `arrivalsMax` to 1 and `entitiesPerArrival` to `n`.

To be able to change the interarrival rate dynamically, you may define a parameter `rateMean` at the upper level object; set `generationType` to `distribution`, set `interarrivalTime` to, e.g., `exponential( rateMean )`; and associate `rateMean` with, a slider. Please note that when a slider occasionally sets `rateMean` to 0, the distribution sample will return a positive infinity value, and the Source object will never generate the next arrival. To bring the Source object back to live, call its `reset()` function – that will result in re-evaluation of the `interarrivalTime`. You may do it directly in the event handling code of the slider.

If you want to generate entities at particular moments of time `arrivalList` mode should be used. Create lookup table with timestamps as arguments and quantities of entities to generate at those timestamps as values. Set `generationType` to `arrivalList` and specify your lookup table as `arrivalList` parameter. To make table periodic specify `period` parameter `period`. Timestamps in the lookup table must be ascending ordered. If table contains repeatable values or unordered the behavior is undefined.

If you connect Source directly to, e.g., Delay or Conveyor object, there may occur situations when the entity generated is not able to exit Source immediately. The correct solution would be to place a Queue in between; but for the simplicity of the model, creation Source has a

parameter `canWaitAtOutput` that allows the entities wait some time at the output port of Source. We recommend to switch the parameter to false once you've made a first successful run.

### 3.1.2 Sink

Disposes incoming entities. Is usually used as an end point of the entity flow. Sink automatically counts incoming entities and calculates average rate of incoming flow.

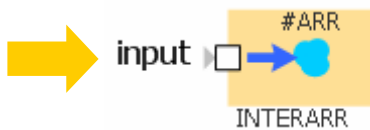


Figure 5 Sink object

#### Variables

Type	Name	Description
Entity	entity	The current entity.

#### Functions

Return type	Name	Description
void	<code>block()</code>	Blocks the input port of the object.
void	<code>unblock()</code>	Unblocks the input port of the object.
boolean	<code>blocked()</code>	Returns true if the input is blocked, otherwise false.
int	<code>getCount()</code>	Returns the number of entities passed through.
void	<code>reset()</code>	Resets counter of incoming entities and statistics for interarrival time to zero.
double	<code>getAvgInterarrivalTime()</code>	Returns average interarrival time.

double	getAverageRate()	Returns average rate of incoming flow.
--------	------------------	--

### Parameters

Type	Name	Default value	Description
code	onEnter		Code executed when the entity enters the object.

### Comments

A “hanging” output port in the AnyLogic™ Enterprise Library is unable to output entities, so wherever you wish to dispose entities, you must place either Sink or Exit objects.

## 3.1.3 Enter

Outputs entities created elsewhere and passed to this object “explicitly” either through `inputExternal` port or by calling its `take()` function. In combination with `Exit` can be used to route entities between arbitrary places in the model, not necessarily connected graphically. Also, can serve as an interface between the entity flow and other parts of the model. Once appeared at this object, the entity leaves it immediately.

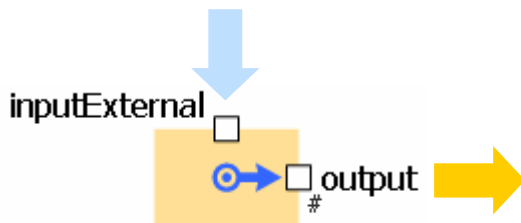


Figure 6 Enter object

### Variables

Type	Name	Description
Entity	entity	The current entity.

## Functions

Return type	Name	Description
void	take( Entity e )	Outputs the entity e via output port.

## Parameters

Type	Name	Default value	Description
code	onExit		Code executed when the entity exits the object.

## Comments

An Enter object can be used when:

- An entity generated outside the entity flow (in another part of the model described by other means) must be fed into it. In this case Enter serves as a kind of interface object between different modeling styles. If the entity appears at a port of external object, that port can be connected directly to inputExternal; otherwise, you may use the function take( ). Note that inputExternal is an ordinary port and it does not support the entity flow protocol.
- When a graphical connection between two points of the entity flow is impossible or undesirable; e.g., when entities are dynamically routed from one place to multiple destinations depending on some condition. Enter is then used together with Exit.
- When an entity extracted from the entity flow by means of API (e.g. by calling remove() function of the Queue) is fed into the flow.

### 3.1.4 Exit

Accepts incoming entities. Is usually used as an end point of the entity flow. In combination with Enter, can be used to route entities between arbitrary places in the model, not necessarily connected graphically. Also, upon reception of an entity, Exit outputs it “explicitly” via outputExternal port, thus providing an interface between the entity flow and other parts of the model, if needed. The operation takes zero time.



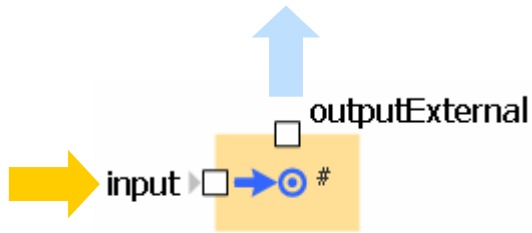


Figure 7 Exit object

### Variables

Type	Name	Description
Entity	entity	The current entity.

### Functions

Return type	Name	Description
void	block()	Blocks the input port of the object.
void	unblock()	Unblocks the input port of the object.
boolean	blocked()	Returns true if the input is blocked, otherwise false.
int	getCount()	Returns the number of entities passed through.
void	reset()	Resets counter of arrived entities and statistics for interarrival time to zero.
double	getAvgInterarrivalTime()	Returns average interarrival time.
double	getAvgRate()	Returns average rate of incoming flow.

### Parameters

Type	Name	Default value	Description
code	onEnter		Code executed when the entity enters the object.

## Comments

A “hanging” output port in the AnyLogic™ Enterprise Library is unable to output entities, so wherever you wish to dispose entities, you must place an Exit object.

Exit can not only dispose the entities, but also route them to an arbitrary location. For example, to route an entity to an Enter object named `myEnter`, write `myEnter.take(entity)` in the `onEnter` parameter of Exit.

An entity that exits the entity flow at the Exit object is always forwarded to the `outputExternal` port. If no one is connected to that port, the entity disappears. Otherwise, it is passed to the connected ports. Note that `outputExternal` is an ordinary port and it does not support the entity flow protocol.

### 3.1.5 Hold

Blocks/unblocks the entity flow along a particular connection. Unlike any other object in the library (except `SelectInputPriority`), Hold does not hold entities inside and can be treated as a modifier to the inter-object communication. Is particularly useful when you need to block entities coming to an input port from a particular object, or objects, whereas entities from other directions are allowed to enter.



Figure 8 Hold object

#### Variables

Type	Name	Description
Entity	entity	The current entity.

## Functions

Return type	Name	Description
void	block()	Blocks the input port of the object.
void	unblock()	Unblocks the input port of the object.
boolean	blocked()	Returns true if the input is blocked, otherwise false.

## Parameters

Type	Name	Default value	Description
code	onEnter		Code executed when an entity passes through the object.
boolean	initiallyBlocked	false	If true, the input port of the object is blocked right at startup.

## Comments

As long as you can in any way block/unblock any input port of any of the Enterprise Library objects, Hold only makes sense if you wish to implement only a partial blocking; e.g., to prohibit entities from one source and allow from another.

Hold is one of the few objects that do not store entities inside: it will only accept an entity if the receiving object is really requesting it.

### 3.1.6 Split

For each incoming entity, creates the specified number of new entities (copies) and outputs them via outputCopy port. The class of the new entities is specified by the user. The operation takes zero time; once entered Split, the entity and the copies leave it immediately.

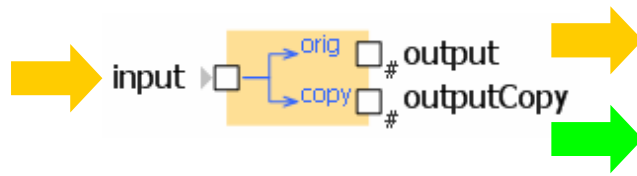


Figure 9 Split object

## Variables

Type	Name	Description
Entity	entity	The current original entity.
Entity	copy	The current copy of the original entity
int	copyNumber	The number of the current copy.

## Functions

Return type	Name	Description
void	block()	Blocks the input port of the object.
void	unblock()	Unblocks the input port of the object.
boolean	blocked()	Returns true if the input is blocked, otherwise false.

## Parameters

Type	Name	Default value	Description
code	onEnter		Code executed when the original entity enters the object.
code	onExit		Code executed when the original entity exits the object.
code	onExitCopy		Code executed for each copy when the copy exits the object.
Class	newCopy	Entity.class	Type of entity to generate as a copy.

code<int>	numberOfCopies	1	Expression evaluated to determine the number of copies of the current entity.
-----------	----------------	---	---

### Comments

The number of copies may vary and may depend on the original entity. Filling the new copy with contents is your responsibility. This may be done in the `onExitCopy` code. While the copies are made, the original entity is available as `entity` and the number of the current copy is available as `copyNumber`.

Split may be also used as an alternative to Dropoff: for example, if you write `copy = (Entity)entity.contents.remove(0)` in the `onExitCopy` parameter and `entity.contents.size()` in the `numberOfCopies`, the entity will drop off all its contents when passing through Split. In that case the `newCopy` parameter should be `Entity.class`.

## 3.1.7 Combine

Waits for two entities coming from `input1` and `input2` ports (in arbitrary order), then produces a new entity and outputs it. The class of the new entity is specified by the user. An entity that arrives first is accepted and waits for the other one inside this object. Once the other entity arrives, the resulting new entity leaves this object immediately.

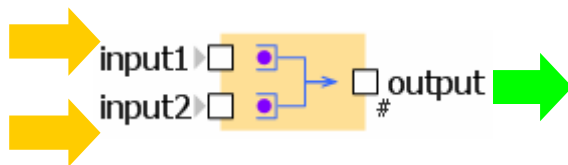


Figure 10 Combine object

### Variables

Type	Name	Description
Entity	entity1	The current entity arrived at input1 port
Entity	entity2	The current entity arrived at input2 port
Entity	entity	The current resulting entity that will exit

## Functions

Return type	Name	Description
void	block1()	Blocks the input1 port of the object.
void	unblock1()	Unblocks the input1 port of the object.
boolean	blocked1()	Returns true if the input1 is blocked, otherwise false.
void	block2()	Blocks the input2 port of the object.
void	unblock2()	Unblocks the input2 port of the object.
boolean	blocked2()	Returns true if the input2 is blocked, otherwise false.
boolean	hasFirstArrived()	Returns true if entity has arrived from input1 and waits for entity from input2.
boolean	hasSecondArrived()	Returns true if entity has arrived from input2 and wait for entity from input1.

## Parameters

Type	Name	Default value	Description
code	onEnter1		Code executed when the original entity enters the object via input1.
code	onEnter2		Code executed when the original entity enters the object via input2.
code	onCombine		Code executed when the two original entities and the resulting entity are all available.
code	onExit		Code executed when the resulting entity exits the object.
Class	newEntity	Entity.class	Type of resulting entity to create. If entity1 or entity2 is specified here then resulting entity will be entity1 or entity2 correspondingly.

ShapeBase	animationShape1		Animation template for the original entity entered via input1 and waiting for the other one to come. SINGLE type of animator is used.
ShapeBase	animationShape2		Animation template for the original entity entered via input2 and waiting for the other one to come. SINGLE type of animator is used.

### Comments

You may use this object for synchronization purposes only. For example, you wish to let entity1 out, only after entity2 arrives.

Also, Combine may be used as an alternative to Pickup: if you add to the above scheme `entity1.contents.add( entity2 )` written in `onCombine` parameter, then the first entity will wait until the other one arrives, pick it up, and exit. `newEntity` parameter in that case should be entity1.

Furthermore, if you loop the exiting entity back to the input1, you will be able to assemble a complex structure from arbitrary number of components. `onCombine` then will have a meaning of adding a part to the product. Of course, you have to insert `SelectOutput` into the loop to detect the completion.

Filling the resulting entity with contents is your responsibility; this may be done in the `onCombine` code. The difference between `onCombine` and `onExit` parameters is that when `onCombine` is called, `entity1` and `entity2` contain the two original entities. When `onExit` is called, they are both set to null.

## 3.1.8 Assembler

That block allows certain number of entities from several sources (5 or less) to be joined into a single entity. It can be used for instance to combine different parts of a job. The class of the new entity, as well as its initialization, is specified by the user. The number of entities for each input required to produce one new entity is also specified as object parameter. All arrived entities wait inside the object until all required entities arrive. Once the new entity can be build, it leaves this object immediately.

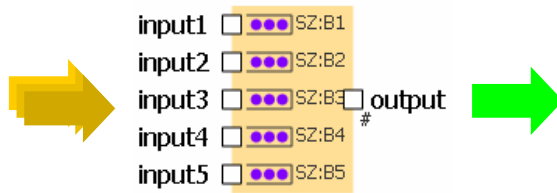


Figure 11 Assembler object

## Variables

Type	Name	Description
Entity	entity	The current resulting entity that will exit

## Functions

Return type	Name	Description
void	block()	Blocks all input ports of the object.
void	unblock()	Unblocks all input ports of the object.
boolean	blocked()	Returns true if all inputs are blocked, otherwise false.

## Parameters

Type	Name	Default value	Description
int[]	billOfMaterials	{1,1,1,1,1}	List of material quantities (number of entities) for each input required to produce one new entity.
Class	finalEntity	Entity.class	Type of resulting entity.
	queueCapacities	same as bill of materials	Incoming entities are stored in queues before the final entity can be assembled. By default each queue can store only quantity of entities specified in the bill. If 'custom' is selected then you can specify size for each queue.



integer	capacityQ1, capacityQ2, capacityQ3, capacityQ4, capacityQ5	100	If 'custom' queueCapacities selected then those parameters are capacities for queues of input 1,2,3,4 and 5.
	queueTypes	FIFO	Allows setting types for all queues to one type. If 'custom' specified as parameter value type for each queue separately can be specified.
	queueTypeQ1, queueTypeQ2, queueTypeQ3, queueTypeQ4, queueTypeQ5	FIFO	If 'custom' queueTypes selected then those parameters are types of a queues for input 1,2,3,4 and 5. The following types are possible: FIFO, LIFO, RANDOM and PRIORITY.
boolean	animateQueues	false	If false (default) queues are not animated.
	animationTypeQ1, animationTypeQ2, animationTypeQ3, animationTypeQ4, animationTypeQ5	AUTO	If animateQueues is true then those parameters are types of animators for queue 1,2,3,4 and 5.
ShapeBase	animationShapeQ1, animationShapeQ2, animationShapeQ3, animationShapeQ4, animationShapeQ5,		Animation template for the queues. The types depend on the animators.
code	onExit		Code executed when the resulting entity exits the object.

### Comments

Default bill of materials is {1, 1, 1, 1, 1}. That means that assembler will combine 5 entities from 5 inputs to one resulting entity. If queue capacities are 'same as bill of materials' then size of each queue on input 1,2 and etc. will be 1. Thus if there is one entity in first queue then input 1 will be able to accept entities until new entity will be assembled. If you want

to store more entities than specified in the bill of materials select ‘custom’ in queueCapacities parameter and specify size for each queue separately.

If zero is specified in the bill of materials for some port then previous objects connected to that port will be unable to pass entity to it.

You can specify not full bill of materials: i.e. {2, 3, 1}. It is equal to {2, 3, 1, 0, 0}. In that case input 4 and 5 should not be used.

### 3.1.9 SelectInputPriority

Merges the entities coming from two directions on a priority basis. The object connected to the output port is notified about entities coming from both input ports; however, when the entity is allowed to flow, the one on the inputPriority port passes first. This object only makes sense if there are cases when the object following it is sometimes temporarily unable to accept entities (say, a merge of two conveyors or lanes into one)—as otherwise, all entities will just flow through in the order they arrive to the inputs. SelectInputPriority does not hold entities inside and can be treated as a modifier to the inter-object communication.

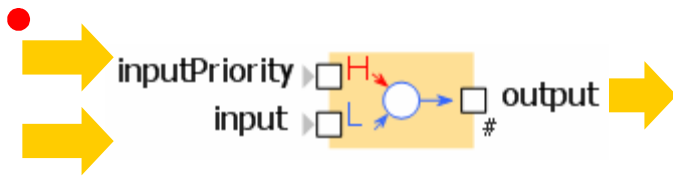


Figure 12 SelectInputPriority object

#### Variables

Type	Name	Description
Entity	entity	The current entity.

#### Functions

Return type	Name	Description
void	blockPriority()	Blocks the inputPriority port of the object.

void	unblockPriority ()	Unblocks the inputPriority port of the object.
boolean	blockedPriority ()	Returns true if the inputPriority is blocked, otherwise false.
void	block()	Blocks the input port of the object.
void	unblock()	Unblocks the input port of the object.
boolean	blocked()	Returns true if the input is blocked, otherwise false.

### Parameters

Type	Name	Default value	Description
code	onEnter		Code executed when an entity passes through the object.

### Comments

This is what can happen if the object after SelectInputPriority does not make pauses between accepting entities. For example: You merge two queues into one. The Queue after SelectInputPriority is empty and blocked, and the two queues before it are filled with entities. Then, when the Queue at the output gets unblocked, the entities (except the first one) will be taken with random probability from both inputs, even though you might expect them to be taken from the Queue at the inputPriority first. This happens because simultaneous events are executed in the random order. On the contrary, if the object at the output port would make a pause between the two requests (as, say, a Conveyor with non-zero spaces), then the priority Queue will exit first.

SelectInputPriority is one of the few objects that does not store entities inside: it will only accept an entity if the receiving object is really requesting it.

## 3.1.10 SelectOutput

Accepts an entity, then forwards it along one of the output ports depending on the user-specified condition. The condition may depend on the entity itself as well as on any other information. Once entered, the entity leaves this object immediately.

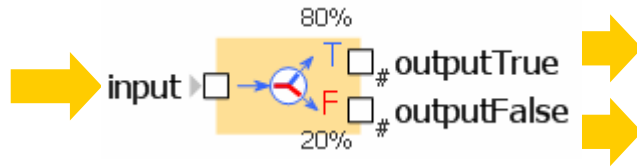


Figure 13 SelectOutput object

## Variables

Type	Name	Description
Entity	entity	The current entity.

## Functions

Return type	Name	Description
void	block()	Blocks the input port of the object.
void	unblock()	Unblocks the input port of the object.
boolean	blocked()	Returns true if the input is blocked, otherwise false.

## Parameters

Type	Name	Default value	Description
code	onEnter		Code executed when the entity enters the object.
code	onExitTrue		Code executed when the entity exits the object via outputTrue port.
code	onExitFalse		Code executed when the entity exits the object via outputFalse port.

code<boolean>	selectCondition	uniform() < 0.5	Condition evaluated for each entity. If evaluates to true, the entity exits via outputTrue port, otherwise via outputFalse. By default is set to uniform() < 0.5, i.e. the entities will exit via both ports with equal probabilities.
---------------	-----------------	-----------------	--

### Comments

The default value of selectCondition gives an example of probabilistic routing of entities. You can implement a deterministic routing as well. Suppose you wish to route entities of type `MyEntity` depending on the value of their integer field `size`. Then you may write in selectCondition: `((MyEntity)entity).size > 98`. The choice may also be made depending e.g. on the state of another object. For example, you wish to forward entities to outputTrue only if some object `queue` of type `Queue` is not full. Then you write: `queue.canEnter()`.

SelectOutput is obviously capable of sorting entities depending on their types. For example, you may specify entity `instanceof Cat` in the selectCondition, then cats will exit via outputTrue and all other creatures – via outputFalse.

## 3.1.11 Queue

Stores incoming entities in the specified order, which may be: FIFO (first in, first out), LIFO (last in, first out), RANDOM (entity is placed at a random position), or PRIORITY (the entity is placed in the queue according to its value of its priority field). An entity may leave the Queue in a number of ways:

- “normally,” via output port when the following object is ready to accept it
- via outputTimeout port after it has been waiting for the specified amount of time in the queue (timeout option should be on)
- via outputPreempted port being preempted by another incoming entity when the queue is full (preemption option is on)
- “manually” by calling the function `remove(int i)`

In the first case the entity leaves the Queue from the position 0; in all other cases the position may be arbitrary. Once directed to the outputTimeout or outputPreempted port, the entity must be able to leave immediately. When the preemption option is on, Queue is always ready to accept an entity, otherwise it would not accept if the capacity is reached. The timeout time is calculated individually for each entity.

Capacity of the queue can be set to 'Infinity'. To animate only subset of those entities entitiesToAnimate parameter should be used. That parameter defines how much entities from the beginning of queue will be animated. Animator must be able to show specified number of entities. I.e. you can use animator of type SET with polyline as shape and 'number of points in polyline' as value of entitiesToAnimate parameter.

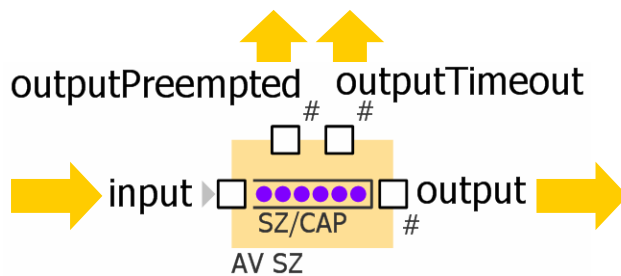


Figure 14 Queue object

### Variables

Type	Name	Description
Entity	entity	The current entity.
int	position	The position at which the current arriving entity is inserted in the queue.
double	timeoutValue	The timeout value set for the current arriving entity.

### Functions

Return type	Name	Description
void	block()	Blocks the input port of the object.
void	unblock()	Unblocks the input port of the object.
boolean	blocked()	Returns true if the input is blocked, otherwise false.

int	size()	The number of entities in the queue.
Entity	get( int i )	Returns the entity at i-th position (0 is at the exit).
Entity	remove( int i )	Removes the entity at position i and returns it.
boolean	canEnter()	Returns true if a new entity may be accepted by the queue. If preemption is on, this is always the case; otherwise this is equivalent to size() < capacity.
TimedDataSet	getStatsSize()	Returns the statistics on the queue size.
void	resetStats()	Resets the statistics collected at this object.

### Parameters

Type	Name	Default value	Description
code	onEnter		Code executed when the entity enters the object.
code	onExitPreempted		Code executed when the entity exits via outputPreempted port as a result of preemption by another entity.
code	onExitTimeout		Code executed when the entity exits via outputTimeout port as a result of timeout event.
code	onAtExit		Code executed when the entity comes to the position 0 of the queue and becomes ready to exit.
code	onExit		Code executed when the entity leaves the queue either via output port or by calling remove() function.

int	queueType	FIFO	The type of the queue. One of the following: FIFO LIFO RANDOM PRIORITY
integer	capacity	100	The capacity of the queue.
integer	entitiesToAnimate	all	How much entities should be animated.
boolean	preemption	false	If true, preemption is on.
boolean	timeout	false	If true, timeout option is on.
code<double>	timeoutTime	infinity	Expression evaluated to obtain the timeout time for the entity. Applies only when timeout is true.
boolean	statsEnabled	false	If true, statistics are collected for this object, otherwise not.
ShapeBase	animationShape		Animation template for the queue. The type depends on the animator.
int	animationType	AUTO	The animator type for the queue. Valid types are: AUTO SINGLE SET BAG ARRANGED QUEUE
boolean	animationForward	true	Orientation of polyline-based animations



## Comments

When `onEnter` is executed, the `position` and `timeoutValue` are known, and the entity is already placed in the queue. When `onExit`, `onExitPreempted` or `onExitTimeout` are executed, the entity is already removed from the queue; in the latter case, its latest position is available as `position`.

Preemption works this way. An incoming entity is always accepted and placed in the queue according to the queue type. If the queue capacity becomes exceeded, the last entity in the queue (it may well be the same that has just come) is removed and passed out via `outputPreempted` port.

When `timeout` is on, a timer is launched for each entity that joins the queue. The timeout time is evaluated individually for each entity as defined by `timeoutTime`; if 0 is returned, timer is not created. If the timeout expires before the entity exits the queue, it is removed and passed out via `outputTimeout` port.

Queue of type `RANDOM` places an incoming entity at a random position. The priority field of the entity only matters when the type is `PRIORITY`. Higher value means higher priority.

Capacity of the queue may be changed dynamically.

### 3.1.12 MatchQ

Stores the entities coming from the two input ports in two different queues—`queue1` and `queue2`—and tries to find a match between each two entities from different inputs. The match condition is defined by the user, and `MatchQ` checks it on any arrival. Once the match is found, the two entities are passed to the two output ports correspondingly; otherwise, the incoming entity is stored in its queue (Queue objects are used). After a match, the entities leave the object immediately. The functionality and interface of `Queue`, including preemption, timeouts, etc., is fully inherited by `MatchQ` object.

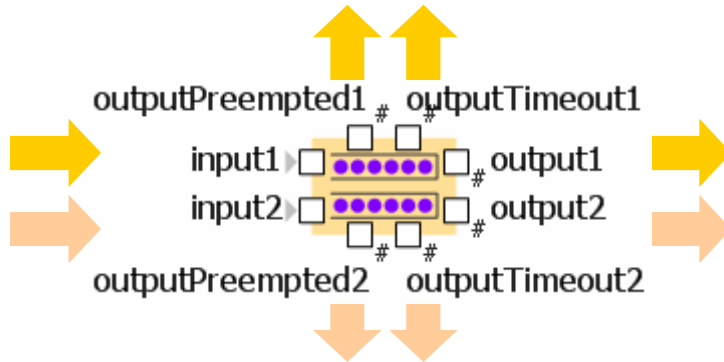


Figure 15 MatchQ object

### Variables

Type	Name	Description
Entity	entity1	Variables of encapsulated queue1, see Queue
int	position1	
double	timeoutValue1	
Entity	entity2	Variables of encapsulated queue2, see Queue
int	position2	
double	timeoutValue2	

### Functions

Return type	Name	Description
void	block1()	Blocks the input1 port of the object.
void	unblock1()	Unblocks the input1 port of the object.
boolean	blocked1()	Returns true if the input1 is blocked, otherwise false.
void	block2()	Blocks the input2 port of the object.
void	unblock2()	Unblocks the input2 port of the object.
boolean	blocked2()	Returns true if the input2 is blocked, otherwise false.
int	size1()	Functions of encapsulated queue1, see Queue
Entity	get1( int i )	

Entity	remove1( int i )	Functions of encapsulated queue1, see Queue
boolean	canEnter1()	
TimedDataSet	getStatsSize1()	
int	size2()	
Entity	get2( int i )	
Entity	remove2( int i )	
boolean	canEnter2()	
TimedDataSet	getStatsSize2()	Resets the queue size statistics for both queues.
void	resetStats()	

### Parameters

Type	Name	Default value	Description	
code	onEnter1		Parameters of the encapsulated queue1, see Queue	
code	onExitPreempted1			
code	onExitTimeout1			
int	queueType	FIFO		
int	capacity1			
boolean	preemption1	false		
boolean	timeout1	false		
code<double>	timeoutTime1	0		
ShapeBase	animationShape1			
int	animationType1	QUEUE		
boolean	animationForward1	true		
code	onEnter2			Parameters of the encapsulated queue2, see Queue
code	onExitPreempted2			
code	onExitTimeout2			

int	queueType2	FIFO	
int	capacity2		
boolean	preemption2	false	
boolean	timeout2	false	
code<double>	timeoutTime2	0	
ShapeBase	animationShape2		
int	animationType2	QUEUE	
boolean	animationForward2	true	
code<boolean>	matchCondition	true	Condition evaluated when a new entity arrives at either of the queues for that entity and each entity in the opposite queue to determine match.
code	onMatch		Code executed when the match is found.
code	onExit1		Code executed when the entity matched from queue1 exits the object.
code	onExit2		Code executed when the entity matched from queue2 exits the object.
boolean	statsEnabled	false	If true, statistics are collected for this object, otherwise not.

### Comments

When writing the `matchCondition` and `onMatch`, you may refer to the entity in `queue1` as `entity1`, and the entity in `queue2` as `entity2`. For example, if you have declared two subtypes of `Entity`: `Passenger` and `Baggage`, each having the `id` field; then `match` can be defined as `((Passenger)entity1).id == ((Baggage)entity2).id`.

Entity positions in the queues and timeouts are available at the time `matchCondition` and `onMatch` are evaluated according to Queue API.

By default, the match condition is true, which means any two entities will match. In this case `MatchQ` works as a synchronization object: it synchronously outputs pairs of entities.

### 3.1.13 RestrictedAreaStart

Allows only a specified number of entities to be in a restricted area of the model. The area is ended with `RestrictedAreaEnd` block. The `RestrictedAreaEnd` block must contain reference to the first block to determine which area to end. Using `RestrictedAreaStart` block you can limit the number of entities passing to the sub model that can hold only a specific number of entities at any moment of time. Number of entities allowed to be in that area is specified by user as parameter of the block. Entities are accepted until the number is reached. If the area is filled up to capacity then new entities will be allowed to pass through the block only when some entities will leave the area.



Figure 16 `RestrictedAreaStart` object

#### Variables

Type	Name	Description
Entity	<code>entity</code>	The current entity.

#### Functions

Return type	Name	Description
boolean	<code>blocked()</code>	Returns true if the input is blocked, otherwise false.

integer	allowedEntities()	Returns current number of entities allowed to enter the area.
---------	-------------------	---

### Parameters

Type	Name	Default value	Description
integer	entitiesLimit	1	Number of entities allowed to be in the restricted area.
code	onEnter		Code executed when the entity enters the object.

### Comments

RestrictedAreaStart is an object that does not store entities inside: it accepts an entity only if the number of entities in the restricted area is less than allowed.

If you will not place RestrictedAreaEnd object or no one of RestrictedAreaEnd objects will refer to this RestrictedAreaStart object then only first entitiesLimit entities will be allowed to enter area. After this object will remain blocked all time.

## 3.1.14 RestrictedAreaEnd

Indicates end of a restricted area. As a parameter contains reference to the RestrictedAreaStart object which starts the area.



Figure 17 RestrictedAreaEnd object

### Variables

Type	Name	Description
------	------	-------------

Entity	entity	The current entity.
--------	--------	---------------------

### Functions

Return type	Name	Description
void	block()	Blocks the input port of the object.
void	unblock()	Unblocks the input port of the object.
boolean	blocked()	Returns true if the input is blocked, otherwise false.

### Parameters

Type	Name	Default value	Description
RestrictedAreaStart	restrictedAreaStart	null	Name of the RestrictedAreaStart object which starts the area.
code	onEnter		Code executed when the entity enters the object.

### Comments

RestrictedAreaEnd is an object that does not store entities inside: after updating information about entities allowed to enter the area it passes current entity further immediately.

If restrictedAreaStart parameter is undefined then at the moment when first entity enters the object an error will arise.

## 3.2 Working with entity contents

### 3.2.1 BatchQ

Accumulates the specified number of entities in the queue, then creates a new entity (“container”), adds the accumulated entities to its contents, and outputs it. The class of the new entity is specified by the user. Once created, a batch leaves the object immediately.

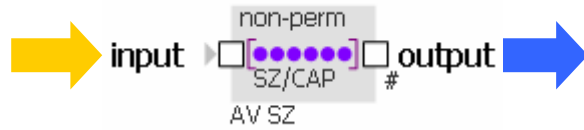


Figure 18 BatchQ object

## Variables

Type	Name	Description
Entity	entity	The current incoming entity.
Entity	container	The container entity created at this object.
int	position	Position of the entity in the queue and in the container.

## Functions

Return type	Name	Description
void	block()	Blocks the input port of the object.
void	unblock()	Unblocks the input port of the object.
boolean	blocked()	Returns true if the input is blocked, otherwise false.
int	size()	Functions of the encapsulated queue, see Queue.
Entity	get( int i )	
Entity	remove( int i )	
boolean	canEnter()	
TimedDataSet	getStatsSize()	
void	resetStats()	

## Parameters

Type	Name	Default value	Description
------	------	---------------	-------------



code	onEnter		Code executed when the entity enters the object.
code	onAdd		Code executed for each entity when the container is formed, i.e. just before it exits.
code	onExit		Code executed when the container exits the object.
int	size		The size of the batch.
boolean	permanent	true	If true, the entities are not added to the contents of the container entity, although onAdd is executed. If false, the entities are added and can be extracted later on, e.g., by Unbatch.
Class	newEntity	Entity.class	Type of container entity to create.
Int	queueType	FIFO	Parameters of encapsulated queue, see Queue.
boolean	statsEnabled	false	
ShapeBase	animationShape		
Int	animationType	AUTO	
boolean	animationForward	true	

### Comments

When onAdd is executed, the position of the entity in the queue and in the container is available as `position`. When onExit is executed, all entities are removed from the queue by that time.

Preemption and timeout options of the encapsulated Queue are not available.

Example of a permanent batch. Suppose you wish to accumulate `n` entities of type `Packet` having the field `size` and create a new one of the same type having the size equal to the sum of all packet sizes plus a constant `c`. Then you leave the parameter `permanent` set to `true`, in `newEntity` you specify `new Packet()`, in `onAdd` you write `((Packet)container).size += ((Packet)entity).size`, and finally in `onExit`: `((Packet)container).size += c`.

Consider also using Pickup to construct temporary batches.

The size of batch may be changed dynamically.

### 3.2.2 Unbatch

Extracts all entities contained in the incoming entity (“container”), if any, and outputs them one by one preserving the order they were stored in. The original container entity is discarded. The operation takes zero time; once an entity enters the Unbatch object, all its contents leave it immediately.



Figure 19 Unbatch object

#### Variables

Type	Name	Description
Entity	container	The current container entity.
Entity	entity	The current entity extracted from the container.
int	position	Position of the entity in the container contents.

#### Functions

Return type	Name	Description
void	block()	Blocks the input port of the object.
void	unblock()	Unblocks the input port of the object.
boolean	blocked()	Returns true if the input is blocked, otherwise false.

## Parameters

Type	Name	Default value	Description
code	onEnter		Code executed when the container enters the object.
code	onExit		Code executed for each entity that is extracted from the container and exits the object.

## Comments

You may use onExit to transfer any properties of the container to the exiting entities. Position of the entity in the container is available as `position` when onExit is executed.

Permanent batch cannot be unbatched because the entities that have been used to form it are discarded—see BatchQ.

Consider also using Dropoff to extract the entity contents.

### 3.2.3 Pickup

Removes entities from a Queue object and adds them to the contents of the incoming entity (“container”). The Queue object must be connected to the inputPickup port. Entities are selected from the Queue and added, according to the specified condition. The condition may depend on the entity being added as well as on the container itself. The operation takes zero time; once the container enters the Pickup, it leaves it immediately.



Figure 20 Pickup object

## Variables

Type	Name	Description
Entity	container	The current container entity.
int	initialSize	The size of the container when it arrives to the object.
Entity	entity	The current entity in the queue.
int	position	Position of the entity in the queue, not in the container.

## Functions

Return type	Name	Description
void	block()	Blocks the input port of the object.
void	unblock()	Unblocks the input port of the object.
boolean	blocked()	Returns true if the input is blocked, otherwise false.

## Parameters

Type	Name	Default value	Description
code	onEnter		Code executed when the container enters the object.
code	onPickup		Code executed for each entity that is removed from the queue and added to the container.
code	onExit		Code executed when the container exits the object.
code<boolean>	pickupCondition	true	Condition evaluated for each entity in the queue to determine whether it should be picked up.

## Comments

While pickupCondition is evaluated, the entity is, of course, still in the queue, and its position is available as position. When onPickup is executed, the entity has been already

removed from the Queue object and added to the container entity.

When the Queue is empty, the pickupCondition is not evaluated.

As long as the default value of pickupCondition is true, the container by default picks up the whole contents of the queue no matter what. If you wish, say, to pickup no more than 20 entities, you may write: `container.contents.size() <= initialSize + 20`. If you wish to pickup as many entities as possible, but not exceeding the maximum size of the container, say `maxSize`, then the correct condition would be: `container.contents.size() <= maxSize`. If you wish to pickup only entities of type `Cat` whose age is less than 3 years, you write: `entity instanceof Cat && ((Cat)entity).age < 3`.

### 3.2.4 Dropoff

Removes entities from the contents of the incoming entity (“container”) and outputs them via `outputDropoff` port. Entities are selected to be removed according to the specified condition that may depend on the entity being removed as well as on the container itself. The operation takes zero time; once the container enters the Dropoff, it leaves it immediately.

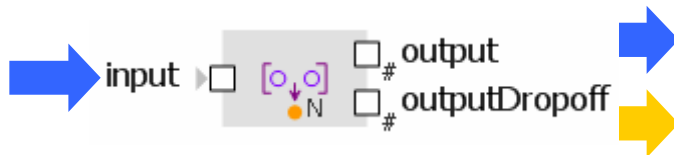


Figure 21 Dropoff object

#### Variables

Type	Name	Description
Entity	container	The current container entity.
int	initialSize	The size of the container when it arrives to the object.
Entity	entity	The current entity in the container being considered.
int	position	Position of the entity in the container.

## Functions

Return type	Name	Description
void	block()	Blocks the input port of the object.
void	unblock()	Unblocks the input port of the object.
boolean	blocked()	Returns true if the input is blocked, otherwise false.

## Parameters

Type	Name	Default value	Description
code	onEnter		Code executed when the container enters the object.
code	onExit		Code executed when the container exits the object.
code	onExitDropoff		Code executed for each entity that is removed from the container and exits.
code<boolean>	dropoffCondition	container.contents.size() > 0	Condition evaluated for each entity in the container to determine whether it should be dropped off. By default is set to <code>container.contents.size() &gt; 0</code> (all entities are dropped off).

## Comments

While `dropoffCondition` is evaluated the entity is of course still in the container, and its position is available as `position`. When `onExitDropoff` is executed the entity has been already removed from the container.

When the container is empty, the `dropoffCondition` is not evaluated.

Similarly to `Pickup`, you can control the process of dropping entities off by writing different

expressions in `dropoffCondition`. To drop off not more than 10 entities write `initialSize - container.contents.size() < 10`. When dropping off entities of type `Passenger`, if their destination is equal to `here` (defined in the upper level object), write: `((Passenger)entity).destination == here`.

## 3.3 Processing

### 3.3.1 Schedule

Allows scheduling availability of resources and downtimes. Schedule object is not associated with any real source units. It only informs Delay, Server, Resource and NetworkResource objects how many resource units should be available in the particular moment of time according specified schedule. Several Resource objects can use one Schedule object. In case of Delay the number of resources is considered as capacity.

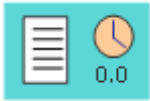


Figure 22 Schedule object

#### Functions

Return type	Name	Description
int	<code>getLimit()</code>	Returns maximal quantity of resource units which can be available at this moment of time (some resources can be failed).
int	<code>getCapacity()</code>	Returns quantity of currently available resource units.
int	<code>totalFailures()</code>	Returns number of total failures.
int	<code>totalRepairs()</code>	Returns number of total repairs.

## Parameters

Type	Name	Default value	Description
LookupTable	scheduleTable		Name of lookup table which contains moment of capacity changing and new values of capacity.
code<double>	TTF	infinity	Time to failure. Expression used to evaluate time between last repair and next failure.
code<double>	TTR	0	Time to repair. Expression used to evaluate time needed to repair from the last failure.
real	period	aperiodic	Period for lookup table.
int	initialCapacity	1	Initial capacity of the Schedule object.

## Comments

Lookup table for Schedule must contain timestamps of capacity changes and values to which capacity will be increased or decreased in those times.

### 3.3.2 Delay

Delays entity by the specified delay time. Multiple entities (up to the specified capacity) can be delayed simultaneously and independently, unlike in Server object. The delay time is calculated individually for each entity. Once the delay time is over for an entity, it leaves the object immediately. Delay will not accept an incoming entity if the capacity is reached.

Capacity of the Delay objects can be controlled by Schedule object. Schedule object will automatically control capacity according specified TTF, TTR and working schedule.



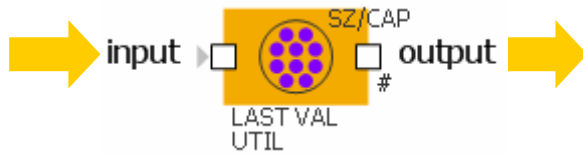


Figure 23 Delay object

## Variables

Type	Name	Description
Entity	entity	The current entity.
double	delayTimeValue	The delay value set for the current entity.

## Functions

Return type	Name	Description
void	block()	Blocks the input port of the object.
void	unblock()	Unblocks the input port of the object.
boolean	blocked()	Returns true if the input is blocked, otherwise false.
int	size()	The number of entities currently delayed.
Entity	get( int i )	Returns the i-th entity.
boolean	canEnter()	Returns true if a new entity may be accepted, i.e. the Delay capacity is not yet reached.
TimedDataSet	getStatsUtilization()	Returns the statistics on the delay utilization.
void	resetStats()	Resets the statistics collected for this object.

## Parameters

Type	Name	Default value	Description
code	onEnter		Code executed when the entity enters the object.
code	onExit		Code executed when the entity exits the object.

code<double>	delayTime	triangular( 0.5, 1, 1.5 )	Expression evaluated to obtain the delay time for the current entity.
double	scale	1	If 'length of polyline' is selected as delayTime then actual delay time will be equal to length of polyline multiplied by that scale. Thus scale may be 1/speed.
int	capacity	1	The capacity of the Delay object.
boolean	statsEnabled	false	If true, statistics are collected for this object, otherwise not.
ShapeBase	animationShape		Animation template for the Delay. The type depends on the animator.
int	animationType	AUTO	The animator type for the Delay. Valid types are: AUTO SINGLE SET BAG ARRANGED MOVEMENT
boolean	animationForward	true	Orientation of polyline-based animations
Schedule	schedule	without_schedule	Name of schedule object to control capacity of the Delay.

### Comments

When onEnter is executed, the delay value for the entity is known and available as delayTimeValue.

Delay time may be made stochastic (as e.g. the default value), deterministic, depending on

the entity or any other information. Suppose, you are delaying entities of type `Packet` proportionally to the value of its field `size`, then you write: `((Packet)entity).size*k`.

Capacity of the Delay may be changed dynamically by calling `set_capacity` function or using Schedule object. One Schedule object can control capacity of several Delays. If the capacity was decreased to the value less than the number of entities currently delayed those entities will remain in the Delay according they times. Thus `size()` (number of entities being delayed) can be greater than capacity. The object will accept new entities only after those entities will exit and `size()` will be less than capacity.

### 3.3.3 Server

Delays entity until it receives the specified amount of service time. Multiple entities share Server, i.e., the service time received by an entity during a model time unit is reverse proportional to the number of entities being served concurrently (like in CPU processing multiple tasks). Therefore the entities served simultaneously affect each other, unlike in Delay object. The required service time is calculated individually for each entity. Once the entity has received the full amount of service, it leaves the object immediately. Server will not accept an incoming entity if the capacity is reached.

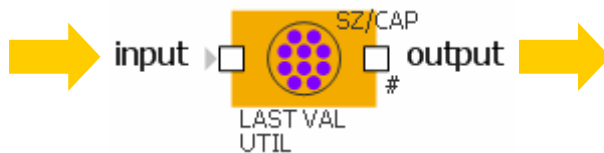


Figure 24 Server object

#### Variables

Type	Name	Description
Entity	Entity	The current entity.
double	serviceTimeValue	The service time required for the current entity.

#### Functions

Return type	Name	Description
-------------	------	-------------

void	block()	Blocks the input port of the object.
void	unblock()	Unblocks the input port of the object.
boolean	blocked()	Returns true if the input is blocked, otherwise false.
int	size()	The number of entities currently delayed.
Entity	get( int i )	Returns the i-th entity.
boolean	canEnter()	Returns true if a new entity may be accepted, i.e., the Server capacity is not yet reached.
TimedDataSet	getStatsUtilization()	Returns the statistics on the server utilization.
void	resetStats()	Resets the statistics collected for this object.

### Parameters

Type	Name	Default value	Description
code	onEnter		Code executed when the entity enters the object.
code	onExit		Code executed when the entity exits the object.
code<double>	serviceTime	exponential( 1 )	Expression evaluated to obtain the service time for the current entity.
int	capacity	100	The capacity of the Server object.
boolean	statsEnabled	false	If true, statistics are collected for this object, otherwise not.
double	performance	1	The Server performance coefficient.
ShapeBase	animationShape		Animation template for the Server. The type depends on the animator.

int	animationType	AUTO	The animator type for the Server. Valid types are: AUTO  SINGLE  SET  BAG  ARRANGED
-----	---------------	------	---

### Comments

The performance of Server means how fast does the Server work with respect to the service time required by the entities. For example, if there is only one entity being processed and it required 10 service time units, the Server with performance 2 will complete this task in  $10/2 = 5$  time units.

When onEnter is executed, the service time for the entity is known and available as `serviceTimeValue`.

Service time may be made stochastic (as e.g. the default value), deterministic, depending on the entity or any other information. Suppose you are servicing entities of type `Task` proportionally to the value of its field `complexity`; then you write:  
`((Task)entity).complexity*k.`

## 3.4 Working with resources

### 3.4.1 Resource

Provides units that are seized and released by entities via `SeizeQ`, `ProcessQ` and `Release` objects. Resource units are objects of the same class `Entity` or any user-defined subclass. Resource object can generate, store, give out and take back units. The capacity of the resource can be changed dynamically. Resource is best used to model relatively small amounts of individually distinguishable units like operators, machines, devices, critical sections, etc.

A unit may be possessed by only one entity at a time; therefore, entities compete for resources. In case of multiple simultaneous requests, Resource satisfies them first in the order of their priorities and then in the order of their timestamps. Priority in this case is a priority parameter of SeizeQ object that tries to seize resource for an entity (do not confuse it with the priority field of the entity). The timestamp is the time the entity started waiting for the resource (got into the queue of SeizeQ). The older requests are served first. In case a request cannot be satisfied because there is insufficient number of units available, it is skipped, and other requests with lower priority or a more recent timestamp may be served.

Resource has a single port access that must be connected to the ports access of SeizeQ, Release or ProcessQ objects. One Resource may be connected to several such objects; and vice versa, one SeizeQ or Release object may be connected to multiple Resource objects.



Figure 25 Resource object

### Variables

Type	Name	Description
Entity	unit	The current resource unit.

### Functions

Return type	Name	Description
int	size()	The number of free resource units.
Entity	get( int i )	Returns the i-th unit.
int	getSeized()	Returns the number of seized resource units.
TimedDataSet	getStatsUtilization()	Returns the statistics on the resource utilization.
void	resetStats()	Resets the resource utilization statistics.

## Parameters

Type	Name	Default value	Description
code	onSeizeUnit		Code executed for a unit when it is seized.
code	onReleaseUnit		Code executed for a unit when it is released.
code	onGenerate		Code executed when new unit is generated.
int	capacity	1	The capacity of the resource.
code<Entity>	newUnit	Entity.class	Type of resource unit.
boolean	statsEnabled	false	If true, statistics are collected for this object, otherwise not.
ShapeBase	animationShape		Animation template for the resource. The type depends on the animator.
int	animationType	AUTO	The animator type for the queue. Valid types are: AUTO SINGLE SET BAG ARRANGED

## Comments

On startup the Resource object generates capacity number of units based on newUnit type. After each generation onGenerate code is executed (resource as unit can be accessed from that code). The capacity of the resource may be changed dynamically. If it grows, the additional units are generated. If it becomes less, the appropriate number of free units is deleted; if this is not sufficient, the Resource will wait for the units to be released to delete them. Eventually the number of existing units will equal capacity.

onSeizeUnit and onReleaseUnit are executed for every unit being seized or released; i.e., if 5

units are seized, `onSeizeUnits` is called 5 times. When `onSeizeUnit` is called, the units are already removed from the Resource. When `onReleaseUnit` is called, the units are already added.

The seized units are not removed from the Resource animator, as in most cases, you need just to change their visual appearance to reflect they are busy.

### 3.4.2 SeizeQ

Seizes the number of units of the specified resource required by the entity. Encapsulates a Queue object where entities wait for the resource to be available. The resource is requested for the first entity in the queue, and until that entity seizes the resource, the resources for the entities behind are not issued (although they may be satisfiable). The request contains the priority parameter of SeizeQ object and the entity timestamp – the time the entity entered SeizeQ. Once the resource is seized for an entity, it leaves the object immediately. One or several Resource objects must be connected to the access port of SeizeQ. The Resource object is selected individually for each entity as well as the number of units. The functionality and interface of Queue, including preemption, timeouts, etc., is fully inherited by SeizeQ object.

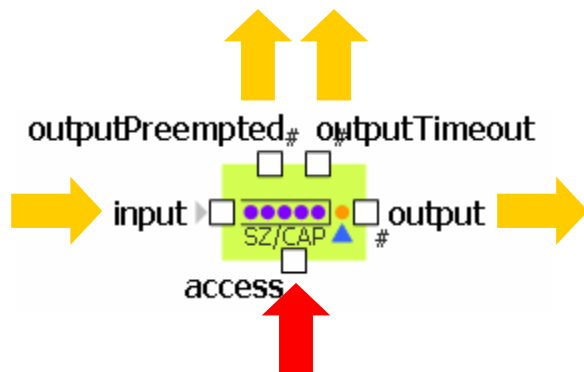


Figure 26 SeizeQ object

#### Variables

Type	Name	Description
Entity	entity	The current entity.



Resource	resource	The selected Resource object.
int	quantityValue	The number of units requested by the entity.
Entity	unit	The current unit being seized.
int	position	Variables of the encapsulated queue, see Queue.
double	timeoutValue	

### Functions

Return type	Name	Description
void	block()	Blocks the input port of the object.
void	unblock()	Unblocks the input port of the object.
boolean	blocked()	Returns true if the input is blocked, otherwise false.
int	size()	Functions of the encapsulated queue, see Queue.
Entity	get( int i )	
Entity	remove( int i )	
boolean	canEnter()	
TimedDataSet	getStatsSize()	
void	resetStats()	

### Parameters

Type	Name	Default value	Description
code	onEnter		Parameters of the encapsulated queue, see Queue.
code	onExitPreempted		
code	onExitTimeout		
int	queueType	FIFO	
int	capacity		
int	entitiesToAnimate	all	

boolean	preemption	false	
boolean	timeout	false	
code<double>	timeoutTime	infinity	
ShapeBase	animationShapeQ		
int	animationTypeQ	AUTO	
boolean	animationForwardQ	true	
boolean	statsEnabled	false	
code	onSeizeUnit		Code executed when a unit is being seized by the entity.
code	onExit		Code executed when the entity exits the object.
code<int>	quantity	1	Expression evaluated to obtain the required number of resource units.
code<Resource>	selectResource	null	Expression evaluated to select the Resource object. If returns null, the object is randomly selected among the connected ones.
int	priority	0	Priority of this SeizeQ object, the larger the higher.

### Comments

If no Resource objects are connected to SeizeQ object, or the Resource object selected by selectResource is not connected, a runtime error is signaled when the entity tries to seize the resource units. If selectResource returns null, the Resource object is selected randomly among the connected ones.

Anytime a new entity appears at the position 0 of the queue, the required number of units is evaluated and the Resource object is selected by execution of quantity and selectResource – in this particular order. As long as, depending on the Queue type, an entity may be shifted back by another entity and then reach position 0 again, this procedure may happen several times for the same entity.

onSeizeUnit is executed for every unit being seized; i.e., if 5 units are seized, onSeizeUnits is called 5 times. When onSeizeUnit is called, the unit is already added to the resources vector of entity. When onExit is called, the entity is already removed from the queue.

### 3.4.3 Release

Releases resource units previously seized by the entity. You may either specify the quantity of units or provide a condition to select them. The Resource object may also be selected individually for each entity. One or several Resource objects must be connected to the access port of Release. The release operation always takes zero time, so once entered, the entity leaves the Release object immediately.

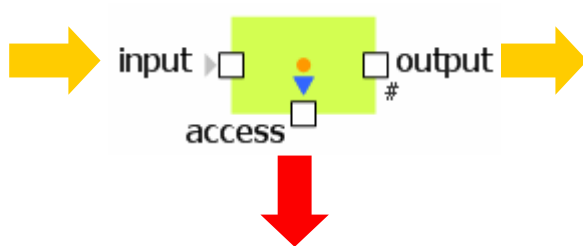


Figure 27 Release object

#### Variables

Type	Name	Description
Entity	entity	The current entity.
Resource	resource	The selected Resource object.
int	quantityValue	The number of units being released by the entity.
Entity	unit	The current unit being released or tested.

#### Functions

Return type	Name	Description
void	block()	Blocks the input port of the object.
void	unblock()	Unblocks the input port of the object.

boolean	blocked()	Returns true if the input is blocked, otherwise false.
---------	-----------	--

### Parameters

Type	Name	Default value	Description
code	onEnter		Code executed when the entity enters the object.
code	onReleaseUnit		Code executed when a unit is being released by the entity.
code	onExit		Code executed when the entity exits the object.
code<int>	quantity	1	Expression evaluated to obtain the number of resource units to be released.
code<Resource>	selectResource	random	Expression evaluated to select the Resource object. If returns null, the object is randomly selected among the connected ones.
code<boolean>	selectUnit	true	Condition evaluated for each unit in the container to determine whether it should be dropped off.

### Comments

If no Resource objects are connected to Release object, or the Resource object selected by selectResource is not connected, a runtime error is signaled when the entity tries to release the resource units. If selectResource returns null, the Resource object is selected randomly among the connected ones. For example, if you wish to release all units of type Operator, you write in selectResource: `unit instanceof Operator`.

For each resource unit possessed by the entity selectUnit is executed, and if true is returned, the unit is released. No more than quantity resources will be released.

onReleaseUnit is executed for every unit being released; i.e., if 5 units are released, onReleaseUnit is called 5 times. When onReleaseUnit is called, the unit is already removed

from the `resources` vector of entity.

### 3.4.4 ProcessQ

Seizes resource units for the entity, delays the entity, and releases the seized units. Encapsulates a sequential combination of `SeizeQ`, `Delay` and `Release`. One or several Resource objects must be connected to the access port of `ProcessQ`. The functionality and interface of these three objects is inherited by `ProcessQ` object (except for `Release` that is tuned to release exactly those units that were previously seized by `SeizeQ`).

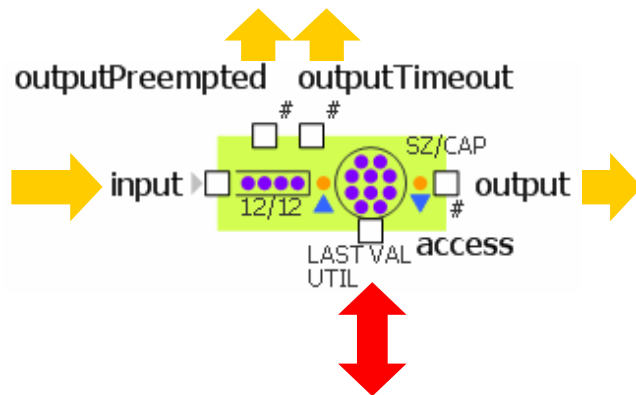


Figure 28 ProcessQ object

#### Variables

Type	Name	Description
Entity	<code>entity</code>	The current entity.
Resource	<code>resource</code>	The selected Resource object.
int	<code>quantityValue</code>	The number of units requested by the entity.
Entity	<code>unit</code>	The current unit being seized or released.
int	<code>position</code>	Variables of the encapsulated queue, see Queue.
double	<code>timeoutValue</code>	
double	<code>delayTimeValue</code>	Variables of the encapsulated delay, see Delay.

## Functions

Return type	Name	Description
void	block()	Blocks the input port of the object.
void	unblock()	Unblocks the input port of the object.
boolean	blocked()	Returns true if the input is blocked, otherwise false.
int	sizeQ()	Functions of the encapsulated queue, see Queue.
Entity	getQ( int i )	
Entity	removeQ( int i )	
boolean	canEnter()	
TimedDataSet	getStatsSizeQ()	
int	sizeDelay()	Functions of the encapsulated delay, see Delay.
Entity	getDelay( int i )	
TimedDataSet	getStatsUtilization()	
void	resetStats()	Resets the statistics collected at this object.

## Parameters

Type	Name	Default value	Description
Code	onEnter		Parameters of the encapsulated seizeQ, see SeizeQ.
Code	onExitPreempted		
Code	onExitTimeout		
Int	queueType	FIFO	
Int	queueCapacity		
Boolean	preemption	false	
Boolean	timeout	false	
code<double>	timeoutTime	0	
ShapeBase	animationShapeQ		

Int	animationTypeQ	AUTO	
boolean	animationForwardQ	true	
code	onSeizeUnit		
code<int>	quantity	1	
code<Resource>	selectResource	null	
int	priority	0	
code<double>	delayTime	exponential(1)	Parameters of the encapsulated delay, see Delay.
int	delayCapacity	100	
ShapeBase	animationShapeDelay		
int	animationTypeDelay	AUTO	
boolean	animationForwardDelay	true	
code	onReleaseUnit		Parameter of the encapsulated release, see Release.
code	onEnterDelay		Code executed when the entity has seized the resource and enters delay.
code	onExit		Code executed when the entity exits the object.
boolean	statsEnabled	false	If true, statistics are collected for this object, otherwise not.

### Comments

Just in case you wish to do something after the entity has seized the resource units and before the delay starts, onEnterDelay parameter is provided. It is actually the onEnter code of the encapsulated Delay object.

## 3.5 Transportation

### 3.5.1 Conveyor

Moves entities along a path with the specified length at a certain speed, preserving their order and a certain space between them. If the entity that has reached the end of the Conveyor is unable to exit, it stops there. The non-accumulating conveyor in this case will stop moving at all, whereas the accumulating Conveyor will continue moving those entities that have enough space ahead of them, i.e., the distance to the previous entity is larger than the space parameter. The speed and accumulating parameters of Conveyor may be changed dynamically at runtime.

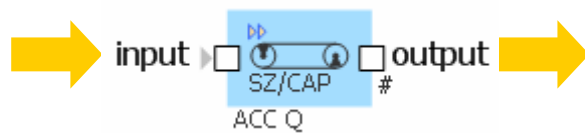


Figure 29 Conveyor object

#### Variables

Type	Name	Description
Entity	entity	The current entity.

#### Functions

Return type	Name	Description
void	block()	Blocks the input port of the object.
void	unblock()	Unblocks the input port of the object.
Boolean	blocked()	Returns true if the input is blocked, otherwise false.
int	size()	The number of entities currently being conveyed.
Entity	get( int i )	Returns the i-th entity (counted from the exit).



Double	getPosition( int i )	Returns position of the i-th entity (0 – entry, length – exit).
double	getSpace( int i )	Returns space ahead of the i-th entity (for the first entity this is the distance to the exit).
int	getAccQueueSize()	Returns size of current accumulation queue in entities.
boolean	canEnter()	Returns true if a new entity may be accepted.

### Parameters

Type	Name	Default value	Description
code	onEnter		Code executed when the entity enters the object.
code	onAtExit		Code executed when the entity reaches the end of the conveyor and exits.
code	onExit		Code executed when the entity exits the object.
code	onOpen		Code executed when the (previously closed for whatever reason) Conveyor becomes ready to accept a new entity.
code	onClose		Code executed when the (previously open) Conveyor becomes closed for whatever reason.
double	length	100	Length of the Conveyor.
double	scale	1	If 'length of polyline' is selected as length parameter then actual length of conveyor is calculated as length of polyline in pixels multiplied by scale.
double	speed	10	Speed at which the Conveyor moves.
double	space	10	Minimum space between the entities.

int	capacity	100	The capacity of the Conveyor object.
boolean	accumulating	true	If true, entities will move even if there is a jam at the output, otherwise the whole Conveyor stops.
ShapePoly	animationShape		Animation template for the Conveyor. The animator type is always CONVEYOR.
boolean	animationForward	true	Orientation of the animation.

### Comments

Conveyor will accept an entity if all the following conditions are true (this may be tested by `canEnter()`):

- The previous entity has moved at least a space away from the entry
- (Non-accumulating only) the conveyor is not blocked by an entity unable to exit
- The capacity is not reached

The “natural” capacity of Conveyor is limited by length/space. The capacity parameter may limit it further.

The entities are assumed to have zero length. Use space parameter to model the entity length if needed.

`getPosition()` and `getSpace()` functions are explained on the figure .

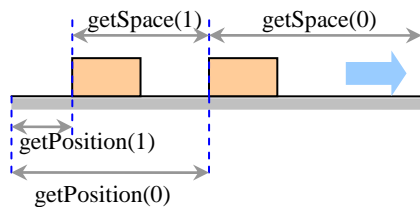


Figure 30 `getSpace()` and `getPosition()` functions

You may use `onOpen` and `onClose` parameters for better control of the other objects that feed the Conveyor.

The speed, accumulating and capacity parameters of the Conveyor may be changed dynamically.

### 3.5.2 Lane

Moves entities along a path with the specified length with individual (and possibly different) speeds. Depending on the overtake parameter, the entities with different speeds will either overtake each other or slow down and follow one another preserving the space in between (like in accumulating Conveyor). If the entity that has reached the end of Lane is unable to exit, it stops there. The overtake parameter of Lane may be changed dynamically at runtime.

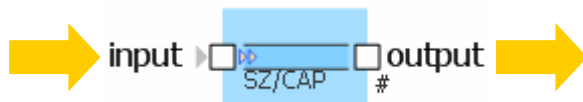


Figure 31 Lane object

#### Variables

Type	Name	Description
Entity	entity	The current entity.
double	speedValue	The “natural” speed value set for the current entity.

#### Functions

Return type	Name	Description
void	block()	Blocks the input port of the object.
void	unblock()	Unblocks the input port of the object.
boolean	blocked()	Returns true if the input is blocked, otherwise false.
int	size()	The number of entities currently on the lane.
Entity	get( int i )	Returns the i-th entity (counted from the exit).
double	getSpeed( int i )	Returns the current speed of the i-th entity.

double	getPosition( int i )	Returns position of the i-th entity (0 – entry, length – exit).
double	getSpace( int i )	Returns space ahead of the i-th entity (for the first entity this is the distance to the exit).
boolean	canEnter()	Returns true if a new entity may be accepted.

### Parameters

Type	Name	Default value	Description
code	onEnter		Code executed when the entity enters the object.
code	onAtExit		Code executed when the entity reaches the end of the lane and becomes to exit.
code	onExit		Code executed when the entity exits the object.
code	onOpen		Code executed when the (previously closed for whatever reason) Lane becomes ready to accept a new entity.
code	onClose		Code executed when the (previously open) Lane becomes closed for whatever reason.
double	length	100	Length of the Lane.
double	scale	1	If 'length of polyline' is selected as length parameter then actual length of lane is calculated as length of polyline in pixels multiplied by scale.
double	speed	10	Speed at which the entity moves along the Lane.
double	space	10	Minimum space between the entities.
int	capacity	100	The capacity of the Lane object.

boolean	overtake	true	If true, entities may overtake each other.
ShapePoly	animationShape		Animation template for the Lane. The animator type is always LANE.
boolean	animationForward	true	Orientation of the animation.

### Comments

Lane will accept an entity if all the following conditions are true:

- (Overtake is false) the previous entity has moved at least space away from the entry
- The capacity is not reached

The “natural” capacity of Lane with no overtakes is limited by length/space. The capacity parameter may limit it further.

You may use onOpen and onClose parameters for better control of the other objects that feed the Lane.

The speed, accumulating and capacity parameters of the Lane may be changed dynamically.

### 3.5.3 Node

A source, destination and/or routing point of a transportation network. Is used in combination with Segment objects that provide transportation of entities between nodes. Segment objects are usually connected to the network port of Node, whereas entities (“transporters”) that finish or start transportation at the Node exit/enter the Node via output and input ports correspondingly.

For any transporter entering Node (either via network port or input port), Node checks its destination field (of type Node). If destination equals this Node, it is set to null, and the transporter exits via output port. If destination is null, the transporter is added to the set of idle transporters parked at this node. If destination is neither null nor this Node, the transporter is forwarded to another node according to the routing table. The routing tables at each Node are automatically formed when the model is created and the network topology becomes known.

A transporter passed to the output port leaves the Node immediately. On the contrary, a transporter that is passed to the network port (entering the network or “transit”) may be kept in a queue if the corresponding segment cannot accept the transporter immediately. The maximum number of idle transporters and transporters waiting to enter the network (“pending” transporters) is limited by `capacityIdle` and `capacityPending` parameters. Node would not accept an incoming transporter if either of the capacities is reached.

You can order a transporter to the Node by calling its function `order()`. The network then finds an idle transporter at a Node from which this Node can be reached and sends it to this Node. If there are several nodes that can serve the order, the nearest one is chosen (the network distances are compared either by the number of segments or by their lengths depending on chosen routing type). If there are no such nodes, the order is kept in a global queue. You can cancel such pending order by calling the function `cancelOrder()` of the Node that has requested a transporter.

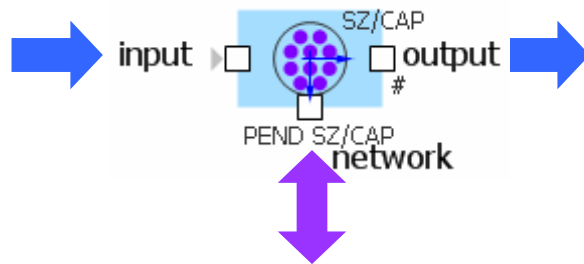


Figure 32 Node object

### Constants

Type	Name	Value	Description
int	MINIMUM_ROUTE_LENGTH	1	Constant is to be used as parameter for <code>setRoutingType</code> function to select shortest path based routing.
int	MINIMUM_NUMBER_OF_HOPS	2	Constant for selecting routing based on minimum number of hops.

## Variables

Type	Name	Description
Entity	entity	The current transporter.

## Functions

Return type	Name	Description
void	block()	Blocks the input port of the object.
void	unblock()	Unblocks the input port of the object.
boolean	blocked()	Returns true if the input is blocked, otherwise false.
void	blockNetwork()	Blocks the network port of the object.
void	unblockNetwork()	Unblocks the network port of the object.
boolean	blockedNetwork()	Returns true if the network port is blocked, otherwise false.
int	size()	The number of idle transporters at this Node.
Entity	get( int i )	Returns the i-th idle transporter.
boolean	canEnter()	Returns true if a new transporter may be accepted at the input port.
void	order()	Brings an idle transporter at this Node. The transporter exits via output port.
boolean	cancelOrder()	Cancels a previous order. Returns true if the order was cancelled, false if it is too late.
void	setRoutingType( int t )	Static function which set routing type for the network. As parameter of function the following constants can be used: MINIMUM_ROUTE_LENGTH (network will select shortest path for passing transporter from one node to another) and MINIMUM_NUMBER_OF_HOPS(route with minimum number of hops).

## Parameters

Type	Name	Default value	Description
code	onEnter		Code executed when the transporter enters the Node via input port.
code	onExit		Code executed when the transporter exits the Node via output port.
code	onNetworkEnter		Code executed when the transporter enters the Node via network port.
code	onNetworkExit		Code executed when the transporter exits the Node via network port.
int	capacityIdle	100	The maximum number of idle transporters that can be simultaneously “parked” at this Node.
int	capacityPending	100	The maximum number of transporters simultaneously waiting to exit to the network.
ShapeBase	animationShapeIdle		Animation template for the idle transporters. The type depends on the animator.



int	animationTypeIdle	AUTO	The animator type for the idle transporters. Valid types are: AUTO SINGLE SET BAG ARRANGED QUEUE
boolean	animationForwardIdle	true	Orientation of polyline-based animations of idle transporters.
ShapeBase	animationShapePending		Animation template for the transporters waiting to enter the network. The type depends on the animator.
int	animationTypePending	AUTO	The animator type for the transporters waiting to enter the network. Valid types are: AUTO SINGLE SET BAG ARRANGED
boolean	animationForwardPending	true	Orientation of polyline-based animations of transporters waiting to enter the network.

### Comments

Unlike ports of other objects that are unidirectional, network port of Node is bi-directional: it can receive and send transporters.

By default network chooses route with minimum length to pass transporter from one location to another. `setRoutingType` function allows to select another routing – routing based on minimum number of hops. That function should be called once for any node of

network.

### 3.5.4 Segment

A network segment that provides bi-directional movement of transporters: from port left to port right and vice versa. Can be connected only to network ports of Node objects and not to other objects. Encapsulates two objects of type Lane with identical characteristics that move transporters in opposite directions. The functionality and interface of Lane is fully inherited by Segment object.

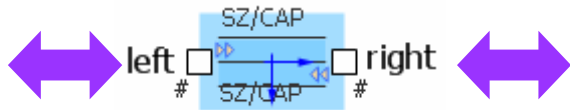


Figure 33 Segment object

#### Variables

Type	Name	Description
Entity	entity	The current transporter.
double	speedValue	The “natural” speed value set for the current transporter.

#### Functions

Return type	Name	Description
void	blockL()	Blocks the left port of the object for input.
void	unblockL()	Unblocks the left port of the object for input.
boolean	blockedL()	Returns true if the left port is blocked for input, otherwise false.
void	blockR()	Blocks the right port of the object for input.
void	unblockR()	Unblocks the right port of the object for input.

boolean	blockedR()	Returns true if the right port is blocked for input, otherwise false.
int	sizeLR()	Functions of the encapsulated laneLR, see Lane.
Entity	getLR( int i )	
double	getSpeedLR( int i )	
double	getPositionLR( int i )	
double	getSpaceLR( int i )	
boolean	canEnterLR()	
int	sizeRL()	Functions of the encapsulated laneRL, see Lane.
Entity	getRL( int i )	
double	getSpeedRL( int i )	
double	getPositionRL( int i )	
double	getSpaceRL( int i )	
boolean	canEnterRL()	

### Parameters

Type	Name	Default value	Description
code	onEnterL		Parameters of the encapsulated laneLR, see Lane.
code	onAtExitR		
code	onExitR		
code	onOpenL		
code	onCloseL		
code	onEnterR		Parameters of the encapsulated laneRL, see Lane.
code	onAtExitL		
code	onExitL		

code	onOpenR		Parameters shared by both encapsulated laneLR and laneRL, see Lane.
code	onCloseR		
double	length	100	
double	scale	1	
double	speed	10	
double	space	10	
int	capacity	100	
boolean	overtake	true	
ShapePoly	animationShape		Animation template shared by both lanes (in opposite directions). The animator type is always LANE.
boolean	animationForward	true	Orientation of the animation.
int	duplex	full	Is this segment full duplex or half.

### Comments

Segment can be either full-duplex (default) or half-duplex. In case of full duplex capacity of each lane is equal to capacity of segment (thus actual segment capacity is 2\*capacity). Half-duplex segment can accept no more than capacity entities in total no matter in which direction they move.

## 3.6 Animating active objects

Active objects that perform various operations over entities (resources, transporters) can animate their activity. As in any AnyLogic™ model, you can draw arbitrary shapes on your animation diagram and associate their graphical properties with the states of active objects. However, the most informative animation is the animation of entities that are being handled by the active objects. AnyLogic™ suggests an easy way of doing this: you draw a shape on the animation diagram—say, a polyline or a rectangle—and pass this shape to the active object. The active object then uses it as a template, a guideline, to animate the entities. This provides for fast creation of sophisticated customized animations.

There are several methods of animating entities that are possessed by the active objects called “animators.” Some animators may be applied to almost all active objects; some are specifically developed for particular classes such as Conveyor or Lane. The objects that may only possess an entity for a zero time (e.g. SelectOutput, Dropoff, Release, Split, etc.) do not have animators. If multiple animators are possible, the type of animator should be provided in a separate parameter along with a template shape. There are eight types of animators (they are constants of a class Animator):

## **SINGLE**

Applies to: BatchQ, Combine, Delay, MatchQ, Node, ProcessQ, Queue, Resource, SeizeQ, Server.

Template shape type: ShapeBase – any shape will do.

The entity animation is displayed at the base position of the shape with the rotation of the shape. In case the shape is a polyline, the rotation is taken as the angle of its first segment. Not more than one entity at a time may be displayed by this animator.

## **SET**

Applies to: BatchQ, Delay, MatchQ, Node, ProcessQ, Queue, Resource, SeizeQ, Server.

Template shape type: ShapePoly – a polyline.

Animations of entities are displayed at the positions of points of the polyline. The number of simultaneously displayed entities is limited by the number of points in the polyline. The rotation of the entity animations is set to the (global) rotation angle of the polyline. A new entity is displayed at the first free position of the polyline.

## **BAG**

Applies to: BatchQ, Delay, MatchQ, Node, ProcessQ, Queue, Resource, SeizeQ, Server.

Template shape type: ShapeRect – a rectangle.

Animations of entities are displayed at random positions within the specified rectangle (irrespective of the other entities). The rotation of the entity animations is set to the rotation angle of the rectangle. This animator may display unlimited number of entities

simultaneously.

## **ARRANGED**

Applies to: BatchQ, Delay, MatchQ, Node, ProcessQ, Queue, Resource, SeizeQ, Server.

Template shape type: ShapeRect – a rectangle.

Animations of entities are displayed arranged in 2D array that fits the given rectangle. The rotation of the entity animations is set to the rotation angle of the rectangle. The maximum number of entities (the size of the array) is provided by the active object at the startup (usually this is capacity of the object) and may not be changed afterwards. A new entity is displayed at the first free position of the array.

## **MOVEMENT**

Applies to: Delay, ProcessQ(Delay).

Template shape type: ShapePoly or ShapeLine – a polyline or line.

Animations of entities are displayed moving along the polyline with constant speed, so that by the time they exit the Delay object, they cover the whole distance from the first to the last point of the polyline. As long as the delay times may vary for different entities, they may have different speeds. If the width of the polyline is greater than one, the entities will travel randomly shifted from the polyline axis. The rotation of the entity animations is governed by the angle of the current polyline segment. This animator may display unlimited number of entities simultaneously. The movement direction can be oriented either way with respect to the polyline direction.

## **QUEUE**

Applies to: BatchQ, MatchQ, Node, ProcessQ(Queue), Queue, SeizeQ.

Template shape type: ShapePoly – a polyline.

Animations of entities are displayed along the polyline in the same order they are ranked in the Queue object. The entity at the position 0 is displayed at the end point of the polyline. The distance between the two subsequent entity animations is set to the polyline length divided by the Queue capacity - 1 (as it was known at the startup). So, the entity at the

position 1 is displayed that distance behind the end point, etc. If the new entity is added somewhere in the middle of the Queue, the entity animations are shifted as needed. The rotation of the entity animations is governed by the angle of the current polyline segment. This animator may display unlimited number of entities simultaneously, however the distance is not reset if the capacity of the Queue changes dynamically. The queue head can be oriented either way with respect to the polyline direction.

## **CONVEYOR**

Applies to: Conveyor.

Template shape type: ShapePoly – a polyline.

Animations of entities are displayed along the polyline at the same positions moving with the same speeds as they have on the Conveyor, provided the Conveyor length is mapped to the full polyline length. The rotation of the entity animations is governed by the angle of the current polyline segment. This animator may display unlimited number of entities simultaneously. The movement direction can be oriented either way with respect to the polyline direction.

## **LANE**

Applies to: Lane.

Template shape type: ShapePoly – a polyline.

Animations of entities are displayed along the polyline at the same positions moving with the same speeds as they have on the Lane, provided the Lane length is mapped to the full polyline length. The rotation of the entity animations is governed by the angle of the current polyline segment. This animator may display unlimited number of entities simultaneously. The movement direction can be oriented either way with respect to the polyline direction.

## **AUTO**

In some cases it is possible to guess type of animation based on type of shape provided as animationShape. For instance, if rectangle is specified as animation for delay the BAG animator type will be assumed.

In a table below objects and corresponding types of animators are listed if AUTO is used.

Object	Type of shape	Animator	Description
Queue, Assembler, BatchQ, MatchQ, SeizeQ	ShapePoly (polyline)	QUEUE	
	ShapeRect (rectangle)	BAG	
Delay, Server	ShapeLine (line) or ShapePoly (polyline)	MOVEMENT	
	ShapeRect (rectangle)	BAG	
Node	animationShapeIdle: ShapePoly (polyline)	QUEUE	For node different animation templates can be specified: for idle transporters and for pending.
	animationShapeIdle: ShapeRect (rectangle)	BAG	
	animationShapePending: ShapePoly (polyline)	SET	
	animationShapePending: ShapeRect (rectangle)	BAG	
Resource	ShapePoly (polyline)	SET	
	ShapeRect (rectangle)	BAG	

ProcessQ encapsulates queue and delay. Types for them are animationTypeQ and animationTypeDelay correspondingly.

### 3.7 Advanced transportation

Advanced transportation is a set of objects for defining and using complex networks with resources. Entities can move along paths of the network and use resources located in that network.



## 3.7.1 Network and resources

Network consists of nodes and links between them which are generated automatically from an animation graphics. To define a network you draw a set of rectangles (one for each node) and a set of lines or polylines (one for each link) that have their end points inside the connected rectangles. Then you add all those shapes to a pivot and provide that pivot as a parameter of a Network object. The Network object will iterate automatically through the shapes and create the corresponding logical network. Names of the network locations are equal to names of graphical shapes.

Various resources are available across the network. The resources may be of three types: staff, portable and static. An example of a staff resource: nurse, fork-lift truck. An example of a portable resource is portable X-Ray device or car (in a railway system). Static resources are rooms and etc. Resources are defined using NetworkResource objects.

Each resource has its home location. A staff resource unit returns there if it gets idle. When a staff gets released at some location, and there is a pending request for such resource, it proceeds directly to another location where it is needed. Portable resources can be carried over the network by staff. An unmovable resource always resides at its home location.

The use of network resources is controlled by the Network object in centralized way. Resources are seized and released using special blocks (like previously described SeizeQ and Release). When entity enters such block a new order (request for resources) is submitted to the network. Network holds pending requests in the queues – one queue for each block. Requests in each queue are sorted according type of queue. The type of queue is selected in the block parameters. Thus a first order to be fulfilled is located in the head of the queue. When new entity arrives or resource becomes free the network selects one of the requests staying in the head of the queues. That selection is based on the network parameters – it can be longest waiting entity and etc. Requests for seizing of multiple resources are serviced as follows. Let somebody request a doctor and a nurse and there are nurses available and no doctors at this moment. Let a new request for nurse arrived from the same block. If it was placed according queue type behind the previous request that new request will wait until the first one will be fulfilled. If it has arrived from another block and it is first in a queue for that block it will be fulfilled.

### 3.7.1.1 Network

Defines network. Contains parameters for changing routing type, rules applied while

selecting requests for resources. NetworkResource blocks must be connected to resources\_port of the Network.

Network object is to be used as a parameter for a NetworkEnter object. NetworkEnter object adds entities to the specified locations of the network.

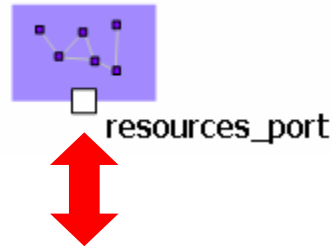


Figure 34 Network object

### Variables

Type	Name	Description
Entity	entity	The current entity.

### Functions

Return type	Name	Description
int	size( ShapeRect sr )	Returns number of entities in the specified node.
Entity	get( int i, ShapeRect sr )	Returns the entity at i-th position in the specified node.
void	setStatsEnabled ( boolean statsEnabled, ShapeRect sr )	Enables or disables statistics collecting for the specified node.
TimedDataSet	getStatsSize( ShapeRect sr )	Returns the statistics on the node size.
void	resetStats( ShapeRect sr )	Resets the statistics collecting for the node.

void	resetAllStats()	Resets utilization and working/idle time statistics for all resource units as well as size statistics for all network nodes.
Vector	getAllResources()	Returns vector of all resource units located in the network.
boolean	isResourceAvailable( Class res )	Returns true if at least one resource of specified type is available.

### Parameters

Type	Name	Default value	Description
ShapeGroup	pivot		Name of the animation pivot group which contains network elements (nodes and links).
boolean	networkIsVisible	false	If true then the elements that compose the network will be visible during model run.
	routingType	minimum route length	Which routing will be used in the network – minimum route length or routing based on minimum number of hops.
double	speed	100	Expression used to calculate speed with which entity moves in the network.
double	scale	1	Actual length of a segment is calculated as length of the segment in pixels multiplied by the scale.

	selectResourceRule		Rule applied when the entity must select one of the available resource units. Includes the following:  Closest resource  Least utilized resource  Longest idle resource
	selectEntityRule		Rule applied when resource unit becomes available and must select one of waiting entities. The following rules can be specified:  Longest waiting entity  Closest entity
	prioritiesWhileSelectE		If several entities are waiting for one resource and this parameter is true then i.e. the longest waiting entity of highest priority will be selected.

### Comments

The travel time in the network is calculated automatically based on the pixel length of the segments, scale and speed parameters. Speed parameter is reevaluated each time an entity enters segment (the current entity is accessible as entity). Thus speed may depend on a type of the entity, its contents and etc. I.e.:

```
( entity instanceof Nurse )? 100 : 50
```

### 3.7.1.2 NetworkResource

Describes a set of resources of a particular type. Network resources are very similar with the resources described in the chapter 3.4. They are objects of user-defined subclass of the Entity class.

Each resource unit has its home location in the network. It returns to that location if it

becomes idle and there are no requests for that resource pending.

Each NetworkResource object must be connected only to one Network object.

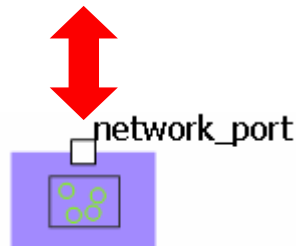


Figure 35 NetworkResource object

### Variables

Type	Name	Description
Entity	unit	The current resource unit.

### Functions

Return type	Name	Description
int	size()	The number of free resource units.
Entity	get( int i )	Returns the i-th unit.
int	getSeized()	Returns the number of seized resource units.
void	resetStats()	Resets the resource utilization statistics.

### Parameters

Type	Name	Default value	Description
	type	staff	What kind of resource those units are: staff, portable or static.
boolean	visible	false	For static resources you can specify is it visible or not.

ShapeBase	homeLocation		Home locations for the resource units (see comment for details).
int	quantity	5	The number of resource units of this type.
Class	newEntity	Entity.class	Type of resource (name of Message Class).
Schedule	schedule		Schedule object which defines working schedule and downtimes for the resource.
boolean	statsEnabled	false	Set this parameter to true to collect average utilization and idle/working times statistics for the resource units.
double	statsRefreshTime	infinity	By default utilization and working times are updated only when resource is seized and released. You can force periodic update of the statistics using that parameter.
code	onGenerate		Code executed when new unit is generated.
code	onSeizeUnit		Code executed for a unit when it is seized.
code	onReleaseUnit		Code executed for a unit when it is released.
code	onChangeState		Code executed for a unit when it changes its state (i.e. starts moving to home position).
code	onUpdateStats		Code executed after statistics was updated according statsRefreshTime parameter.

### Comments

Each resource unit has its own home position assigned to it when a model starts. You can

provide either rectangle (node of network) or polyline in home location parameter. In case of polyline the Resource object will iterate over all its points and will try to find nodes lying under the points of the polyline. First resource unit will be placed to the node under the first polyline point and etc. If a capacity is greater than the number of points in the polyline and last point of the polyline is reached then resources will continue to be placed from the first point. During model run if resource is released and there are no requests for that resource pending the resource unit will return to its home position (static resources remain in its home locations all time).

If statsEnabled parameter is true then for all resource units of that Resource object utilization and working/idle times statistics will be collected. That statistics is accessible via Entity class functions: `getInOperatingTime()`, `getInIdleTime()` and `getUtilization()`. InOperatingTime is a time which includes any time that resource unit moves to (from) an entity, moves with entity and is being used by an entity at a node. Statistics is collected for each unit individually.

Resource at any moment of time can be in one of the following states:

State	Value	Description
Idle	Network.IDLE	Available
Traveling to use	Network.TRAVEL_TO_USE	Resource is traveling to location from which it was called.
In use	Network.IN_USE	Resource unit has arrived and is being used by the entity.
Traveling to home	Network.TRAVEL_TO_HOME	Resource unit is going to home location.
Off shift	Network.OFFSHIFT	Isn't available due to failure.

Current state of the resource unit is accessible via function `getCurrentState()`.

### 3.7.2 Movement

After adding an entity to a network you can force it to move to other locations alone or with an escort. At any time node or location of the entity (reference to rectangle on animation) is accessible as `entity.location`.

### 3.7.2.1 NetworkEnter

Adds entities to the specified location in a network.



Figure 36 NetworkEnter

#### Variables

Type	Name	Description
Entity	entity	The current entity.

#### Functions

Return type	Name	Description
void	block()	Blocks the input port of the object.
void	unblock()	Unblocks the input port of the object.
boolean	blocked()	Returns true if the input is blocked, otherwise false.

#### Parameters

Type	Name	Default value	Description
ShapeRect	dest		Destination location to add entity to.
Network	network		Name of a network object which describes the network to add entity to.
code	onEnter		Code executed when the entity enters the object.
code	onExit		Code executed when the entity exits the object.



## Comments

Destination location is reevaluated for each entity entering the object. Thus it can depend on a state of the entity and etc.

### 3.7.2.2 NetworkExit

Removes entities from the network and outputs them via output port immediately.



Figure 37 NetworkExit

## Variables

Type	Name	Description
Entity	entity	The current entity.

## Functions

Return type	Name	Description
void	block()	Blocks the input port of the object.
void	unblock()	Unblocks the input port of the object.
boolean	blocked()	Returns true if the input is blocked, otherwise false.

## Parameters

Type	Name	Default value	Description
code	onEnter		Code executed when the entity enters the object.
code	onExit		Code executed when the entity exits the object.

## Comments

Entity is removed from the network to which it was previously added. All seized resources must be released before exiting from the network.

### 3.7.2.3 NetworkMoveTo

Moves entity from the current location to new location. As the destination location only rectangles which belong to the network where the entity is located can be used.

NetworkMoveTo block doesn't move portable and staff resources seized by the entity. To move with the previously seized resources NetworkMoveToWithQ block should be used.



Figure 38 NetworkMoveTo

## Variables

Type	Name	Description
Entity	entity	The current entity.

## Functions

Return type	Name	Description
void	block()	Blocks the input port of the object.
void	unblock()	Unblocks the input port of the object.
boolean	blocked()	Returns true if the input is blocked, otherwise false.

## Parameters

Type	Name	Default value	Description
ShapeRect	dest		Destination location to move to.

code	onEnter		Code executed when the entity enters the object.
code	onExit		Code executed when the entity exits the object.

### Comments

Destination location is reevaluated for each entity which enters the NetworkMoveTo object. Location may depend on a state of the entity or can be based on a distribution. I.e.:

```
(uniform() < 0.5) ? animation.rectangle1 : animation.rectangle2
```

If 'location of last seized static resource' is used as dest parameter then the entity will move to location of last seized static resource.

### 3.7.2.4 NetworkMoveToWithQ

Moves an entity to specified location in a network with an escort. The escort is a set of network resources of type staff.

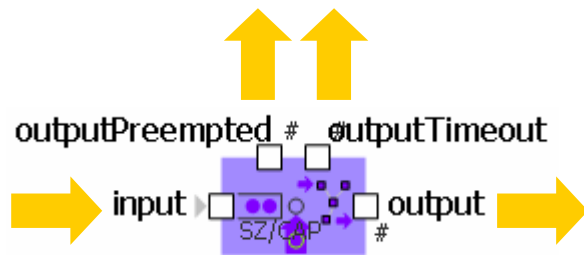


Figure 39 NetworkMoveToWithQ

### Variables

Type	Name	Description
Entity	entity	The current entity.

### Functions

Return type	Name	Description
-------------	------	-------------

void	block()	Blocks the input port of the object.
void	unblock()	Unblocks the input port of the object.
boolean	blocked()	Returns true if the input is blocked, otherwise false.
int	size()	Functions of the encapsulated queue, see Queue.
Entity	get( int i )	
Entity	remove( int i )	
TimedDataSet	getStatsSize()	
void	resetStats()	
int	sizeW	Functions of the encapsulated queue where entities wait for resource arrival.
Entity	getW( int i )	
TimedDataSet	getStatsSizeW()	
void	resetStatsW	

### Parameters

Type	Name	Default value	Description
Class[]	entities		List of entities to compose the escort.
ShapeRect	dest		Destination location to move to.
boolean	ownEscort	false	If true then entity will move to the new location with an escort from the previously seized resources.
boolean	releaseEscortAfterMove	true	If false then all escort units will remain seized after moving.
code	onEnter		Code executed when the entity enters the object.
code	onExit		Code executed when the entity exits the object.

code	onEnterWaiting		Code executed when the entity enters queue where entities wait for resource arrival.
code	onExitPreempted		Parameters of the encapsulated queue, see Queue.
code	onExitTimeout		
int	queueType	FIFO	
int	capacity		
int	entitesToAnimate		
boolean	preemption	false	
boolean	timeout	false	
code<double>	timeoutTime	infinity	
ShapeBase	animationShapeQ		
int	animationTypeQ	AUTO	
boolean	animationForwardQ	true	
boolean	statsEnabled	false	
boolean	statsEnabledW	false	
ShapeBase	animationShapeW		
int	animationTypeW	AUTO	
boolean	animationForwardW	true	

### Comments

If 'all seized resources' is used as a value of the entities parameter then the entity will move to the specified position will the all previously seized staff and portable resources. releaseEscortAfterMove parameter determines will be staff units released from the escort or not.

### 3.7.3 Working with network resources

Network resources can be accessed using special blocks. For different resource types

different blocks should be used. The way how that blocks operate is different. I.e. portable resources should be fetched by staff from their locations, but staff can move without assistance to entity by itself when request arrived. Thus:

Staff: NetworkCallQ, NetworkFree

Portable: NetworkFetchQ, NetworkReturn

Static: NetworkSeizeQ, NetworkRelease

At any moment of time a previously seized resources are accessible via the following variables of the entity:

Vector seizedStaff contains seized network resources of type staff, seizedPortable – portable resources and seizedStatic – static resources.

### 3.7.3.1 NetworkCallQ

Seizes the set of network resources of type staff and moves them to the specified location. Encapsulates two Queue objects; in the first queue entities wait for the resources to be available and in the second queue they wait for the resources arrival. Requests for the resources are serviced according queueType parameter. While waiting in the first queue an entity can exit due to timeout or can be preempted. The second queue has infinite size; timeout and preemption are unavailable.

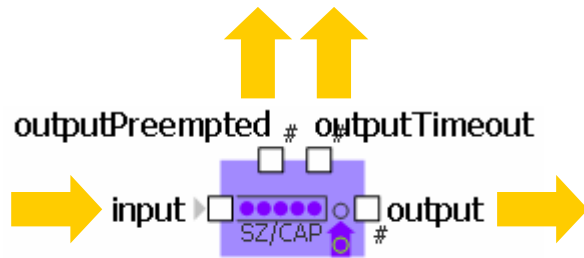


Figure 40 NetworkCallQ object

#### Variables

Type	Name	Description
------	------	-------------

Entity	entity	The current entity.
int	position	Variables of the encapsulated queue, see Queue.
double	timeoutValue	

### Functions

Return type	Name	Description	
void	block()	Blocks the input port of the object.	
void	unblock()	Unblocks the input port of the object.	
boolean	blocked()	Returns true if the input is blocked, otherwise false.	
int	size()	Functions of the encapsulated queue, see Queue.	
Entity	get( int i )		
Entity	remove( int i )		
boolean	canEnter()		
TimedDataSet	getStatsSize()		
void	resetStats()		
int	getExitedTimeout()		
int	getExitedPreempted()		
int	sizeW		Functions of the encapsulated queue where entities wait for resource arrival.
Entity	getW( int i )		
TimedDataSet	getStatsSizeW()		
void	resetStatsW		

### Parameters

Type	Name	Default value	Description
Class[]	entities		List of resources to call (list of resource types).

ShapeRect	dest		Destination point to which resources will be called. If 'location of entity' is selected then resource units will be called to current location of the entity. If null returned then resource units will remain in their positions.
code	onEnter		Parameters of the encapsulated queue, see Queue.
code	onExitPreempted		
code	onExitTimeout		
int	queueType	FIFO	
int	capacity		
int	entitiesToAnimate	all	
boolean	preemption	false	
boolean	timeout	false	
code<double>	timeoutTime	infinity	
ShapeBase	animationShapeQ		
int	animationTypeQ	AUTO	
boolean	animationForwardQ	true	
boolean	statsEnabled	false	
boolean	statsEnabledW	false	Parameters of encapsulated queue where entities wait for resource arrival
ShapeBase	animationShapeW		
int	animationTypeW	AUTO	
boolean	animationForwardW	true	
code	onEnterWaiting		Code executed when the entity enters the waiting for resource arrival queue.
code	onExit		Code executed when the entity exits the object.



## Comments

Only network resources of type staff can be called. In case of trying to call static of portable resources a runtime error will be signaled.

Resources are called from the network in which the entity is located. Parameter ‘entities’ is reevaluated each time when a new entity enters the object. Thus depending on the entity state or other conditions you can call different resources. For instance:

```
(entity.priority == 1) ? new Class[]{ MediumTruck.class } : new Class[]{
BigTruck.class }
```

If priority of entity is 1 then medium truck will be called and big truck otherwise.

Two resource units of the same type can be called by specifying the following string in the entities parameter: { Truck.class, Truck.class }.

After exiting the block all called resources are added to the seizedStaff vector of an entity. Calling means only that resource became seized. If you will use NetworkMoveTo to move the entity then the called resources will remain in its previous locations. To move with resources NetworkMoveToWithQ should be used.

### 3.7.3.2 NetworkFree

Frees network resources of type staff previously seized (called) by the entity. Resources to free are specified by enumerating their types. The release operation always takes zero time, so once entered the entity leaves the NetworkFree object immediately.



Figure 41 NetworkFree object

## Variables

Type	Name	Description
Entity	entity	The current entity.

## Functions

Return type	Name	Description
void	block()	Blocks the input port of the object.
void	unblock()	Unblocks the input port of the object.
boolean	blocked()	Returns true if the input is blocked, otherwise false.

## Parameters

Type	Name	Default value	Description
Class[]	entitiesToRelease	all	List of entities to free. If returns null, all non-attached resource units of type staff will be released.
code	onEnter		Code executed when the entity enters the object.
code	onExit		Code executed when the entity exits the object.

## Comments

If entitiesToRelease parameter returns null, the NetworkFree object will release all seized network resources of type staff. Those resources will be removed from a seizedStaff vector of the entity. If entity hasn't enough resources to free or tries to free resources of type other than staff a runtime error will be signaled.

Resources after releasing will be available in the network in which the entity is located.

### 3.7.3.3 NetworkSeizeQ

Seizes network resource units of type static. Behavior is the same as NetworkCallQ block but without queue where entities wait for the resource arrival. Seized static resources are added to seizedStatic vector of the entity.

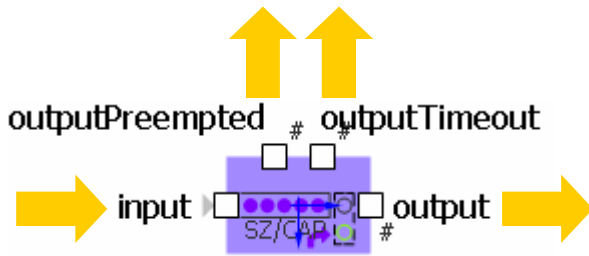


Figure 42 NetworkSeizeQ object

## Variables

Type	Name	Description
Entity	entity	The current entity.
int	position	Variables of the encapsulated queue, see Queue.
double	timeoutValue	

## Functions

Return type	Name	Description
void	block()	Blocks the input port of the object.
void	unblock()	Unblocks the input port of the object.
boolean	blocked()	Returns true if the input is blocked, otherwise false.
int	size()	Functions of the encapsulated queue, see Queue.
Entity	get( int i )	
Entity	remove( int i )	
boolean	canEnter()	
TimedDataSet	getStatsSize()	
void	resetStats()	

## Parameters

Type	Name	Default value	Description
------	------	---------------	-------------

Class	staticResource		Type of network resource of type static to seize.
code	onEnter		Parameters of the encapsulated queue, see Queue.
code	onExitPreempted		
code	onExitTimeout		
int	queueType	FIFO	
int	capacity		
Int	entitiesToAnimate		
boolean	preemption	false	
boolean	timeout	false	
code<double>	timeoutTime	infinity	
ShapeBase	animationShapeQ		
int	animationTypeQ	AUTO	
boolean	animationForwardQ	true	
boolean	statsEnabled	false	
code	onExit		Code executed when the entity exits the object.

### Comments

Only network resource of type static can be seized. In case of trying to seize other type of resources a runtime error will be signaled.

Seized resource is added to vector seizedStatic of the entity. Also reference to resource is stored in lastResource variable of the entity. You can use entity.lastResource.location as destination point for NetworkMoveTo object to move to location of last seized static resource.

### 3.7.3.4 NetworkRelease

Releases resources of type static previously seized by the entity. The resource is specified by

its type, after entity exits the block the resource will be removed from a `seizedStatic` vector of the entity.



Figure 43 NetworkRelease object

### Variables

Type	Name	Description
Entity	entity	The current entity.

### Functions

Return type	Name	Description
void	<code>block()</code>	Blocks the input port of the object.
void	<code>unblock()</code>	Unblocks the input port of the object.
boolean	<code>blocked()</code>	Returns true if the input is blocked, otherwise false.

### Parameters

Type	Name	Default value	Description
Class	<code>staticResource</code>		Type of static resource to release.
code	<code>onEnter</code>		Code executed when the entity enters the object.
code	<code>onExit</code>		Code executed when the entity exits the object.

### 3.7.3.5 NetworkFetchQ

Allows fetching portable network resources. Portable resources are fetched by staff to a current location of the entity. After fetching the escort can be freed or remain seized. Entities wait in queue for the portable resource to be available. Then the entities wait for the

escort, while it moves to portable resource and back to the entity. Portable resource is added to a vector seizedPortable of the entity.



Figure 44 NetworkFetchQ object

## Variables

Type	Name	Description
Entity	entity	The current entity.
int	position	Position of the entity in internal queue.

## Functions

Return type	Name	Description
void	block()	Blocks the input port of the object.
void	unblock()	Unblocks the input port of the object.
boolean	blocked()	Returns true if the input is blocked, otherwise false.
int	size()	Functions of the encapsulated queue, see Queue.
Entity	get( int i )	
Entity	remove( int i )	
boolean	canEnter()	
TimedDataSet	getStatsSize()	
void	resetStats()	

## Parameters

Type	Name	Default value	Description
------	------	---------------	-------------

Class	portableResource		Type of a portable network resource to fetch.
Class[]	escortEntities		List of network resources (of type staff) to fetch portable resource.
boolean	releaseEscortAfterMove	true	If false then all escort entities will remain seized after fetching.
code	onEnter		Parameters of the encapsulated queue, see Queue.
int	capacity	100	
int	entitesToAnimate	all	
boolean	statsEnabled	false	
ShapeBase	animationShapeQ		
int	animationTypeQ	AUTO	
boolean	animationForwardQ	true	
	queueTypeForResource	FIFO	
	queueTypeForEscort	FIFO	Type of queue for orders for escort entities.
code	onExit		Code executed when the entity exits the object.

### Comments

Unlike the NetworkCallQ block during all fetching operation the entity waits in one queue.

After resource has been fetched the escort can be released or can remain seized by the entity.

### 3.7.3.6 NetworkReturn

Returns portable network resource. Portable resources are returned by staff. An entity either can call a new staff to return the resource or can use a previously seized staff.

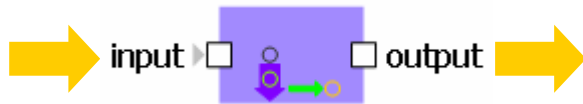


Figure 45 NetworkReturn object

## Variables

Type	Name	Description
Entity	entity	The current entity.

## Functions

Return type	Name	Description
void	block()	Blocks the input port of the object.
void	unblock()	Unblocks the input port of the object.
boolean	blocked()	Returns true if the input is blocked, otherwise false.
int	size()	Returns number of entities in the block.

## Parameters

Type	Name	Default value	Description
Class	portableResource		Type of portable network resource to return.
Class[]	escortEntities		List of network resources (type staff) to return portable resource.
boolean	releaseEscortAfterMove	true	If false then all escort entities will remain seized after returning.
boolean	ownEscort	false	If true then entity will use the previously seized staff resources to return the portable resource.



code	onEnter		Code executed when the entity enters the object.
code	onExit		Code executed when the entity exits the object.
	queueTypeForEscort	FIFO	Type of queue for orders for escort units.

### Comments

Entity does not wait while staff is returning the portable resource to its location if `releaseEscortAfterMove` is false.