AnyLogic™

Enterprise Library Tutorial

www.xjtek.com

XJ Technologies Company Ltd
AnyLogic@xjtek.com
http://www.xjtek.com/products/anylogic

Contents

Enterprise Library and Enterprise-Wide Discrete Simulation

AnyLogic™ provides the Enterprise Library, a discrete-event simulation library containing objects you can use to rapidly simulate complex discrete-events systems like:

- Manufacturing processes with detailed shop floor layout
- Simple and complex service systems (e.g. banks, airports, etc.)
- Business processes with activity based costing
- Logistics and supply chain models

The Enterprise Library allows you to create flexible models, collect basic and advanced statistics, and effectively visualize the process you are modeling to validate and present your model.

You can find models created with Enterprise Library in the AnyLogic™ examples pack, including:

- Airport Terminal
- Billing Department
- Beverage Production
- Packaging Line
- Warehouse And Flexible Assembly
- Multiple Call Centers

In this tutorial you will learn how to create models with the Enterprise Library in the fields of manufacturing and business processes (with activity-based costing).

Note that there are several reference files available for this model representing the milestones of the editing. You can use reference files if you experience any difficulties creating a model and you would like to compare your model with the reference file. Reference files are located under `Examples \ Enterprise Library Tutorial Models` folder, and you can use *Start Page* to open those examples. *Start Page* will appear automatically once you close the model you are editing.

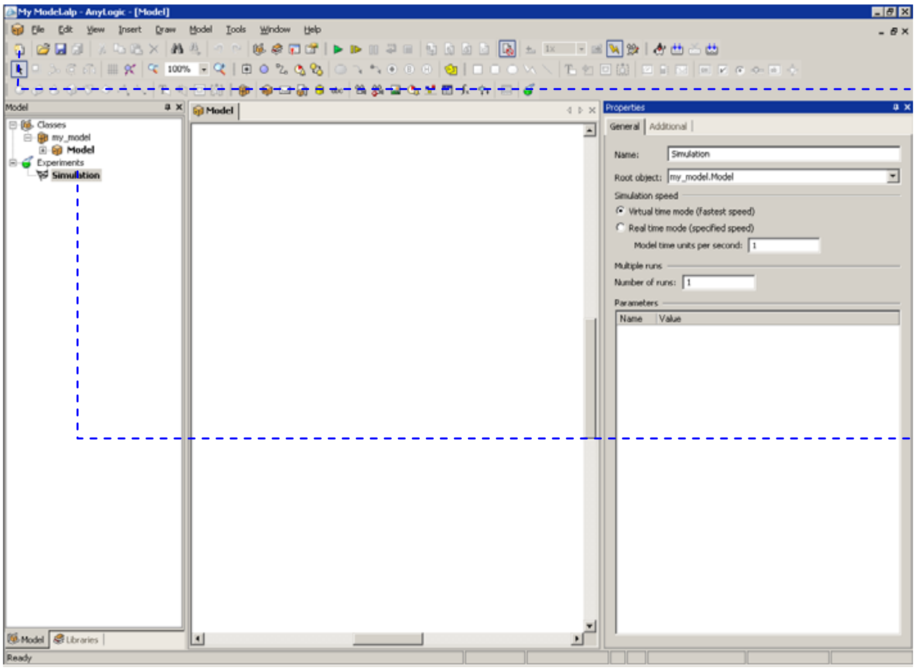
1. Getting Started with AnyLogic™

This section contains general information about creating models with Enterprise Library. You will learn how to create a new model and adjust its settings, how to use the library stencil and connect objects to build a model.

1.1 How to create a new AnyLogic™ model

In this tutorial, you will create several models. This section briefly explains how to create a new model and adjust model settings.

► How to create a new AnyLogic™ model



Name: Simulation

Root object: my_model.Model

Simulation speed

● Virtual time mode (fastest speed)

○ Real time mode (specified speed)

Model time units per second: 1

Multiple runs

Number of runs: 1

Parameters

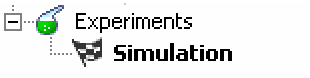
Name	Value

❶ Click the *New* button

❷ Adjust new model settings

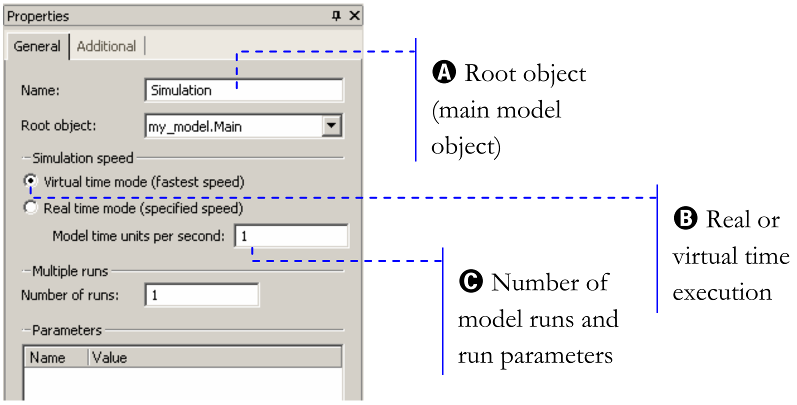
❶ Once you clicked the *New* button, a dialog box opens where you can choose the folder and give a file name for your new model.

❷ You can create several alternative model settings. A group of model settings is called an *experiment*, and experiments are displayed under the *Experiments* item in the model tree.



Experiments
Simulation

One experiment is created by default and named *Simulation*. Click the experiment in the tree, and adjust model settings using the *Properties* window.



Properties

General | Additional

Name: Simulation

Root object: my_model.Main

Simulation speed

● Virtual time mode (fastest speed)

○ Real time mode (specified speed)

Model time units per second: 1

Multiple runs

Number of runs: 1

Parameters

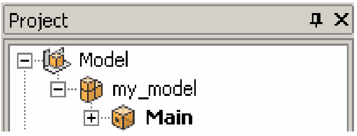
Name	Value

Ⓐ Root object (main model object)

Ⓑ Real or virtual time execution

Ⓒ Number of model runs and run parameters

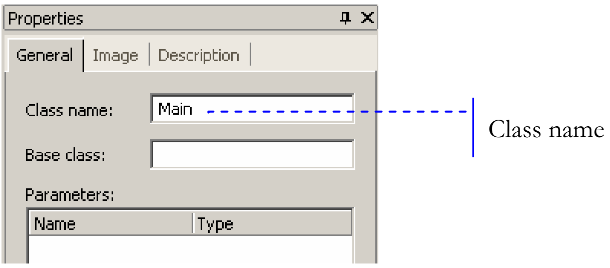
Ⓐ Select the class that will be started once you click the *Run* ▶ button. By default, this property is set to *Main*, a single blank class created automatically in the new model.



Project

Model
my_model
Main

If needed, you can rename the class. Click the class item in the *Model* tree, and adjust its settings using the *Properties* window.



Properties

General | Image | Description

Class name: Main

Base class:

Parameters:

Name	Type

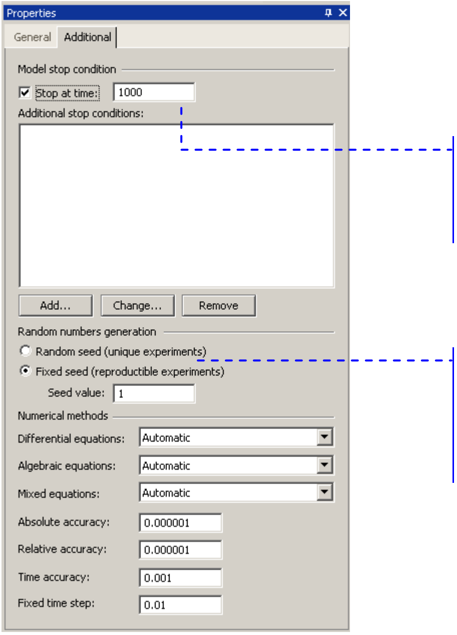
Class name

Ⓑ In real time mode, the model is executed regarding the physical time, i.e., you specify how many model time units will be executed in one second. Real time mode best fits to present model animation. In virtual time mode, the model is executed not regarding physical time—that is, as fast as possible. This mode best fits when you need to run the model for a long period of time.

➲ In this tutorial, we will mainly use *Real time mode* to observe animation.

Ⓒ A model can be automatically set up to be run several times to vary parameters. This way, you can observe system behavior under different conditions and automatically collect valuable statistics.

The additional simulation experiment properties tab allows you to control model execution.



Model stop condition

☑ Stop at time: 1000

Additional stop conditions:

Random numbers generation

○ Random seed (unique experiments)

● Fixed seed (reproductible experiments)

Seed value: 1

Numerical methods

Differential equations: Automatic

Algebraic equations: Automatic

Mixed equations: Automatic

Absolute accuracy: 0.000001

Relative accuracy: 0.000001

Time accuracy: 0.001

Fixed time step: 0.01

Ⓐ Model stop condition

Ⓑ Random number generator control

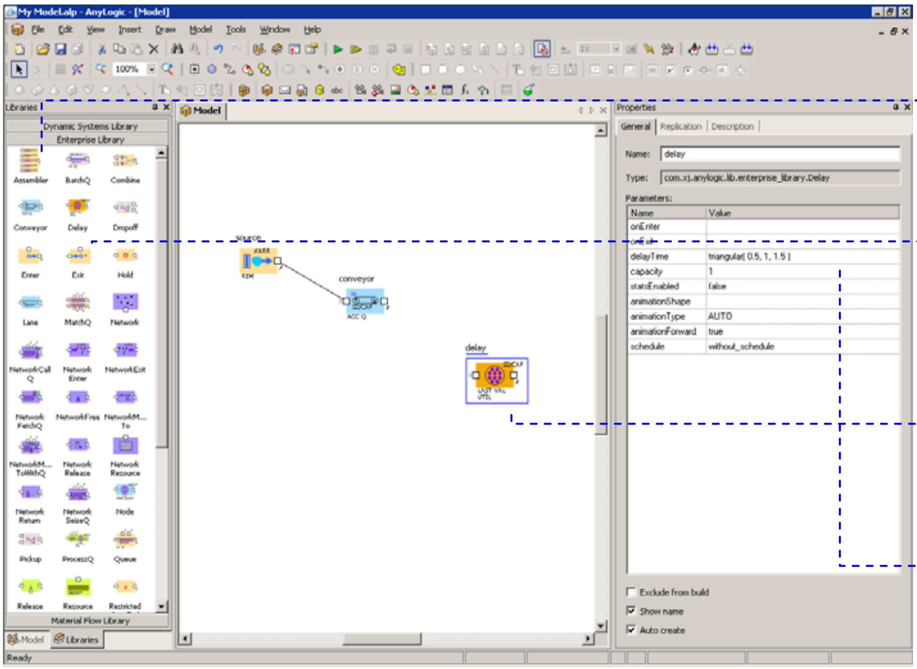
Ⓐ You can run the model infinitely or stop the model at the specific time. Also, you can specify advanced stop conditions, like statistics confidence interval, variable value condition, or any user-defined condition.

Ⓑ If you use random numbers in the model, you can choose to generate random numbers unique for each model run (this is useful to collect statistics for multiple runs) or to generate the same random numbers and thus obtain the same model execution across runs (this is useful to run the model several times and see its behavior).

1.2 How to use the Enterprise Library

You use the Enterprise Library by dragging the objects from the library stencil, setting custom properties for the objects, and connecting objects together.

► How to use Enterprise Library



Name: delay

Type: com.xj.anylogic.lib.enterprise_library.Delay

Parameters:

Name	Value
onEnter	
onExit	
delayTime	triangular(0.5, 1, 1.5)
capacity	1
statsEnabled	false
animationShape	
animationType	AUTO
animationForward	true
schedule	without_schedule

☐ Exclude from build

☑ Show name

☑ Auto create

❶ Enterprise Library stencil

❷ To add an object, drag it from the stencil with the mouse

❸ Drop the object where you want

❹ Change object properties

❶ Enterprise Library stencil shows all the objects in the library.

❷ You can click any object in the stencil and drag it onto the structure diagram.

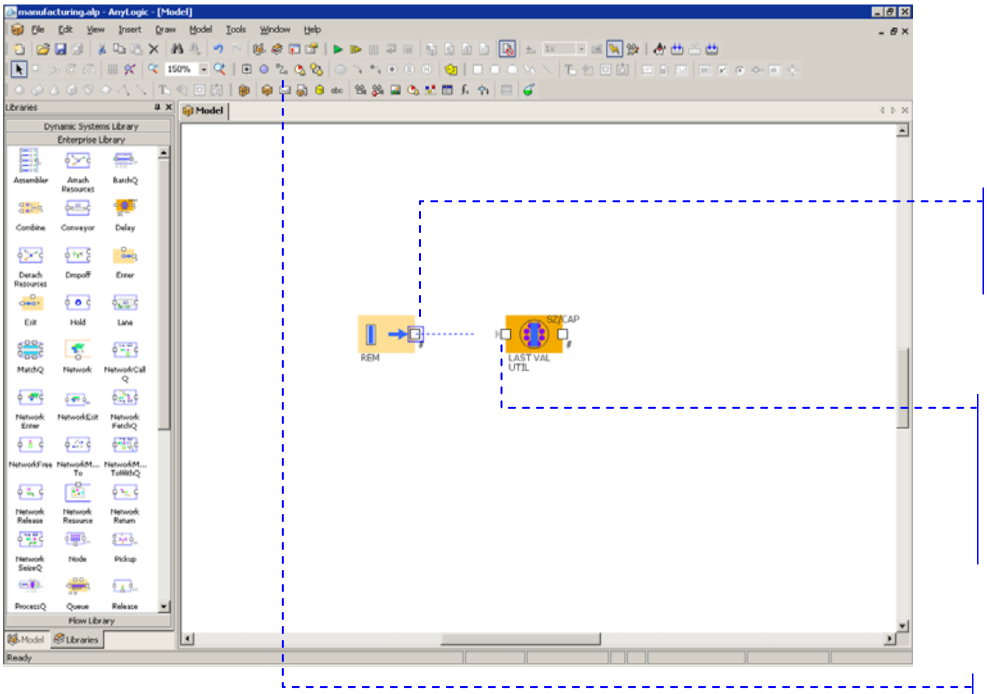
❸ Once you have dropped the object onto the structure diagram, it becomes selected and its properties are displayed in the *Properties* window.

❹ You can adjust object properties as your model requires. To adjust properties at a later time, click the object to select it and modify the properties you want.

1.3 How to connect objects

Objects need to communicate one with another, so you will need to connect one object with another. You will have two options: connect instantly by dragging first object's port onto second object's port, or by means of the special *Connector* tool.

► To connect objects



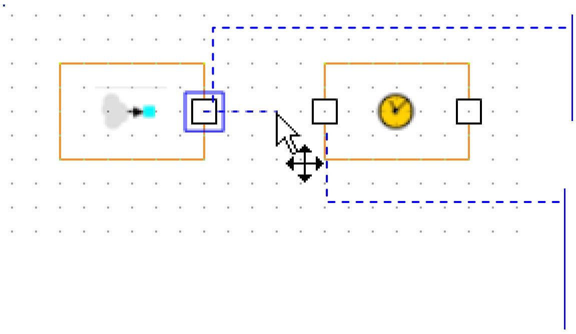
❶ Click the first port

❷ Drag it onto the second port. A connector appears.

❸ You can also use the *Connector* tool

❶ To connect two near-by objects use drag method:

❷



Ⓐ Click the first port to select it

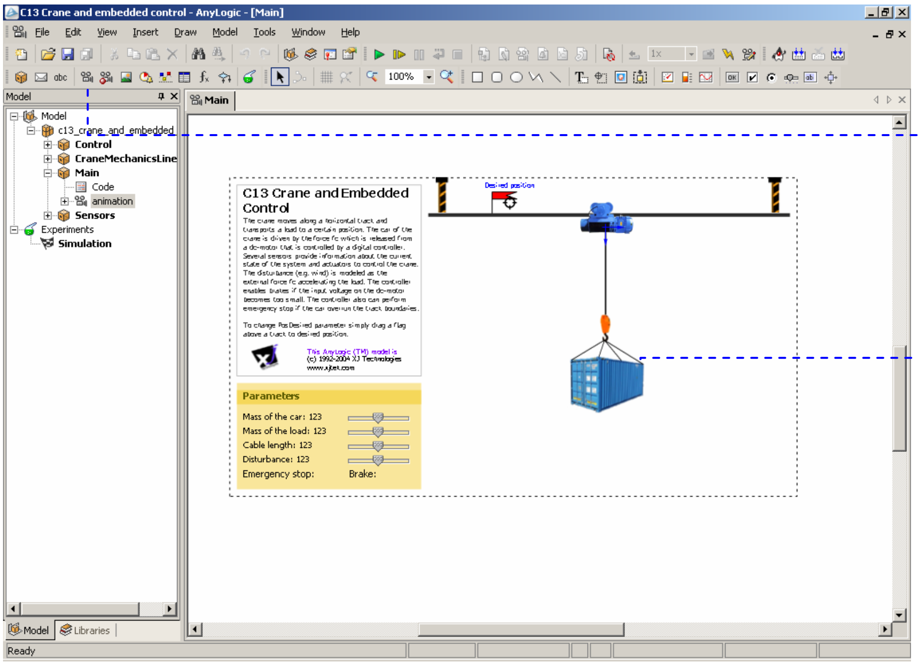
Ⓑ Drag the first port onto the second port to create a connector

❸ For long or curved connectors, click *Connector*, click the first port, and then click the second port.

1.4 How to create a drawing

For each model you can create an animation to visually represent your model. You can create any animation you want. To create an animation you need to draw it using an animation diagram.

► To create a drawing



C13 Crane and Embedded Control

The crane moves along a horizontal track and transports a load to a certain position. The car of the crane is driven by the force fc which is released from a dc-motor that is controlled by a digital controller. Several sensors provide information about the current state of the system and actuators to control the crane. The disturbance (e.g. wind) is modeled as the external force fc accelerating the load. The controller enables brakes if the input voltage on the dc-motor becomes too small. The controller also can perform emergency stop if the car overrun the track boundaries.

To change PosDesired parameter simply drag a flag above a track to desired position.

This AnyLogic (TM) model is (c) 1992-2004 XJ Technologies www.xjtek.com

Parameters

Mass of the car: 123
Mass of the load: 123
Cable length: 123
Disturbance: 123
Emergency stop: Brake:

❶ Create a new animation

❷ Draw shapes you want using drawing tools

❶ To create a new animation for the class, click the *New Animation* 品 toolbar button.

❷ You can draw geometric shapes (lines, circles, rectangles etc.), add images, user-controllable controls (buttons, radio buttons, checkboxes etc.), indicators (charts, bar and arc indicators, text). All the shapes can change the appearance at run time.

2. Bank Department Model

In this section we will create a simple service system of a bank department, consisting of an automatic teller machine and teller lines. ATM provides people with a quick self-service for cash. More complex transactions, e.g. paying bills, are completed by tellers, allowing customers more time without inconveniencing those customers looking for quick cash. We will apply activity-based costing in the example to see how much it costs to serve a customer for the company, and what parts are wait cost and service cost.

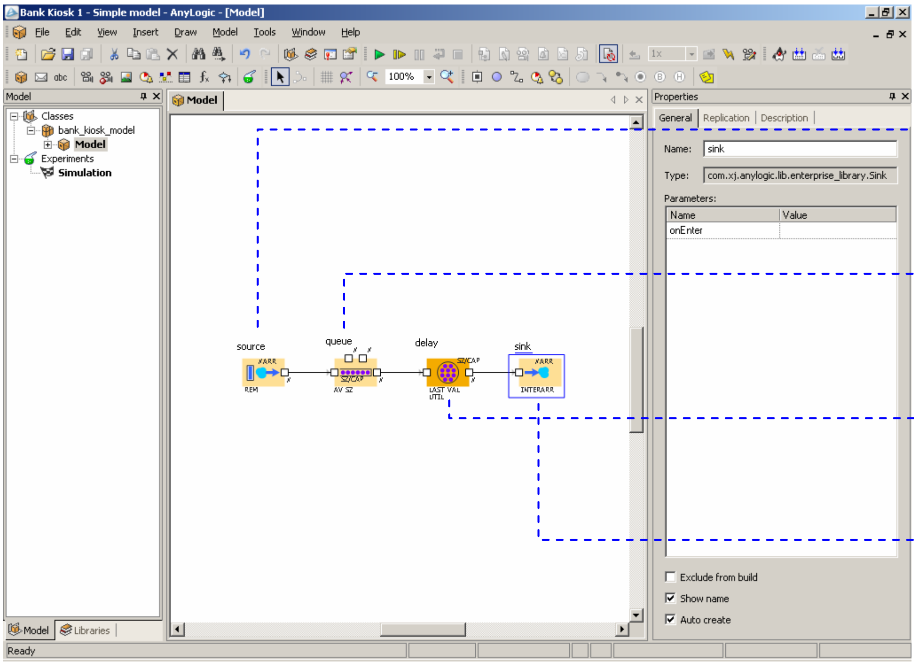
2.1 Creating a new project

Create a new model as described in Section 1.1, "How to create a new AnyLogic™ model". Rename `Main` class to `Model`. Using *Simulation* experiment, specify that the model is executed in real time mode and one model time unit will be executed in one second. In this model we will consider a model time unit as one minute of a bank department lifetime.

2.2 Creating a flowchart

Now we will create the flowchart of the system consisting of the ATM only. We will create and connect several objects as shown below.

► Create the flowchart



Name: sink

Type: com.xj.anylogic.lib.enterprise_library.Sink

Parameters:

Name	Value
onEnter	

☐ Exclude from build

☑ Show name

☑ Auto create

❶ Source

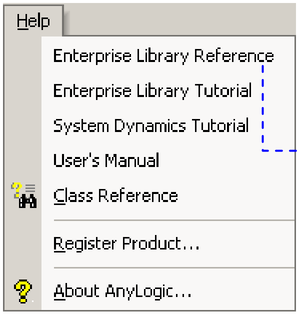
❷ Queue

❸ Delay

❹ Sink

❶ Add Source object. Source object generates entities with the specified interarrival time. In our example, it models customer arrival.

Please refer to *Enterprise Library Reference Guide* for the description of all the Enterprise Library objects. You can find there all object functions and their parameters.



Ⓐ Invokes Enterprise Library Reference Guide

Ⓐ To invoke *AnyLogic™ Enterprise Library Reference Guide*, click the corresponding item on the *Help* menu.

❷ Add Queue object to model a queue of customers waiting for the moment they can be served.

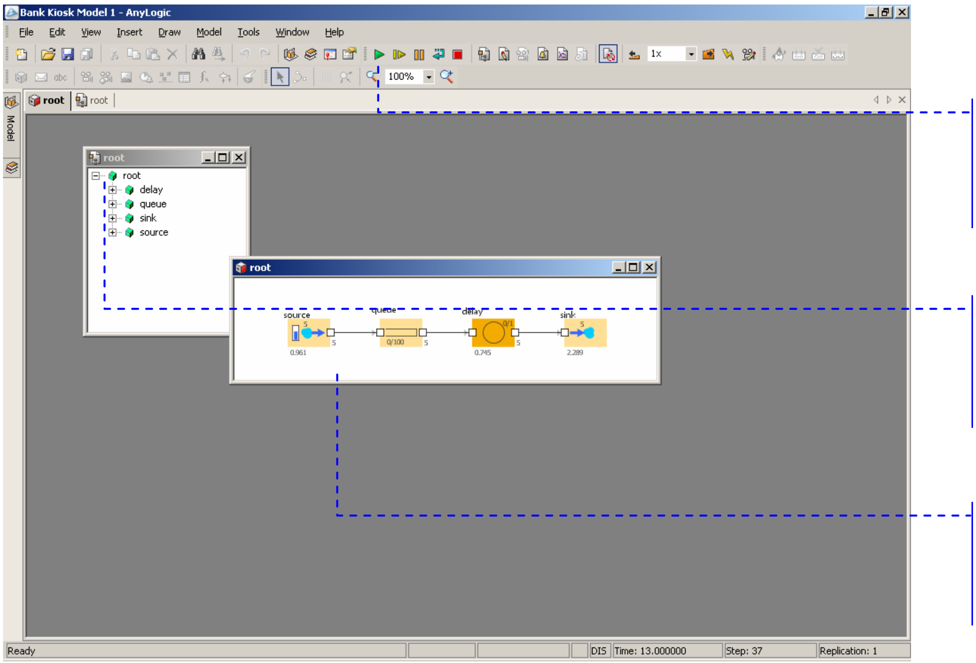
❸ Add Delay object. The Delay object models the ATM that is spending some time serving the customer.

❹ Add Sink object. This object indicates the end of the flowchart.

2.3 Running the model and observing behavior

We are ready to run the model created. Each model created with Enterprise Library instantly has animated flowchart where you can see detailed current object status, for example queue size, number of entities left and so on – completely in graphics!

▶ Run the model and open animated flowchart



❶ Click the *Run* ▶ toolbar button

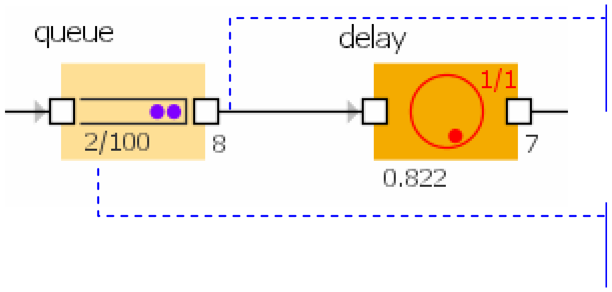
❷ Double-click `root` in the objects window

❸ Animated structure diagram window opens

❶ Click the *Run* ▶ toolbar button. AnyLogic^{TM} switches to run-time layout.

❷ In the *Objects* window appeared double-click the topmost item in the tree. If Objects window is not present, open it by clicking the *Model Root Object* toolbar button.

❸ Once you double-click the topmost item, the window opens showing you the animated flowchart of your model. You can instantly see how many entities exited a queue, or are stored there, etc.



❶ 8 entities exited the queue

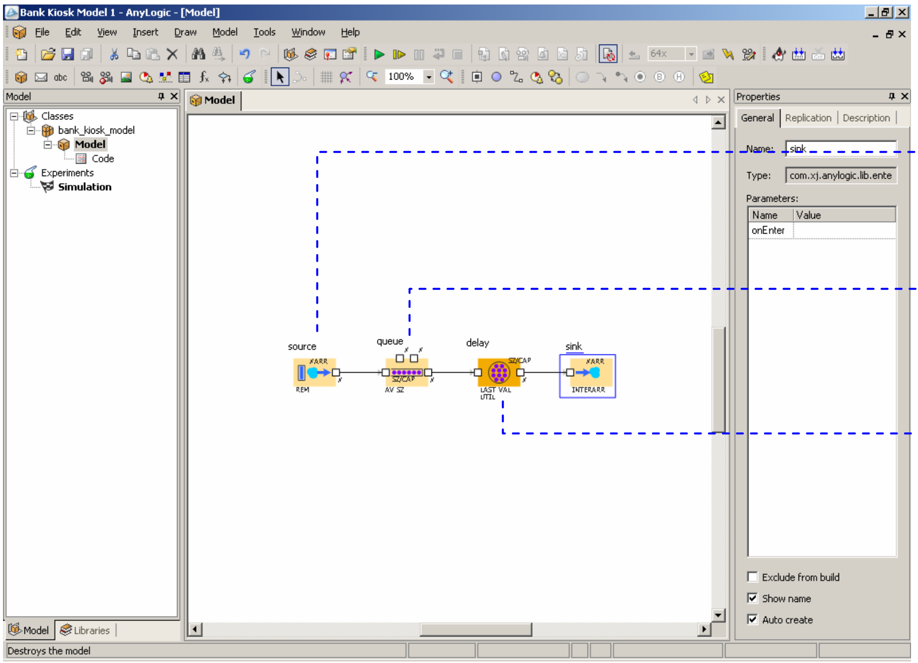
❷ 2 entities are stored.

The reference model for this point is Examples \ Enterprise Library Tutorial Models \ Bank Department 1 - Simple model.alp.

2.4 Defining model data

Now we will define the data of our model, adjusting the properties of the objects created.

► Set up the object properties



Name: sink

Type: com.xj.anylogic.lib.ente

Parameters:

Name	Value
onEnter	

□ Exclude from build

☑ Show name

☑ Auto create

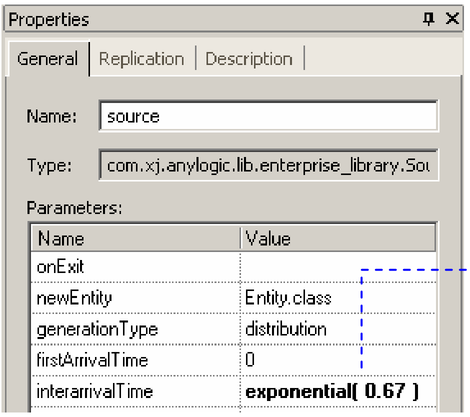
Destroys the model

❶ Source

❷ Queue

❸ Delay

❶ Specify how often customers arrive. You specify interarrival time in the `interarrivalTime` object property:



Name: source

Type: com.xj.anylogic.lib.enterprise_library.Sou

Parameters:

Name	Value
onExit	
newEntity	Entity.class
generationType	distribution
firstArrivalTime	0
interarrivalTime	**exponential(0.67)**

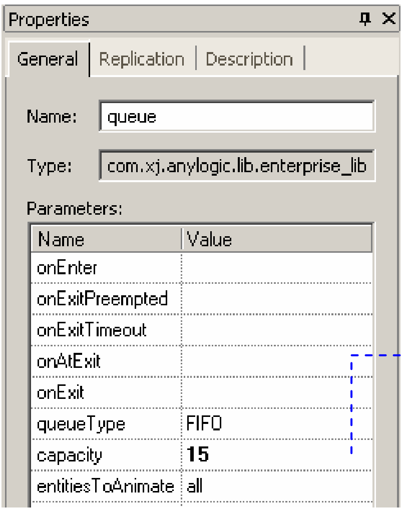
🅐 Type `exponential( 0.67 )` for interarrival time

🅐 Specify that the interarrival time is an exponentially distributed value with the mean of 1.5 time units. Note that `exponential()` function accepts rate as the argument, so we use the rate of 0.67.

The `exponential()` function is the standard AnyLogic™ random number generator. AnyLogic™ provides also other random number distributions, like normal, uniform, triangular etc. Please refer to *User's Manual* for the description of all the random number generators. You can also see all the generator functions and their parameters in *AnyLogic™ Class Reference*, see the `Func` page.

To invoke *AnyLogic™ User's Manual* or *Class Reference*, click these items on the *Help* menu.

❷ Set up the following properties:



Name: queue

Type: com.xj.anylogic.lib.enterprise_lib

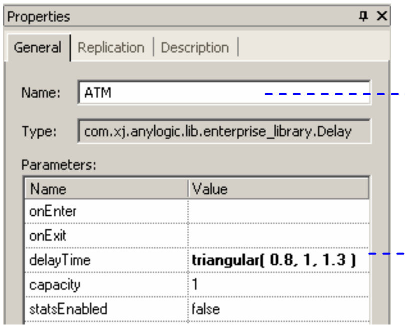
Parameters:

Name	Value
onEnter	
onExitPreempted	
onExitTimeout	
onAtExit	
onExit	
queueType	FIFO
capacity	**15**
entitiesToAnimate	all

🅐 Set queue capacity to 15 entities

🅐 At most 15 customers will wait in a queue.

❸ Change the Delay object properties:



Properties

General | Replication | Description

Name: ATM

Type: com.xj.anylogic.lib.enterprise_library.Delay

Parameters:

Name	Value
onEnter	
onExit	
delayTime	**triangular(0.8, 1, 1.3)**
capacity	1
statsEnabled	false

Ⓐ Name the object `ATM`

Ⓑ Type `triangular(0.8, 1, 1.3)` for delay time

Ⓑ A transaction takes about 1 minute. Specify that processing time is triangularly distributed with mean value of 1, min of 0.8 and max value of 1.3 time unit.

Now you can run the model by clicking the *Run* ▶ button and observe its behavior.

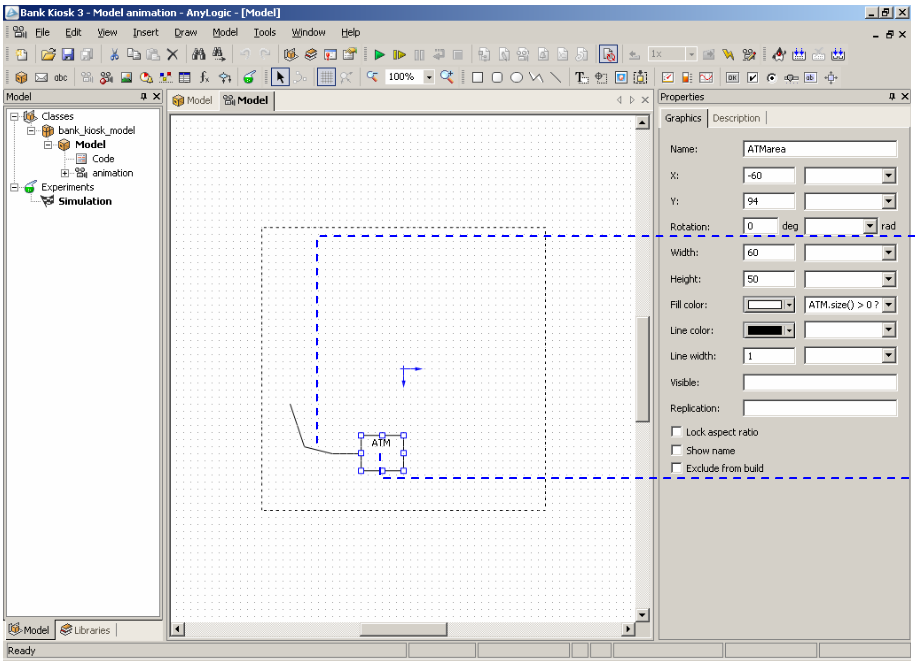
➘ The reference model for this point is Examples \ Enterprise Library Tutorial Models \ Bank Department 2 - Model data.alp.

2.5 Creating a model animation

Although the flowchart is animated, you may want to see the actual bank department layout animated. That is also possible! Now we will draw the layout consisting of the ATM and a queue. First, we will create the animation diagram, and second, we will animate it.

Now we will draw the layout of our bank. You draw the layout in AnyLogic™ using the animation editor. However, if you have existing image of the layout, you can simply import this picture as the bank layout instead of drawing it by yourself.

► Draw queue and ATM


Bank Kiosk 3 - Model animation - AnyLogic - [Model]
File Edit View Insert Draw Model Tools Window Help
Model
Classes
bank_kiosk_model
Model
Code
animation
Experiments
Simulation
Model
Libraries
ATM
Properties
Ready

Graphics | Description

Name: ATMarea

X: -60

Y: 94

Rotation: 0 deg rad

Width: 60

Height: 50

Fill color: ATM.size() > 0 ?

Line color:

Line width: 1

Visible:

Replication:

☐ Lock aspect ratio

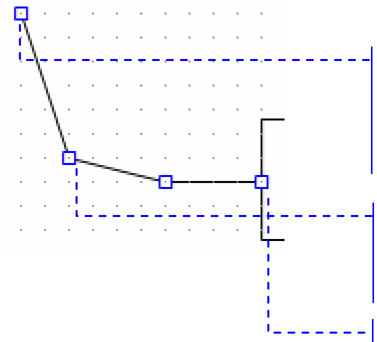
☐ Show name

☐ Exclude from build

❶ Draw queue path

❷ Draw ATM rectangle

❶ Draw the queue path using the *Polyline* drawing tool.



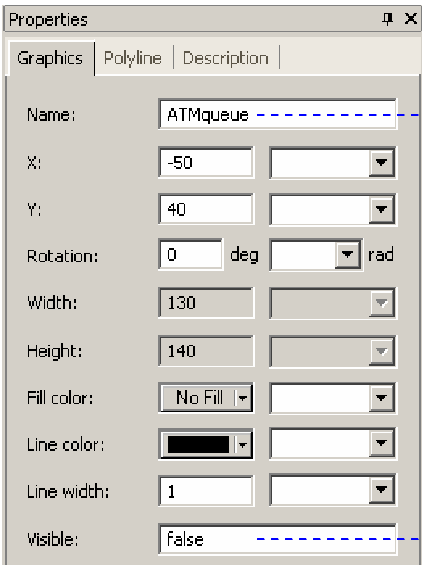
Ⓐ Click *Polyline* toolbar button and click this first point

Ⓑ Click several intermediate points

Ⓒ Double-click the last point

➲ The starting point—that is, the point you click first to draw a polyline—is important. By default, entities will be moving from the point you draw first to the point you draw last.[1]

Set up some properties for the queue path drawn:



Properties

Graphics | Polyline | Description

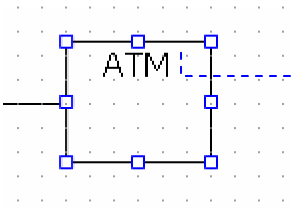
Name:	ATMqueue		
X:	-50		
Y:	40		
Rotation:	0	deg	rad
Width:	130		
Height:	140		
Fill color:	No Fill		
Line color:	■		
Line width:	1		
Visible:	false		

🅐 Specify `ATMqueue` name for the path

🅑 Specify that the polyline is invisible

🅑 Hide the queue path. To do this, click the path and type `false` in its *Visible* property.

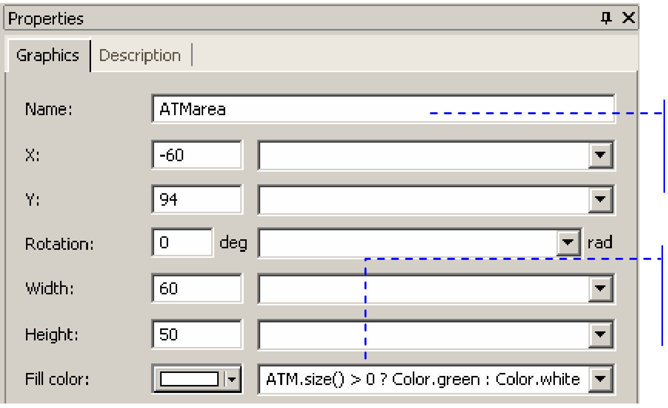
❷ Draw ATM rectangle:



🅐 Click *Rectangle* □ and draw a rectangle depicting the ATM

Set up some properties for the ATM drawn:

[1] However, you can set `animationForward` property of the flowchart object to `false` to move entities in the opposite direction.



Properties

Graphics | Description

Name: ATMarea

X: -60

Y: 94

Rotation: 0 deg rad

Width: 60

Height: 50

Fill color: ATM.size() > 0 ? Color.green : Color.white

Ⓐ Name the rectangle `ATMarea`

Ⓑ Set run-time color for the shape

Ⓑ Type run-time color expression:

```
ATM.size() > 0 ? Color.green : Color.white
```

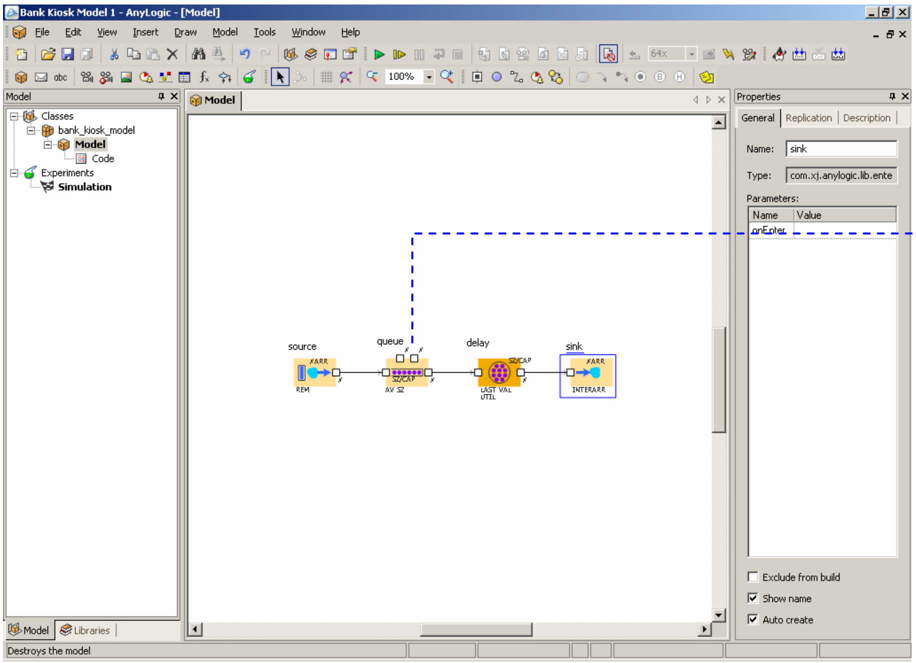
Note that `ATM` is the name of the Delay object we created. The expression determines the rectangle color at run time. The `size()` function returns the number of entities currently being processed. The color will be green, if a customer is served at this time, and white otherwise.

`Color` is the standard Java™ class, containing some predefined colors like black, blue, cyan, magenta, red and so on, and also allowing you to create any custom color.[2]

[2] To see the list of predefined colors, as well as the methods of the Color class that allows you to construct custom colors, open http://java.sun.com/j2se/1.4.2/docs/api/java/awt/Color.html or see your locally installed Java™ documentation.

Now we are ready to animate the layout. This is done by setting up the animation properties for the logic flowchart objects.

► Set up the animation properties



Name: sink

Type: com.xj.anylogic.lib.ente

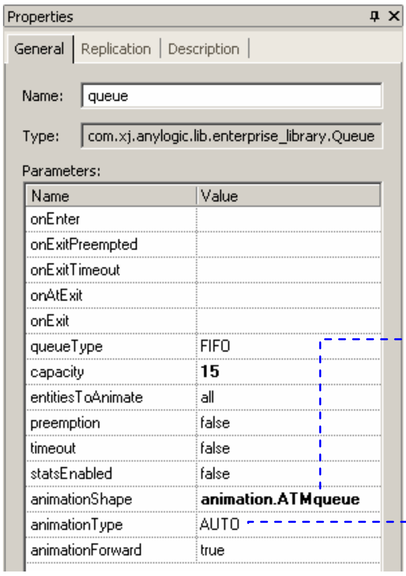
Parameters:

Name	Value
onEnter	

☐ Exclude from build
☑ Show name
☑ Auto create

❶ Animate the queue

❶ Set up queue animation. To animate a queue, just specify the queue path:



Name: queue

Type: com.xj.anylogic.lib.enterprise_library.Queue

Parameters:

Name	Value
onEnter	
onExitPreempted	
onExitTimeout	
onAtExit	
onExit	
queueType	FIFO
capacity	**15**
entitiesToAnimate	all
preemption	false
timeout	false
statsEnabled	false
animationShape	**animation.ATMqueue**
animationType	AUTO
animationForward	true

Ⓐ Set up the path name

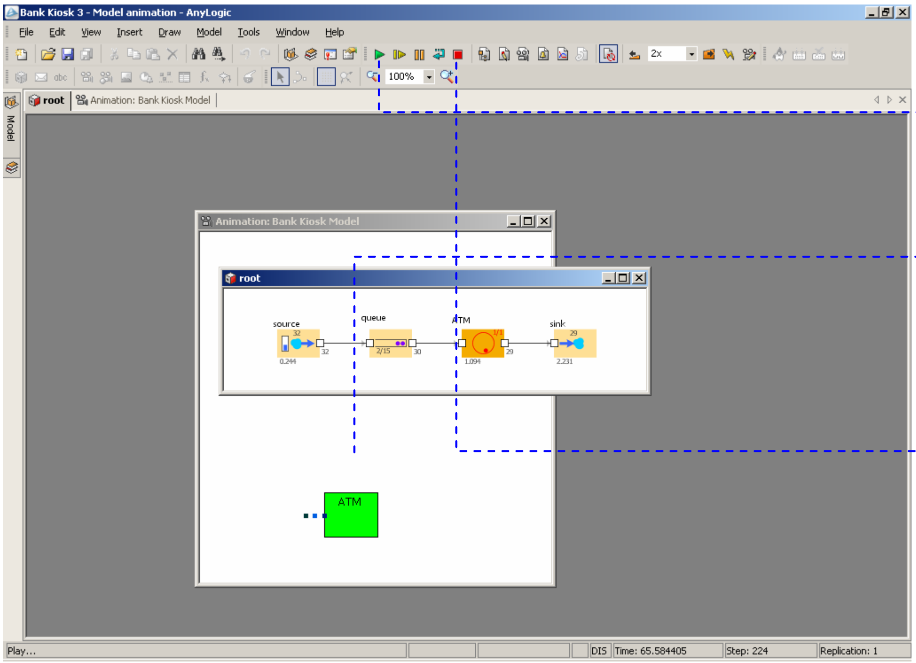
Ⓑ Specify the animation style

Ⓐ Choose `animation.ATMqueue` from the drop-down list.

Ⓑ Many objects of the Enterprise Library support several animation styles. For example, a queue can show its contents like a line of items, a disordered heap of items, arranged items, and so on. For more information on animation styles please refer to *Enterprise Library Reference Guide*. AUTO animation style detects the style automatically based on the shape you provided (for instance, a rectangle or a polyline). In this case, the queue will be animated as a line of customers waiting to be served.

Now you can run the model and observe its behavior.

▶ Run the model


Bank Kiosk 3 - Model animation - AnyLogic
File Edit View Insert Draw Model Tools Window Help
root
Animation: Bank Kiosk Model
source
queue
ATM
sink
ATM
Play...
DIS
Time: 65.584405
Step: 224
Replication: 1

❶ Click the *Run* button

❷ Model starts executing and animation

❸ Click *Stop* to stop execution and edit the model

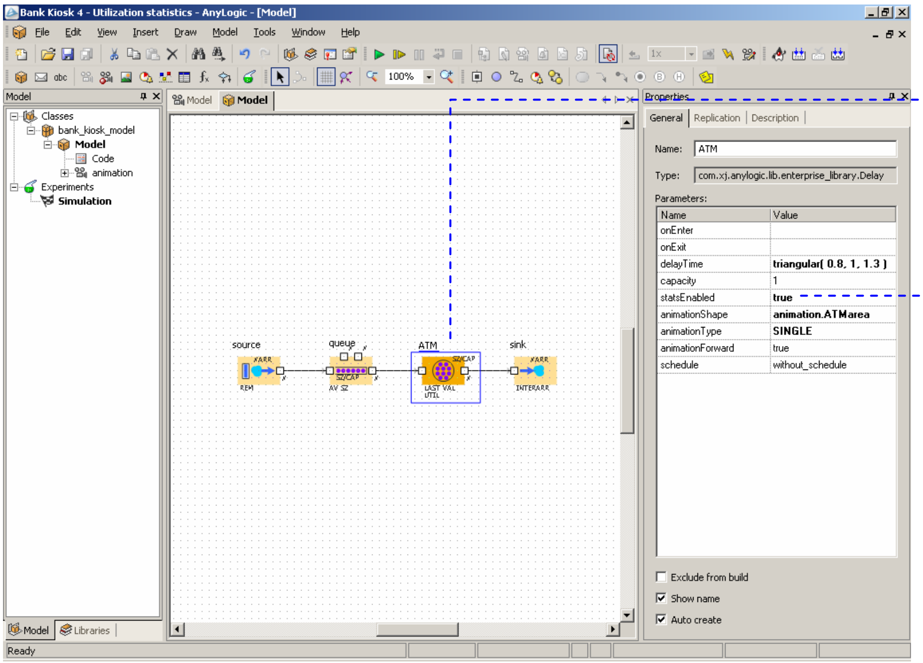
❷ Having started the model, you will see the animation window. Note that when the ATM station is serving a customer, it becomes green, and when it is idle, it is white. A statistics about ATM utilization is probably worth collecting.

The reference model for this point is Examples \ Enterprise Library Tutorial Models \ Bank Department 3 - Model animation.alp.

2.6 Collecting utilization statistics

With AnyLogic™, you can collect complicated statistics whenever you need them. The objects of the Enterprise Library are already capable of collecting the basic statistics. All you need is to turn the statistics collection for the object on, as it is disabled by default to speed up the model execution.

► Turn on statistics collection



Name: ATM

Type: com.xj.anylogic.lib.enterprise_library.Delay

Parameters:

Name	Value
onEnter	
onExit	
delayTime	triangular(0.8, 1, 1.3)
capacity	1
statsEnabled	true
animationShape	animation.ATMarea
animationType	SINGLE
animationForward	true
schedule	without_schedule

□ Exclude from build

☑ Show name

☑ Auto create

❶ Click the ATM object

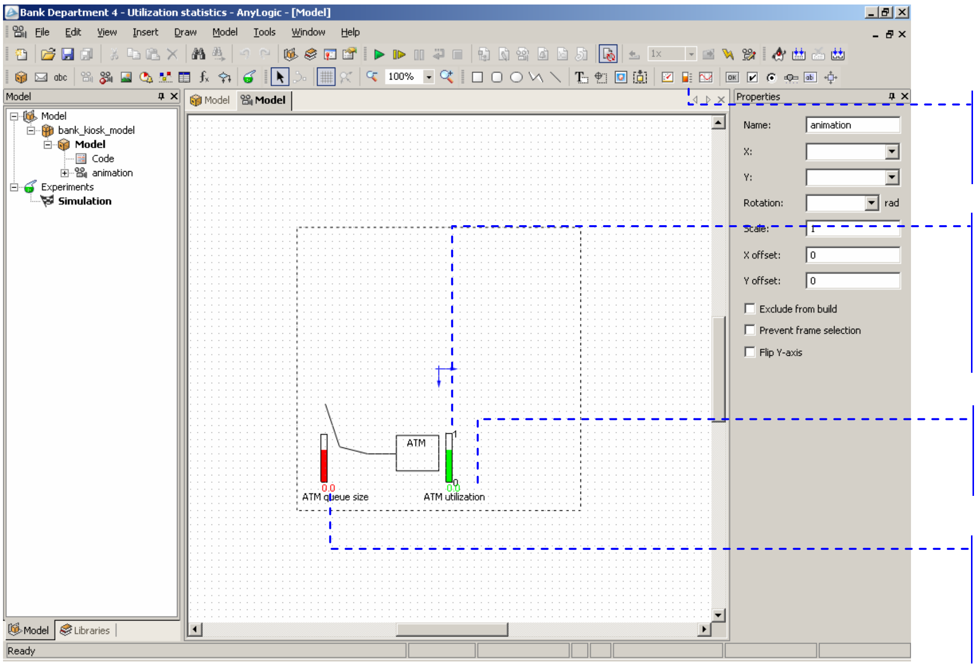
❷ Turn on statistics collection

❷ To turn on statistics collection for the object, select `true` value for its `statsEnabled` property.

You can view the statistics collected as charts or numerical values. You can also plot charts and display numerical values directly in the animation.

We will draw a utilization indicator in the animation.

▶ Add ATM utilization indicator



Name: animation

Rotation: rad

X offset: 0

Y offset: 0

□ Exclude from build

□ Prevent frame selection

□ Flip Y-axis

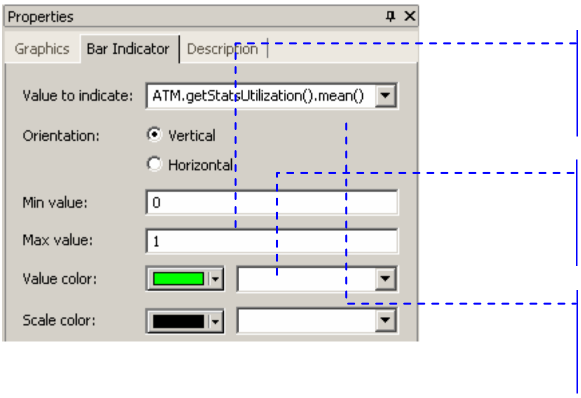
❶ Click *Bar Indicator*

❷ Click to place new indicator. Adjust its size and properties

❸ Add text label

❹ Place a bar indicator for the queue size

❷ Resize the indicator as necessary. Then set up the following properties:



Value to indicate: ATM.getStatsUtilization().mean()

Orientation: ● Vertical ○ Horizontal

Min value: 0

Max value: 1

Value color: 🟩

Scale color: ■

Ⓐ Set min value to 0

Ⓑ Set max value to 1

Ⓒ Set up the value to indicate

Ⓒ Set up the following expression as the value to indicate:

```
ATM.getStatsUtilization().mean()
```

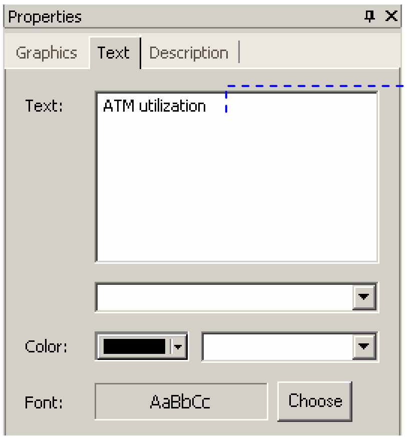
Here `ATM` is the name of the Delay object we created. Each Delay object has a function that returns its statistics. This function is called `getStatsUtilization()`.

The `mean()` is the function that returns the mean value measured. You can use other methods to get statistical values, such as `min()`

and `max()`. You can find the list of all the methods in *Class Reference*, see `DataSet` page.

For more information about the statistics measured by a Delay object, please refer to *Enterprise Library Reference Guide.*

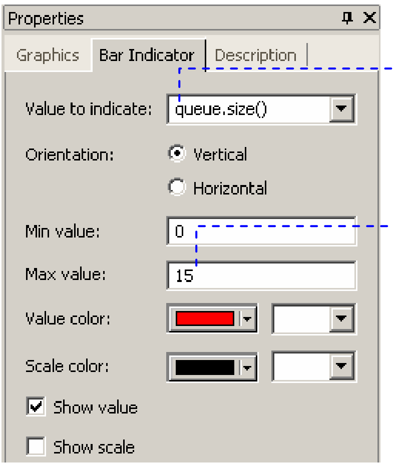
❸ Add a text label by clicking the *Text* T toolbar button and clicking under the indicator created to place it.



Text: ATM utilization

Ⓐ Type the text to be displayed

❹ Add a bar indicator to observe the current queue size.



Value to indicate: queue.size()

Orientation: ● Vertical ○ Horizontal

Min value: 0

Max value: 15

☑ Show value

□ Show scale

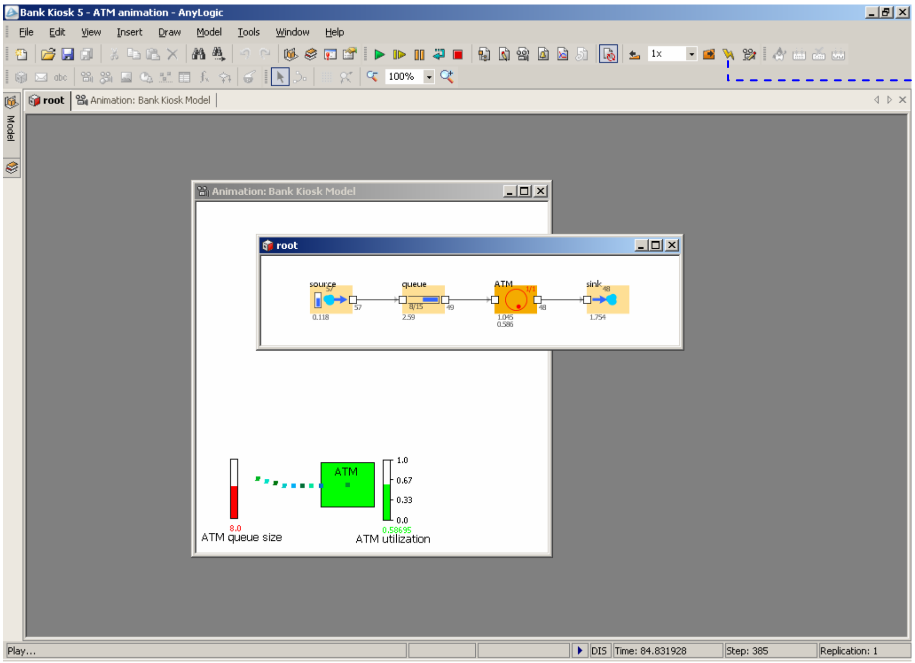
Ⓐ Set up `queue.size()` as the value to indicate

Ⓑ Set max value to 15

Ⓐ The size() function of a Queue object returns the current number of entities in the queue.

Ⓑ Since the queue is set up to contain at most 15 entities, we set the bar indicator max value to 15 as well.

► Play with the model



❶ To speed up the simulation switch to virtual time mode

❶ Switching to virtual time mode allows you to view simulation run at its maximum speed. Therefore, you can simulate a long period of time.

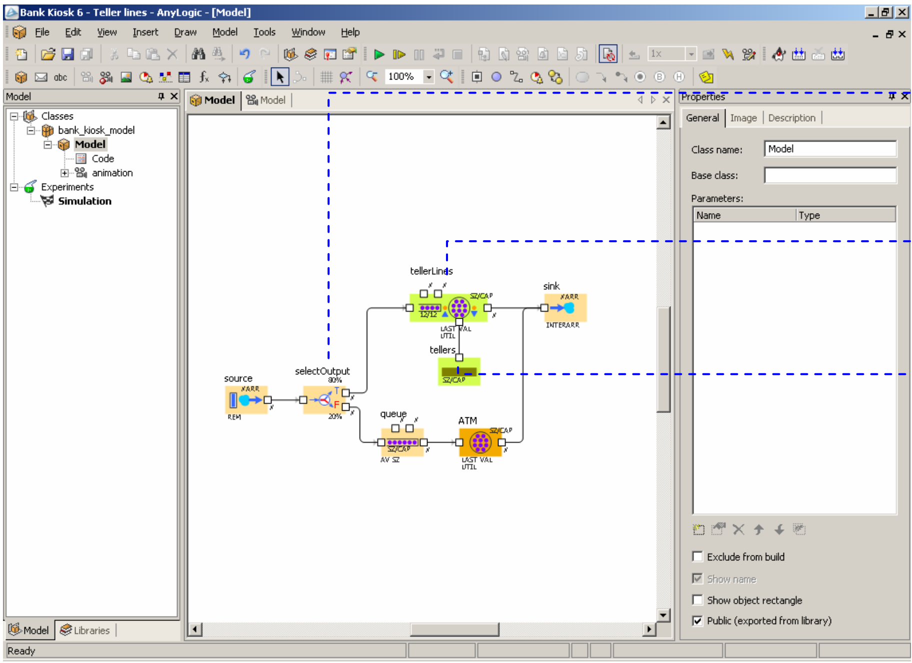
The reference model for this point is Examples \ Enterprise Library Tutorial Models \ Bank Department 4 - Utilization statistics.alp.

This model demonstrated the basics of the Enterprise Library. Now we are ready to create more advanced model.

2.7 Adding teller lines logic

Now we will create another part of the system by modeling how the customers are served by the tellers. We can model tellers using delays in the same way as we modeled ATM. However, modeling tellers using resources is much more convenient. Resource is a special unit that can be possessed by entity. Only one entity can possess a resource at a time; therefore entities compete for resources.

▶ Modify the flowchart



Properties
General | Image | Description

Class name: Model

Base class:

Parameters:

Name	Type

☐ Exclude from build

☑ Show name

☐ Show object rectangle

☑ Public (exported from library)

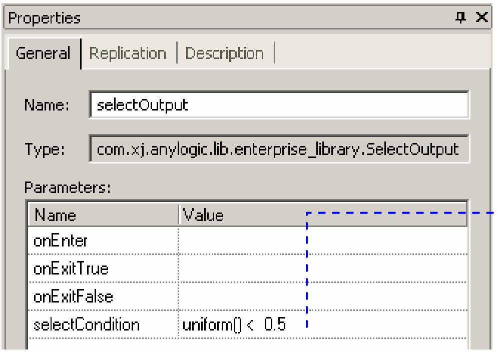
❶ SelectOutput

❷ProcessQ

❸ Resource

❶ Place SelectOutput object. SelectOutput object is a decision making block. The entity arrived at the object is forwarded along one of two output ports depending on the user-defined condition.

Set up the following properties:



Properties

General | Replication | Description

Name: selectOutput

Type: com.xj.anylogic.lib.enterprise_library.SelectOutput

Parameters:

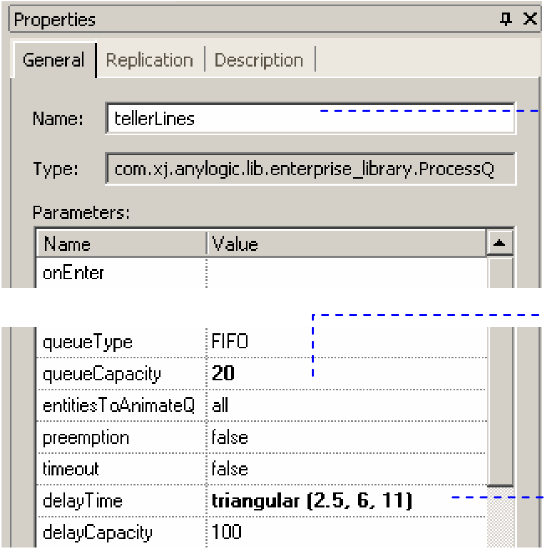
Name	Value
onEnter	
onExitTrue	
onExitFalse	
selectCondition	uniform() < 0.5

Ⓐ Leave the default routing condition

Ⓐ Leave the default `uniform()<0.5` entity routing condition. Thus number of customers competing for ATM and teller service will be approximately equal.

❷ Add a ProcessQ object. ProcessQ seizes resource units for the entity, delays the entity, and releases the seized units.

Set up the following properties:



Properties

General | Replication | Description

Name: tellerLines

Type: com.xj.anylogic.lib.enterprise_library.ProcessQ

Parameters:

Name	Value
onEnter	
queueType	FIFO
queueCapacity	**20**
entitiesToAnimateQ	all
preemption	false
timeout	false
delayTime	**triangular (2.5, 6, 11)**
delayCapacity	100

🅐 Name the object `tellerLines`

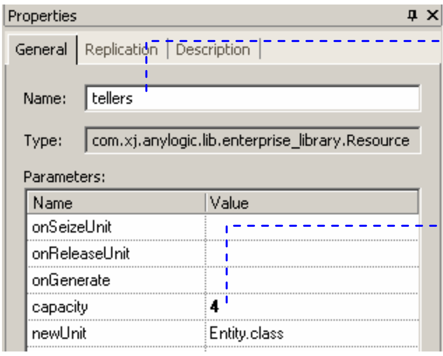
🅑 Set up the queue size to be of 20 places

🅒 Set up delay time

🅒 We assume that service time is triangularly distributed with the min value of 2.5, average value of 6, and the max value of 11 minutes.

❸ Add Resource object. Resource object is storage for resource units. It should be connected to resource seizing and releasing objects (ProcessQ in our case).

Set up the following properties:



Properties

General | Replication | Description

Name: tellers

Type: com.xj.anylogic.lib.enterprise_library.Resource

Parameters:

Name	Value
onSeizeUnit	
onReleaseUnit	
onGenerate	
capacity	**4**
newUnit	Entity.class

🅐 Name the object `tellers`

🅑 Specify that this resource object has only four resource units

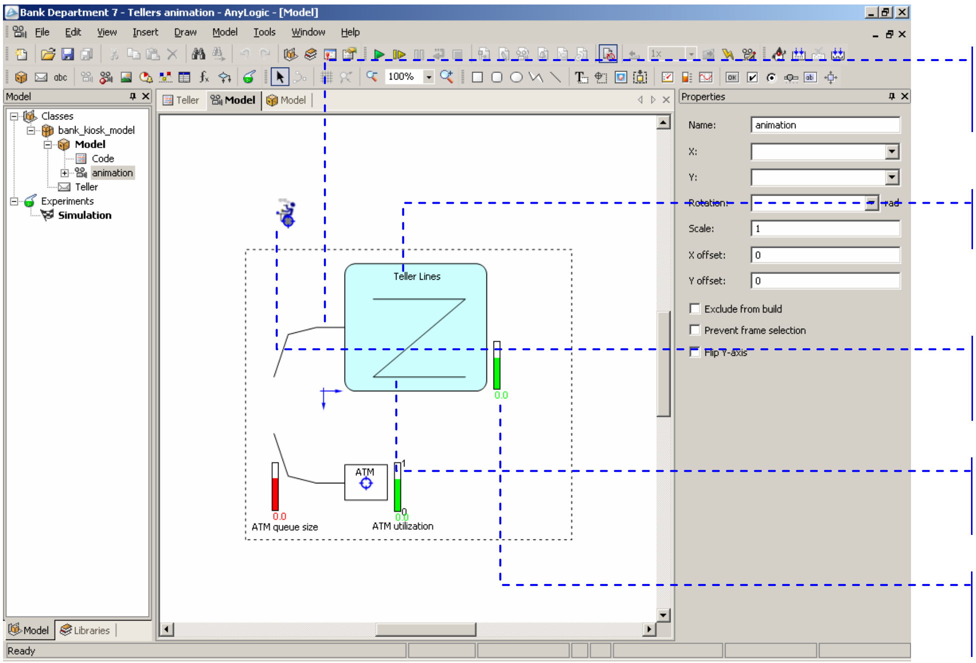
Run the model by clicking the *Run* button and observe the behavior of the modified model.

The reference model for this point is Examples \ Enterprise Library Tutorial Models \ Bank Department 5 - Teller lines.alp.

2.8 Animating tellers

Since the model changed, we need to alter the model animation as well.

► Modify the model animation



Name: animation
X:
Y:
Rotation: rad
Scale: 1
X offset: 0
Y offset: 0
□ Exclude from build
□ Prevent frame selection
□ Flip Y-axis

❶ Waiting customers queue

❷ Activity area

❸ Resource unit symbol

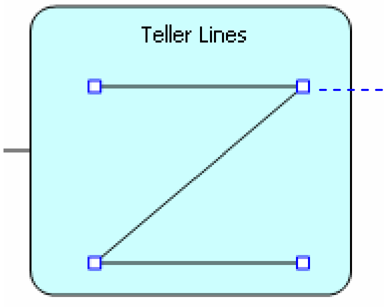
❹ Resource unit locations

❺ Resource utilization indicator

❶ To animate the waiting customers queue, draw a polyline. Using *Polyline* tool, draw the queue shape you like.

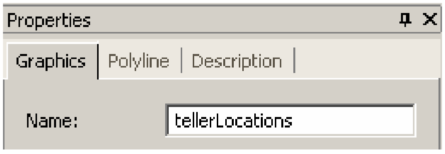
❷ We will visualize the teller lines as a colored box. Draw a rounded rectangle using the *Rounded Rectangle* button. Choose the fill color you like (the box shown in the picture has fill color with red component of 204, green component of 255, and blue component of 255). Add `Teller lines` text label inside the box.

❸ To indicate where resource unit symbols will be placed, draw a polyline using the *Polyline* button.



❶ Polyline points will indicate resource symbol locations

Name the polyline `tellerLocations`.

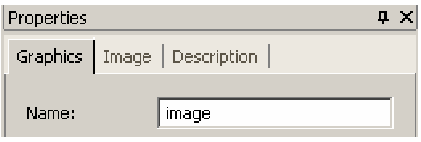


Name: tellerLocations

❹ We will draw tellers inside the activity box.

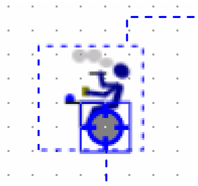
Using an image shape, we will show the teller idle or busy. To create the teller picture, click *Image* 🖼 and place an image shape.

Set up the following properties:



Name: image

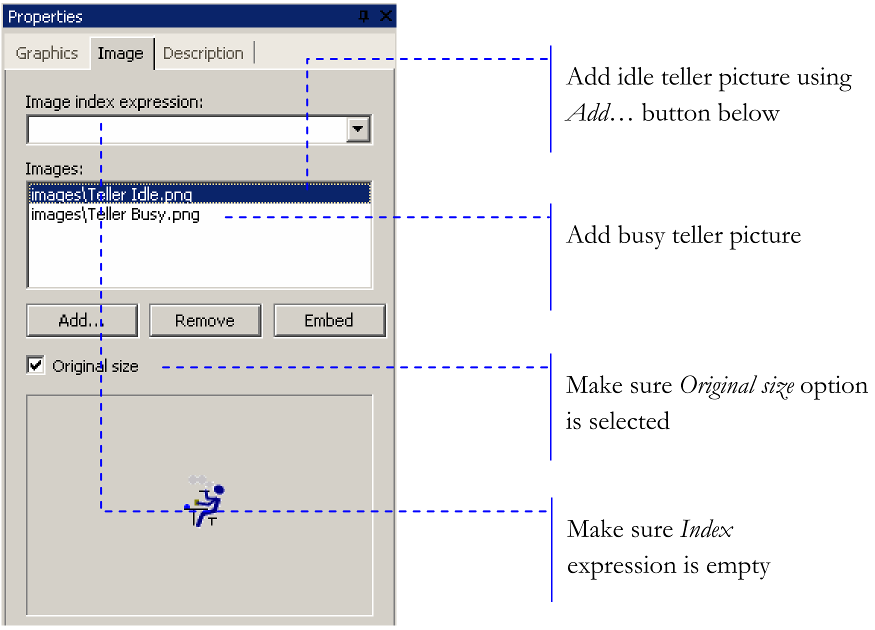
Create a resource visual appearance using a dynamically creating group:



❶ Create an image shape and add idle and busy teller images

❷ Create a pivot and add the image shape to the pivot's group

❶ The image shape should contain the idle and busy teller pictures. Make sure that the shape is named `image`.



Add idle teller picture using *Add…* button below

Add busy teller picture

Make sure *Original size* option is selected

Make sure *Index* expression is empty

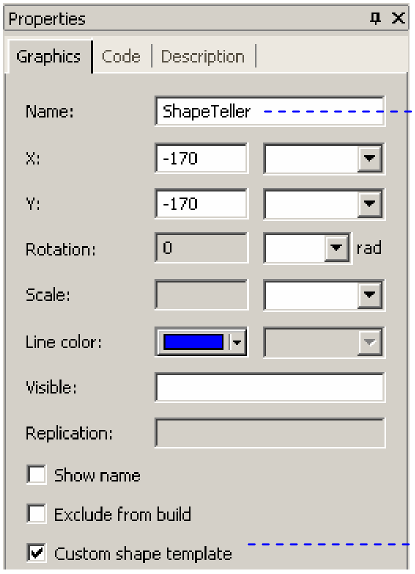
Add the relaxed teller picture to the picture list. Image shape can store multiple pictures and show one of them at run time depending on the expression you write.

The images you can use are Examples \ Enterprise Library Tutorial Models \ images \ Teller Idle.png and Teller Busy.png.

If *Original size* option is not selected, the picture is shown to fit the rectangle of the image shape, and otherwise the picture is shown without any distortions.

Image index expression controls which picture from the list is shown. Make sure that the *Image index expression* is empty.

🅑 Click *Pivot* and click to add a new pivot. Enable dynamic pivot creation. Dynamic pivots can be created and added onto the animation at run time whenever needed. Later, we will associate this pivot with the teller resource to animate tellers.



Properties

Graphics | Code | Description

Name: ShapeTeller

X: -170

Y: -170

Rotation: 0 rad

Scale:

Line color:

Visible:

Replication:

□ Show name

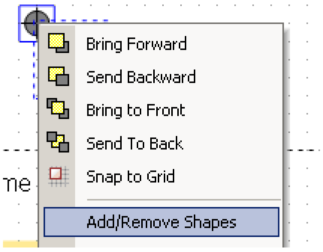
□ Exclude from build

☑ Custom shape template

Name the pivot `ShapeTeller`

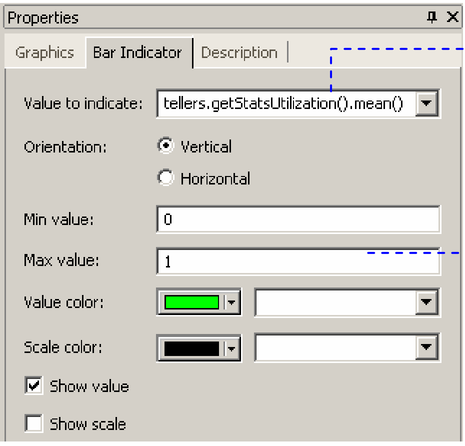
Specify that the pivot instances will be created dynamically

To add the image shape created to the pivot group, right-click the pivot and choose *Add/Remove shapes* from the popup menu.



Then click the image shape to add it to the pivot group. The shape will be highlighted. To leave add/remove mode, click anywhere in the animation.

❺ Place a bar indicator to show operators utilization. Set up the following properties:



Value to indicate: tellers.getStatsUtilization().mean()

Orientation: ● Vertical ○ Horizontal

Min value: 0

Max value: 1

Value color: 🟩

Scale color: ■

☑ Show value

□ Show scale

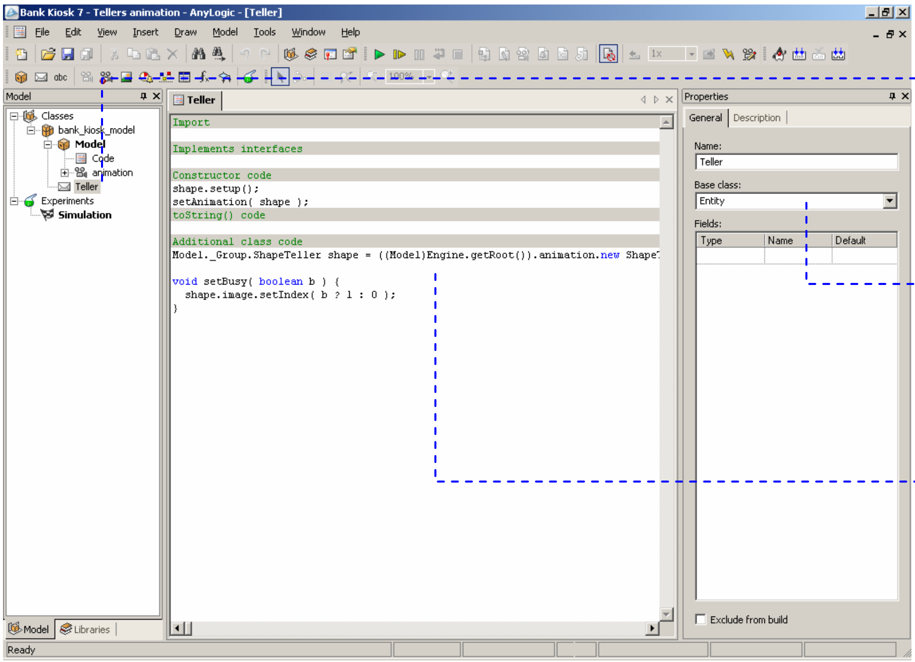
Ⓐ Measure teller resource utilization

Ⓑ Set up the min value of 0 and the max value of 1

Ⓐ To measure operator resource utilization, we use `getStatsUtilization()` function of the Resource object. Then we use `mean()` function to obtain the mean value.

Now we will create a message class to represent teller resource units. We will assign animation to the resource message class created.

► Create `Teller` message class to represent teller resources



```
Import

Implements interfaces

Constructor code
shape.setup();
setAnimation( shape );
toString() code

Additional class code
Model._Group.ShapeTeller shape = ((Model)Engine.getRoot()).animation.new ShapeT

void setBusy( boolean b ) {
  shape.image.setIndex( b ? 1 : 0 );
}
```

Name: Teller

Base class: Entity

Fields:

Type	Name	Default

□ Exclude from build

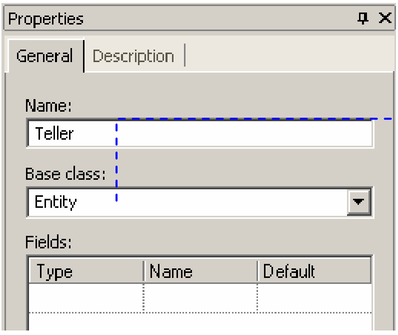
❶ Create `Teller` message class

❷ Select Entity as the base class

❸ Write the animation code

❶ To create a new message class, click the *New Message Class* toolbar button. Name the message class `Teller`.

❷



Name: Teller

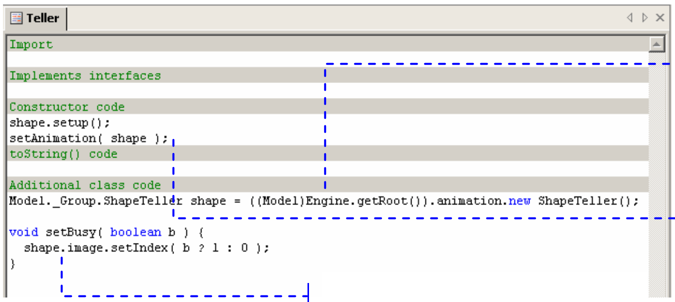
Base class: Entity

Fields:

Type	Name	Default

🅐 Type `Entity` as the base class for the message created

❸



```
Import

Implements interfaces

Constructor code
shape.setup();
setAnimation( shape );
toString() code

Additional class code
Model._Group.ShapeTeller shape = ((Model)Engine.getRoot()).animation.new ShapeTeller();

void setBusy( boolean b ) {
  shape.image.setIndex( b ? 1 : 0 );
}
```

🅐 Create resource animation

🅑 Set up resource animation

🅒 Add a code to change animation at run time

🅐 To create a `ShapeTeller` symbol instance, write the following code:

```
Model._Group.ShapeTeller shape =
((Model)Engine.getRoot()).animation.new
ShapeTeller();
```

This code creates a new dynamic pivot instance.

🅑 To set up the animation, write the following code in the *Constructor code* section:

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

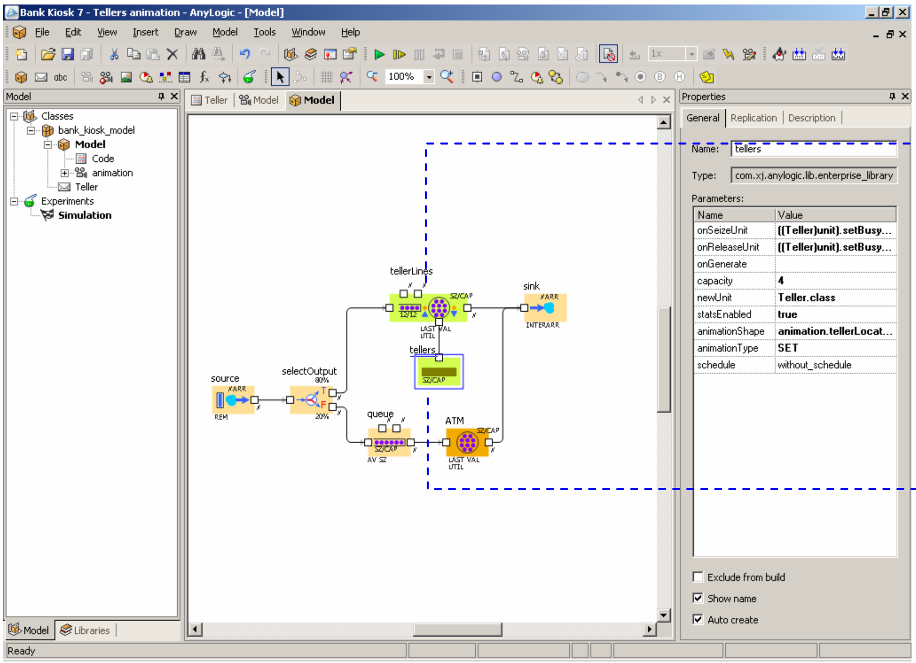
The animation approach will be simplified in the next version of AnyLogic™.

🅒 To alter the appearance at run time, add the following function:

```
void setBusy( boolean b ) {
  shape.image.setIndex( b ? 1 : 0 );
}
```

Now we will animate the flowchart by setting up animation properties of the flowchart objects.

► Animate the model logic



Name: tellers

Type: com.xj.anylogic.lib.enterprise_library

Parameters:

Name	Value
onSeizeUnit	**((Teller)unit).setBusy...**
onReleaseUnit	**((Teller)unit).setBusy...**
onGenerate	
capacity	**4**
newUnit	**Teller.class**
statsEnabled	**true**
animationShape	**animation.tellerLocat...**
animationType	**SET**
schedule	without_schedule

□ Exclude from build

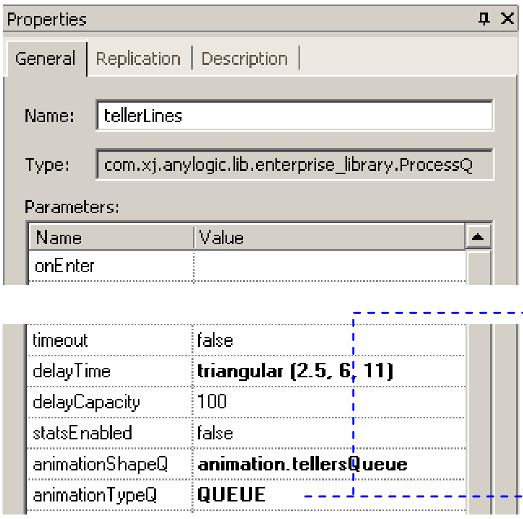
☑ Show name

☑ Auto create

❶ Animate customers queue

❷ Animate tellers

❶ Specify the following animation properties for the ProcessQ object:



Name: tellerLines

Type: com.xj.anylogic.lib.enterprise_library.ProcessQ

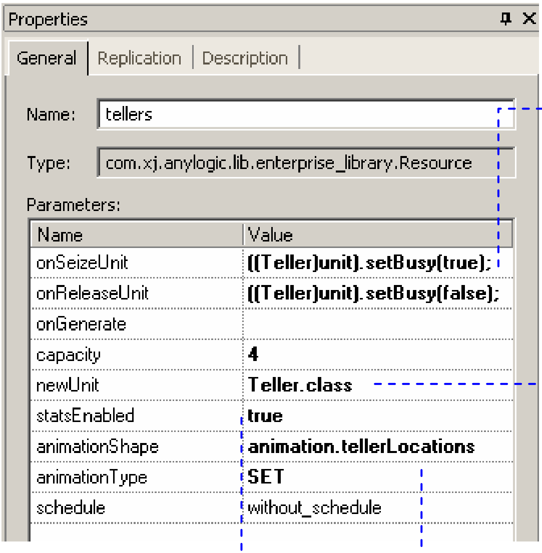
Parameters:

Name	Value
onEnter	
timeout	false
delayTime	**triangular (2.5, 6, 11)**
delayCapacity	100
statsEnabled	false
animationShapeQ	**animation.tellersQueue**
animationTypeQ	**QUEUE**

Ⓐ Set up queue animation path

Ⓑ Set up QUEUE animation type

❷ Set up teller symbol and locations:



Name: tellers

Type: com.xj.anylogic.lib.enterprise_library.Resource

Parameters:

Name	Value
onSeizeUnit	((Teller)unit).setBusy(true);
onReleaseUnit	((Teller)unit).setBusy(false);
onGenerate	
capacity	4
newUnit	Teller.class
statsEnabled	true
animationShape	animation.tellerLocations
animationType	SET
schedule	without_schedule

Ⓐ Switch idle/busy picture when the resource unit is being seized and released

Ⓑ Specify that resource units are messages of the class Teller

Ⓒ Specify resource unit animation locations by setting SET animation style and choosing the polyline shape

Ⓓ Enable resource statistics collection as we will display resource utilization

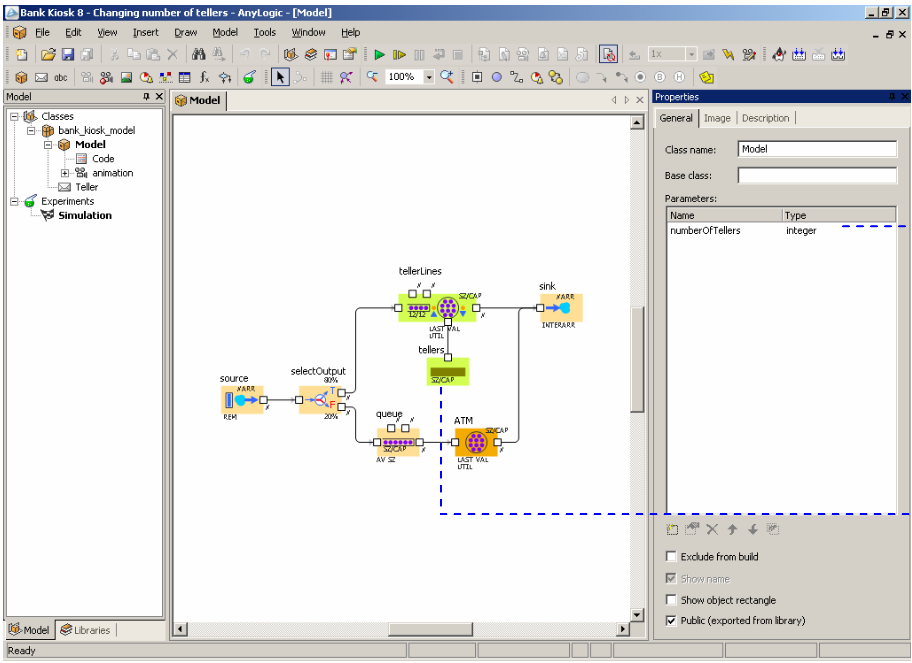
Ⓑ Resource units are messages being passed and received via dedicated object ports. In this model, resources are messages of `Teller` class.

Run the model by clicking the *Run* button. Now you can see that tellers are animated. To adjust the execution speed, use *Decrease model speed* and *Increase model speed* toolbar buttons.

➘ The reference model for this point is Examples \ Enterprise Library Tutorial Models \ Bank Department 6 - Tellers animation.alp.

2.9 Changing resource capacity at run time

Now we will add a slider to change the number of tellers at run time.



Properties — General | Image | Description

Class name: Model

Base class:

Parameters:

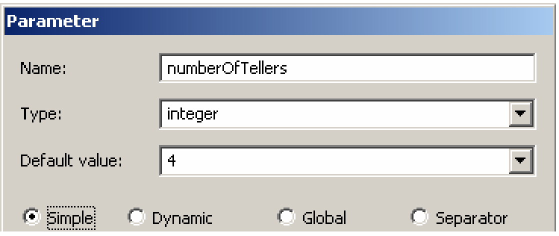
Name	Type
numberOfTellers	integer

☐ Exclude from build
☑ Show name
☐ Show object rectangle
☑ Public (exported from library)

❶ Add new Model parameter

❷ Set up resource capacity

❶ Add a parameter representing the number of tellers. Name the parameter `numberOfTellers` and define it like this:



Parameter

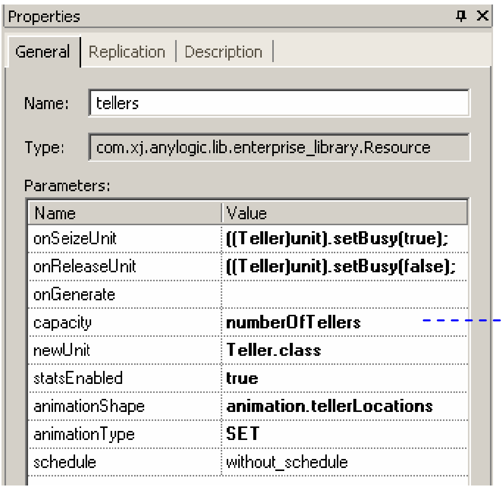
Name: numberOfTellers

Type: integer

Default value: 4

● Simple ○ Dynamic ○ Global ○ Separator

❷



Properties — General | Replication | Description

Name: tellers

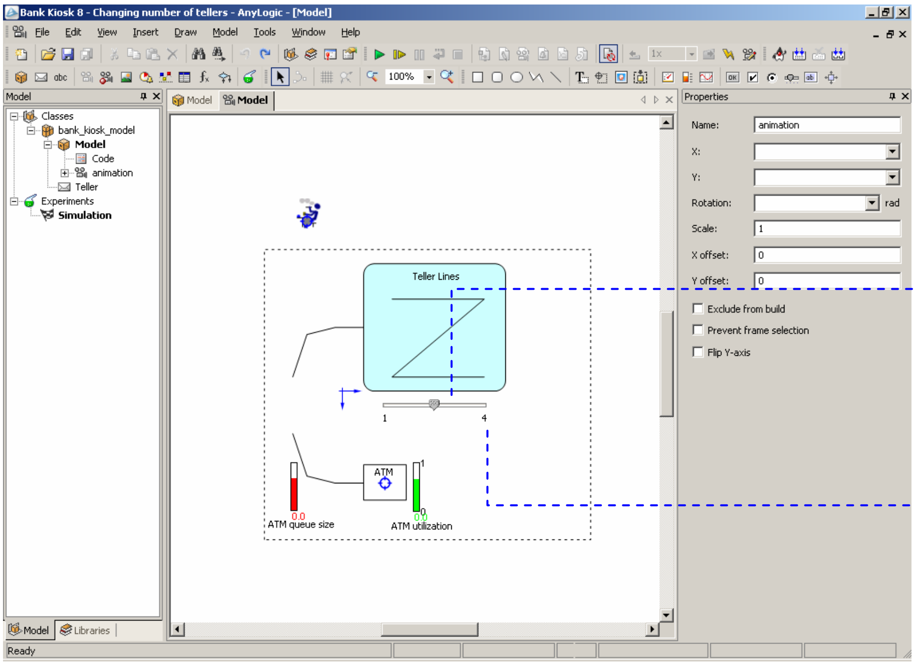
Type: com.xj.anylogic.lib.enterprise_library.Resource

Parameters:

Name	Value
onSeizeUnit	**((Teller)unit).setBusy(true);**
onReleaseUnit	**((Teller)unit).setBusy(false);**
onGenerate	
capacity	**numberOfTellers**
newUnit	**Teller.class**
statsEnabled	**true**
animationShape	**animation.tellerLocations**
animationType	**SET**
schedule	without_schedule

Ⓐ Specify the parameter as the resource capacity

► Add a slider to vary resource capacity



Properties

Name: animation

X:

Y:

Rotation: rad

Scale: 1

X offset: 0

Y offset: 0

□ Exclude from build

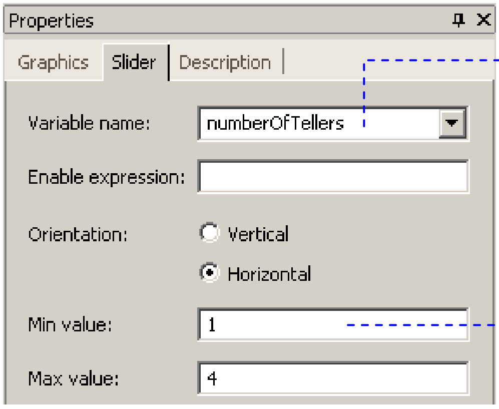
□ Prevent frame selection

□ Flip Y-axis

❶ Add a slider to vary resource capacity

❷ Add slider range labels

❶ Set up the following properties for the slider:



Properties

Graphics | Slider | Description

Variable name: numberOfTellers

Enable expression:

Orientation: ○ Vertical ● Horizontal

Min value: 1

Max value: 4

🅐 Select `numberOfTellers` as the value

🅑 Set up the min value of 1 and the max value of 4

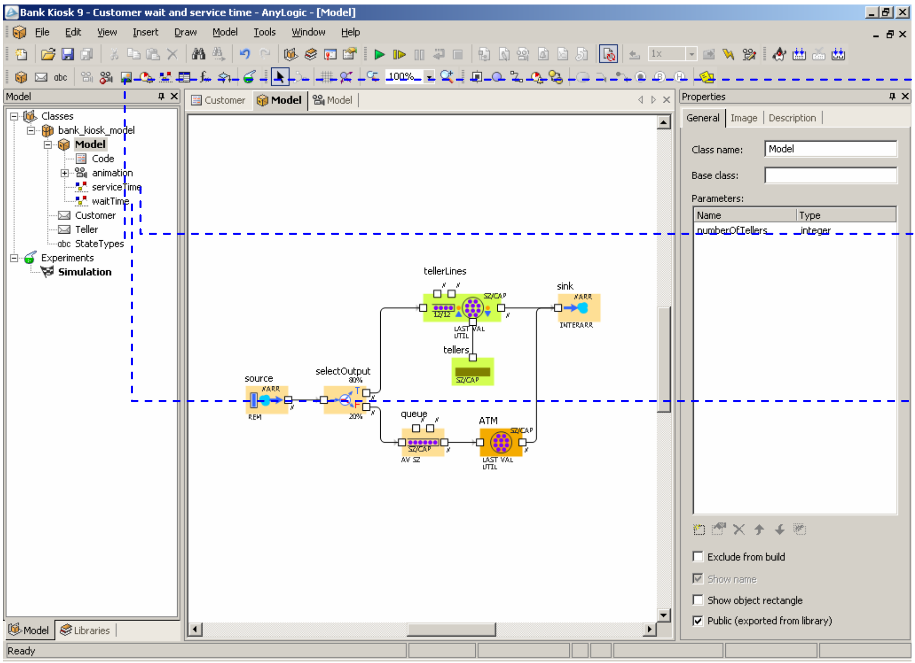
❷ Add text labels displaying min and max values of the slider.

Run the model by clicking the *Run* button. Now you can change the number of tellers. Therefore, you can see how many tellers you need for the specific customer arrival rate.

The reference model for this point is Examples \ Enterprise Library Tutorial Models \ Bank Department 7 - Changing number of tellers.alp.

2.10 Collecting customer time statistics

We want to know how much time customer spends in the bank department being served and waiting for the service. We will collect time statistics using AnyLogic™ datasets.



Class name: Model

Base class:

Parameters:

Name	Type
numberOfTellers	integer

☐ Exclude from build

☑ Show name

☐ Show object rectangle

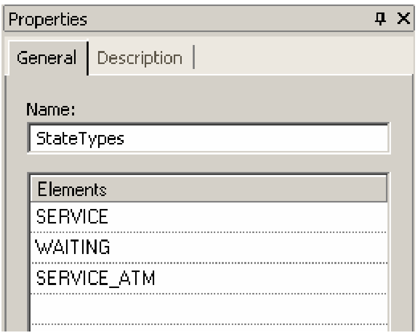
☑ Public (exported from library)

❶ Create `StateTypes` enumeration

❷ Create `serviceTime` dataset

❸ Create `waitTime` dataset

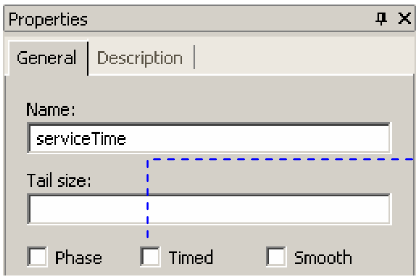
❶ Create `StateTypes` enumeration by clicking the *New Enumeration* abc toolbar button. Define the following enumeration elements:



Name: StateTypes

Elements
SERVICE
WAITING
SERVICE_ATM

❷ Add a dataset to measure customer service time. Create `serviceTime` dataset by clicking the *New Dataset* toolbar button. For the dataset, leave all properties by default.



Name: serviceTime

Tail size:

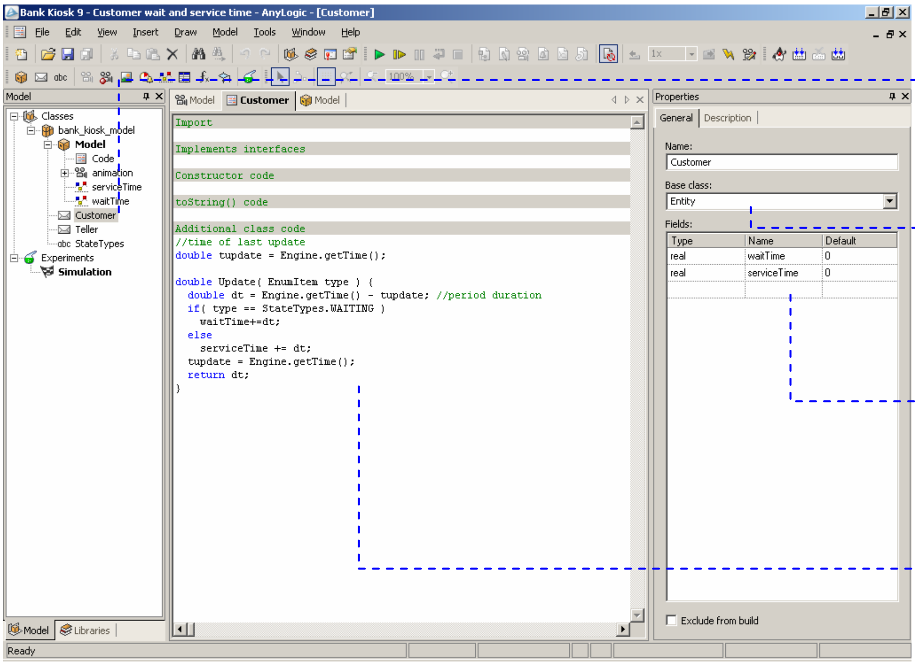
☐ Phase ☐ Timed ☐ Smooth

Ⓐ Make sure the dataset is not timed

❸ Add `waitTime` dataset in the same manner.

Now we will create a message class that is able to carry and count time.

► Create time-enabled message



```
Import

Implements interfaces

Constructor code

toString() code

Additional class code
//time of last update
double tupdate = Engine.getTime();

double Update( EnumItem type ) {
  double dt = Engine.getTime() - tupdate; //period duration
  if( type == StateTypes.WAITING )
    waitTime+=dt;
  else
    serviceTime += dt;
  tupdate = Engine.getTime();
  return dt;
}
```

Name: Customer

Base class: Entity

Fields:

Type	Name	Default
real	waitTime	0
real	serviceTime	0

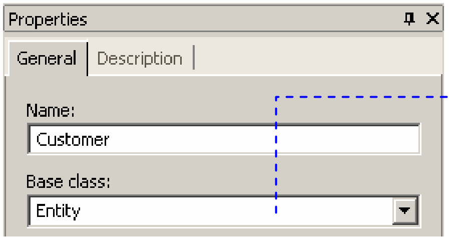
❶ Create `Customer` message class

❷ Set Entity as the base class

❸ Add some message class parameters

❹ Add some message class code

❷

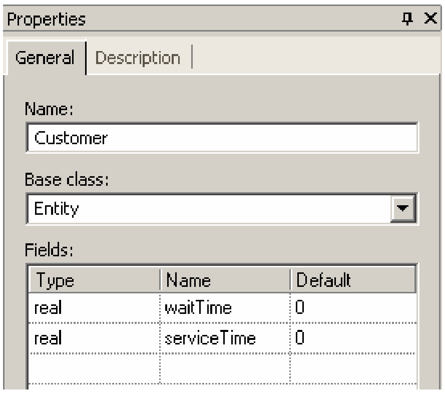


Name: Customer

Base class: Entity

Ⓐ Choose Entity as the base class for the message created

❸ Add message class parameters to carry time information:



Name: Customer

Base class: Entity

Fields:

Type	Name	Default
real	waitTime	0
real	serviceTime	0

`waitTime` and `serviceTime` fields will be incurred by the model as the entity flows through the model.

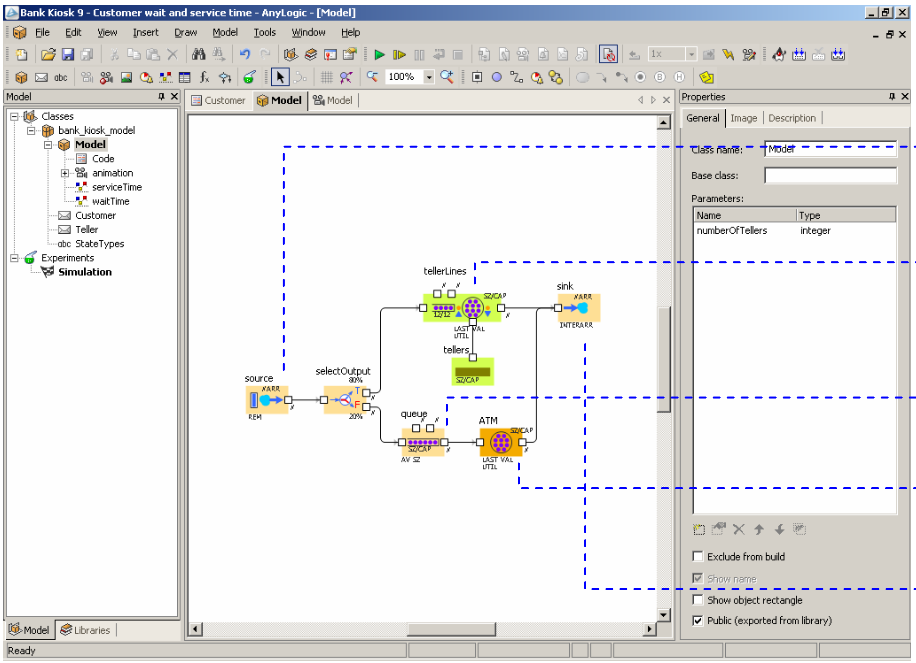
❹ Write the following code in the *Additional class code* section to provide function that will incur `waitTime` and `serviceTime`:

```
double tupdate = Engine.getTime();
double Update( EnumItem type ) {
  double dt = Engine.getTime() - tupdate;
  if( type == StateTypes.WAITING )
    waitTime+=dt;
  else
    serviceTime+=dt;
  tupdate = Engine.getTime();
  return dt;
}
```

This function is invoked after some activity took place. It adds the time entity has spent at this activity to the corresponding dataset; that is, this function collects entity time statistics.

Now we can calculate the customer service time, as well as the customer wait time.

▶ Calculate customer service and wait time



Class name: Model

Base class:

Parameters:

Name	Type
numberOfTellers	integer

□ Exclude from build

☑ Show name

□ Show object rectangle

☑ Public (exported from library)

❶ Source

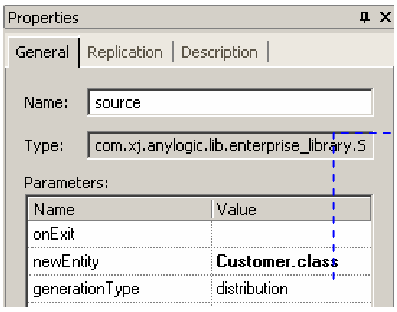
❷ ProcessQ

❸ Queue

❹ Delay

❺ Sink

❶ Set the following properties for the Source:



Name: source

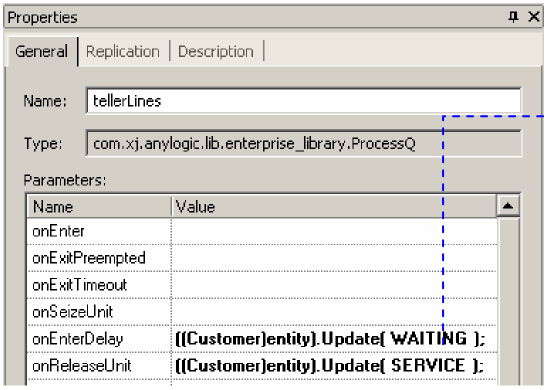
Type: com.xj.anylogic.lib.enterprise_library.S

Parameters:

Name	Value
onExit	
newEntity	**Customer.class**
generationType	distribution

Ⓐ Set `Customer` entity type

❷ Set the following properties:



Name: tellerLines

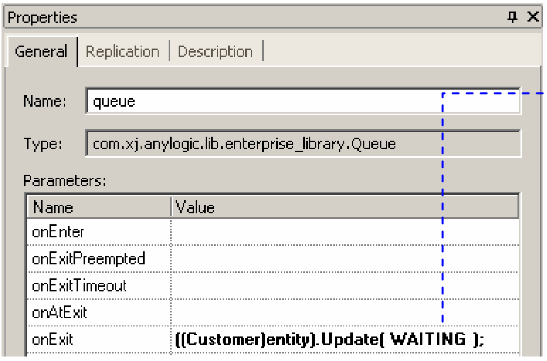
Type: com.xj.anylogic.lib.enterprise_library.ProcessQ

Parameters:

Name	Value
onEnter	
onExitPreempted	
onExitTimeout	
onSeizeUnit	
onEnterDelay	**((Customer)entity).Update(WAITING);**
onReleaseUnit	**((Customer)entity).Update(SERVICE);**

Ⓐ Write this code to incur the entity cost

❸ Set the following properties:



Name: queue

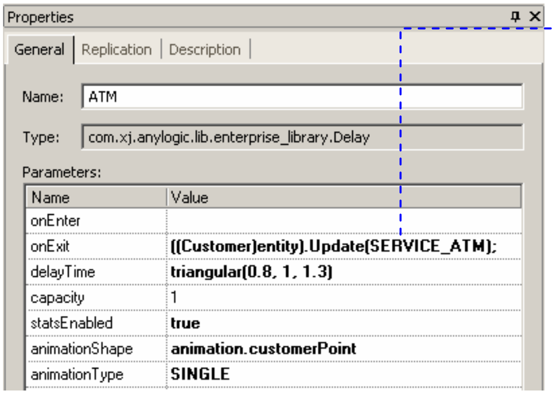
Type: com.xj.anylogic.lib.enterprise_library.Queue

Parameters:

Name	Value
onEnter	
onExitPreempted	
onExitTimeout	
onAtExit	
onExit	((Customer)entity).Update(WAITING);

Ⓐ Write this code to incur the entity cost

❹ Set the following properties:



Name: ATM

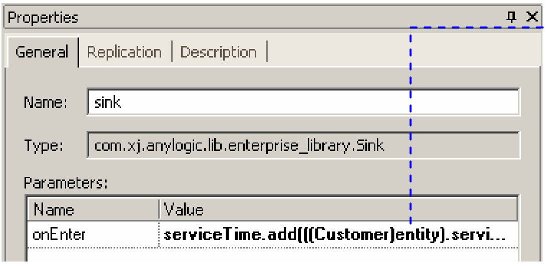
Type: com.xj.anylogic.lib.enterprise_library.Delay

Parameters:

Name	Value
onEnter	
onExit	((Customer)entity).Update(SERVICE_ATM);
delayTime	triangular(0.8, 1, 1.3)
capacity	1
statsEnabled	true
animationShape	animation.customerPoint
animationType	SINGLE

Ⓐ Write this code to incur the entity cost

❺ Set the following properties:



Name: sink

Type: com.xj.anylogic.lib.enterprise_library.Sink

Parameters:

Name	Value
onEnter	serviceTime.add(((Customer)entity).servi...

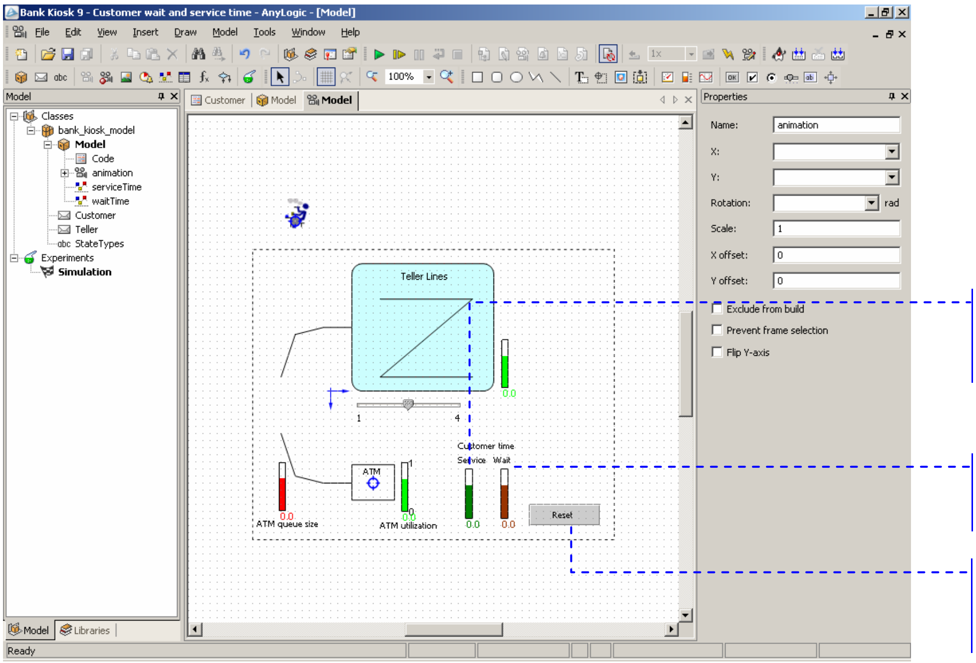
Ⓐ Write code to incur the entity cost

Ⓐ Write the following code to add exiting customer time statistics to the datasets:

```
serviceTime.add(((Customer)entity).serviceTime);
waitTime.add(((Customer)entity).waitTime);
```

We want to display the statistics collected on the model animation.

► Add indicators to the animation



Properties

Name: animation

X:

Y:

Rotation: rad

Scale: 1

X offset: 0

Y offset: 0

□ Exclude from build

□ Prevent frame selection

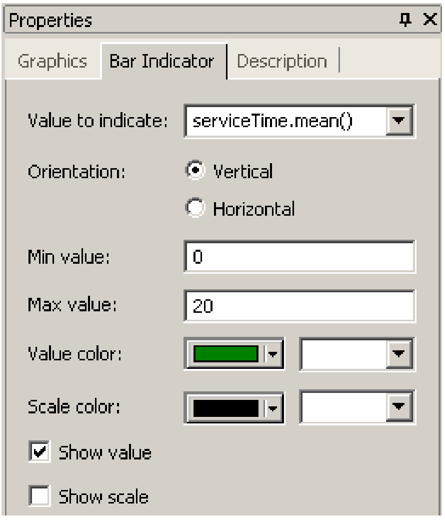
□ Flip Y-axis

❶ Add customer service time indicator

❷ Add customer wait time indicator

❸ Add reset statistics button

❶ Set up the following indicator properties:



Value to indicate: serviceTime.mean()

Orientation: ● Vertical ○ Horizontal

Min value: 0

Max value: 20

Value color: 🟩

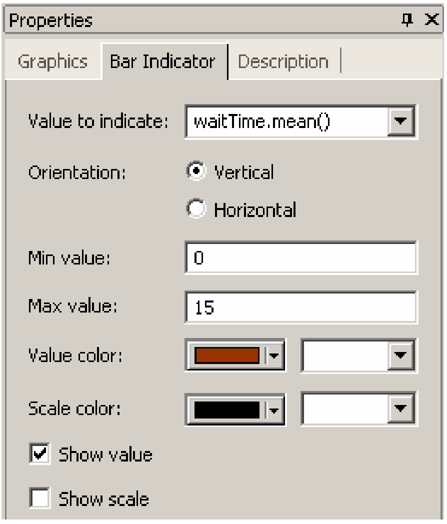
Scale color: ■

☑ Show value

□ Show scale

http://www.xjtek.com

❷ Set up the following indicator properties:



Properties

Graphics | Bar Indicator | Description

Value to indicate: waitTime.mean()

Orientation: ● Vertical ○ Horizontal

Min value: 0

Max value: 15

Value color:

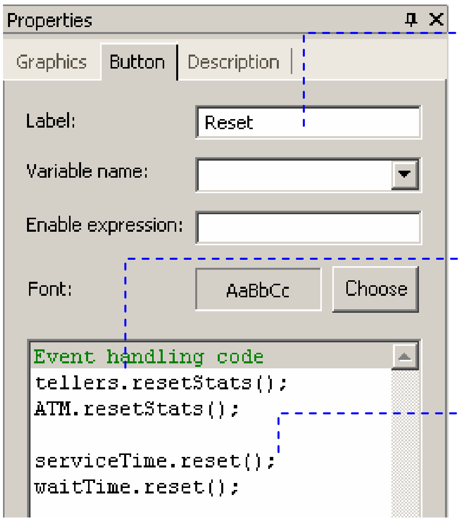
Scale color:

☑ Show value

□ Show scale

❸ We will add a button to reset the collected statistics after changing the number of tellers, to collect some statistics for the new value only.

Click the *Button* OK toolbar button and then click the animation diagram to place it. Set the following button properties:



Properties

Graphics | Button | Description

Label: Reset

Variable name:

Enable expression:

Font: AaBbCc Choose

```
Event handling code
tellers.resetStats();
ATM.resetStats();

serviceTime.reset();
waitTime.reset();
```

🅐 Type Reset as the button label

🅑 Write the code that will reset tellers and ATM utilization statistics

🅒 Write the code that will reset time statistics

You reset statistics calling the special functions in the button's action code.

🅑 The `resetStats()` function of a Delay object resets the statistics collected by the object.

🅒 The `reset()` function of a dataset resets the statistics collected by the dataset.

Run the model by clicking the *Run* ▶ button. Time statistics is now collected. You may click the Reset button to reset statistics.

➲ In this example, we reset statistics using a button. You also can reset statistics at certain time, or as a reaction to any event using the AnyLogic^{TM} timer (see *User's Manual* for information about timers).

➘ The reference model for this point is Examples \ Enterprise Library Tutorial Models \ Bank Department 8 - Customer wait and service time.alp.

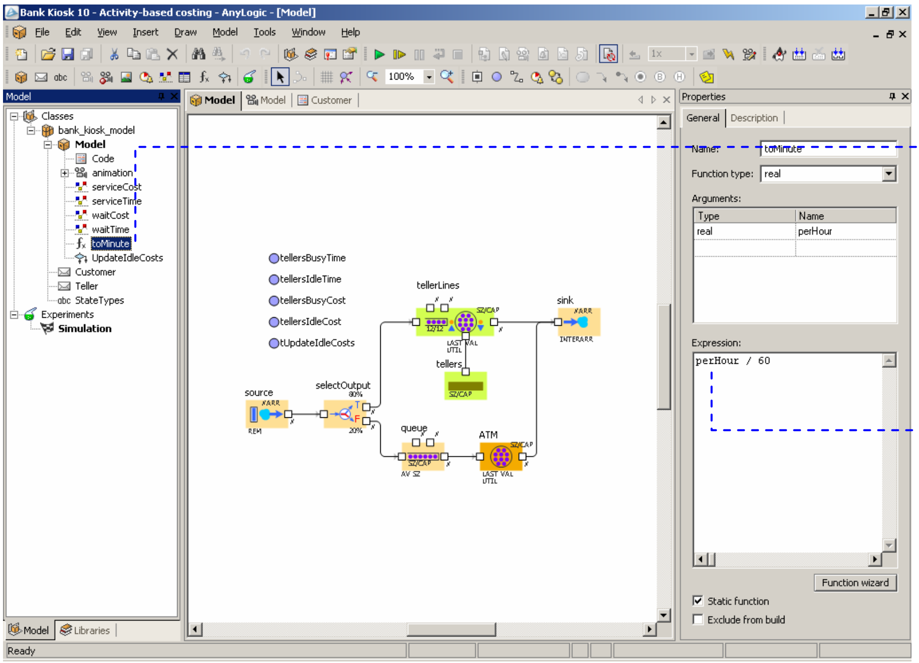
2.11 Activity-based costing

AnyLogic^{TM} helps you understand, visualize, analyze and improve business processes, and with the Enterprise Library, modeling and analyzing business processes has never been so easy. With AnyLogic^{TM} and the Enterprise Library, you can also make use of activity-based costing to see how much the process really costs and where the cost comes from, discover the options to reduce cost, improve efficiency, and provide better customer service.

We will apply ABC in the bank department example to see how much it costs to serve a customer for the company, and what parts are wait cost and service cost.

Since resource costs are usually measured per hour, first we will write a function that will convert the per-hour cost to per-minute cost.

► Write salary conversion function



Properties — General | Description

Name: toMinute

Function type: real

Arguments:

Type	Name
real	perHour

Expression: perHour / 60

☑ Static function

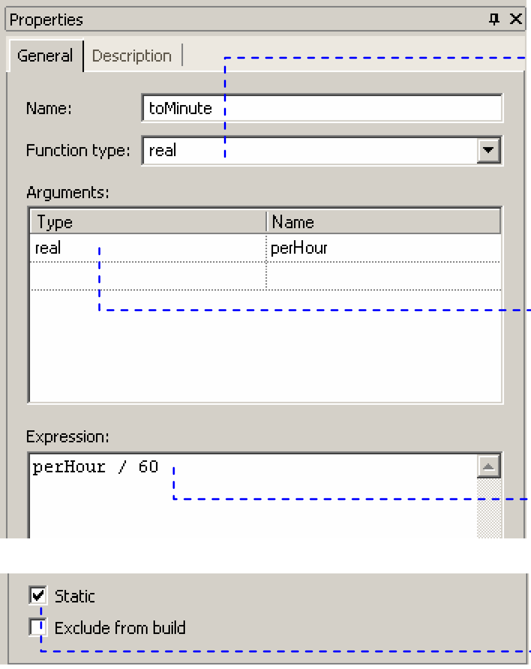
□ Exclude from build

❶ Create new mathematical function

❷ Write function expression

❶ Click the *New Mathematical Function* toolbar button. Name the function `toMinute`.

❷ Specify the following function type, arguments and body:


Properties — General | Description

Name: toMinute

Function type: real

Arguments:

Type	Name
real	perHour

Expression:

```
perHour / 60
```

☑ Static

□ Exclude from build

Ⓐ Select `real` result type

Ⓑ Add `perHour` argument of type `real`

Ⓒ Write function expression

Ⓓ Mark this function as *Static*

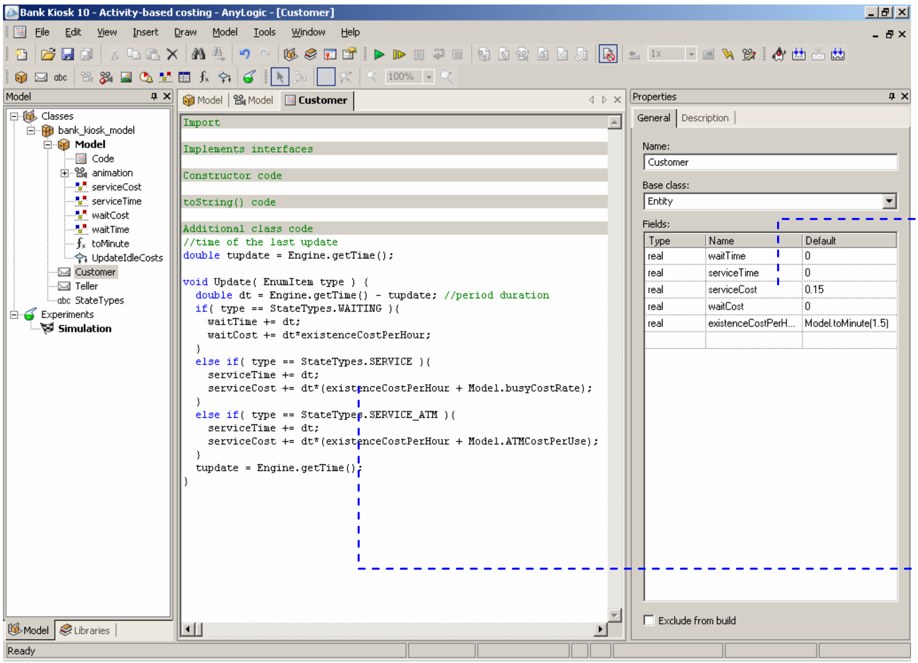
Ⓒ Write the following expression:

```
perHour / 60
```

Ⓓ We specify the function as `static`—that means that this function does not use any value specific to instances of the Model class. Therefore, this function relates to the Model class and can be called using `Model.toMinute()` notation. Otherwise, you would need to get a reference to the specific instance of the Model class.

Now we will modify our message class to enable entities to collect cost statistics.

▶ Modify Customer message class



```
Import

Implements interfaces

Constructor code

toString() code

Additional class code
//time of the last update
double tupdate = Engine.getTime();

void Update( EnumItem type ) {
  double dt = Engine.getTime() - tupdate; //period duration
  if( type == StateTypes.WAITING ){
    waitTime += dt;
    waitCost += dt*existenceCostPerHour;
  }
  else if( type == StateTypes.SERVICE ){
    serviceTime += dt;
    serviceCost += dt*(existenceCostPerHour + Model.busyCostRate);
  }
  else if( type == StateTypes.SERVICE_ATM ){
    serviceTime += dt;
    serviceCost += dt*(existenceCostPerHour + Model.ATMCostPerUse);
  }
  tupdate = Engine.getTime();
}
```

Name: Customer

Base class: Entity

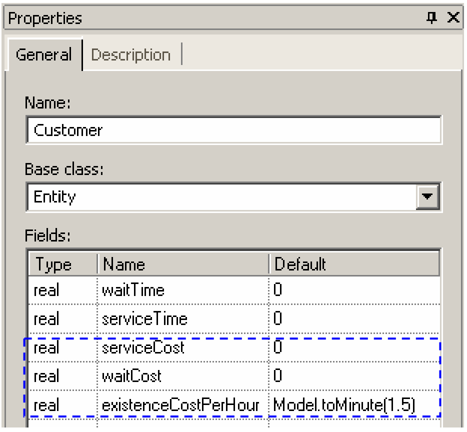
Fields:

Type	Name	Default
real	waitTime	0
real	serviceTime	0
real	serviceCost	0.15
real	waitCost	0
real	existenceCostPerH...	Model.toMinute(1.5)

❶ Add some message class parameters

❷ Add some message class code

❶ Add message class parameters to carry cost information:



Name: Customer

Base class: Entity

Fields:

Type	Name	Default
real	waitTime	0
real	serviceTime	0
real	serviceCost	0
real	waitCost	0
real	existenceCostPerHour	Model.toMinute(1.5)

serviceCost field will count how much it costs to the company to serve this customer.

Note that besides service cost, wait cost is also considered.

existenceCostPerHour stands for how much it costs to hold a customer in the system. We specify holding cost rate per minute. Since holding cost is often specified per hour, we will use the function toMinute() to obtain per-minute cost.

serviceCost and waitCost fields will be incurred by the model as the entity flows through the model.

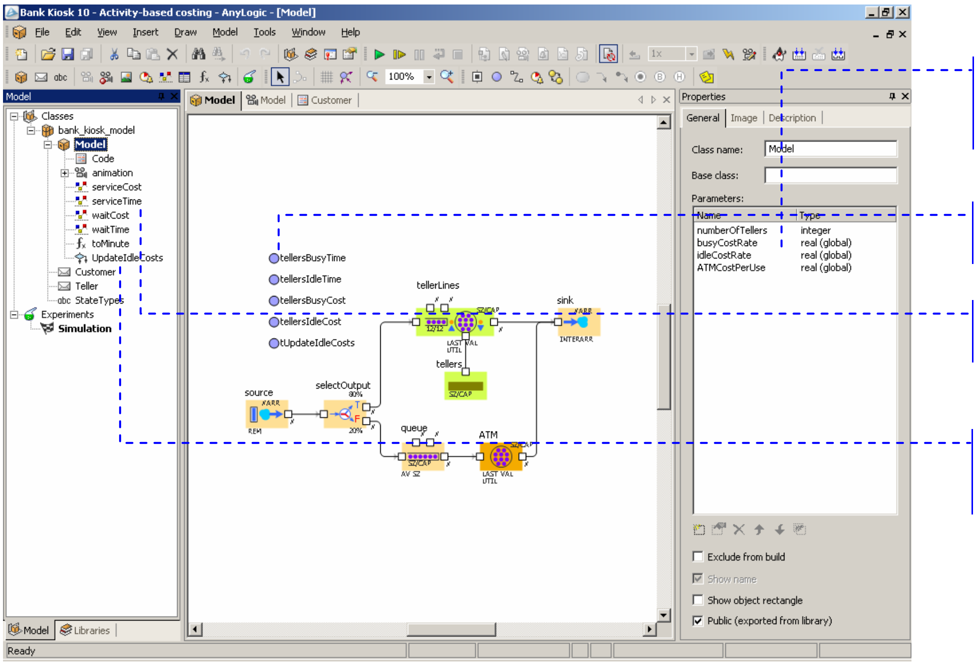
❷ Double-click Customer in the tree to open its window and modify the Update() function in the *Additional class code* section:

```
void Update( EnumItem type ) {
  double dt = Engine.getTime() - tupdate;
  if( type == StateTypes.WAITING ){
    waitTime += dt;
    waitCost += dt*existenceCostPerHour;
  }
  else if( type == StateTypes.SERVICE ){
    serviceTime += dt;
    serviceCost += dt*(existenceCostPerHour +
Model.busyCostRate);
  }
  else if( type == StateTypes.SERVICE_ATM ){
    serviceTime += dt;
    serviceCost += dt*(existenceCostPerHour +
Model.ATMCostPerUse);
  }
  tupdate = Engine.getTime();
}
```

The function will now incur customer service and wait costs.

Now we will add some auxiliary elements to accumulate customer and teller cost information.

► Create cost accumulators



Class name: Model

Parameters:

Name	Type
numberOfTellers	integer
busyCostRate	real (global)
idleCostRate	real (global)
ATMCostPerUse	real (global)

❶ Add Model class parameters

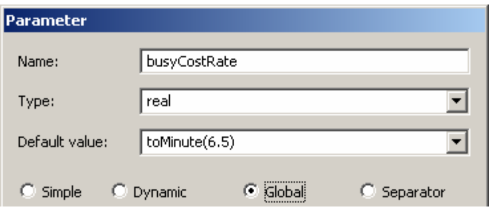
❷ Add variables

❸ Add datasets

❹ Define an algorithmic function

❶ Create parameters defining busy and idle teller costs.

Define the parameters as following:



Parameter

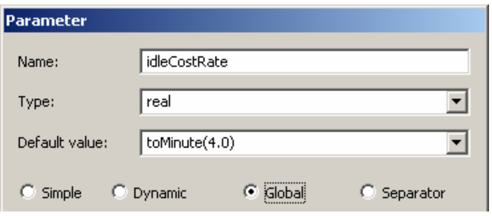
Name: busyCostRate

Type: real

Default value: toMinute(6.5)

○ Simple ○ Dynamic ● Global ○ Separator

We specify that a working hour of a teller costs $6.5 while idle teller hour costs $4.0.



Parameter

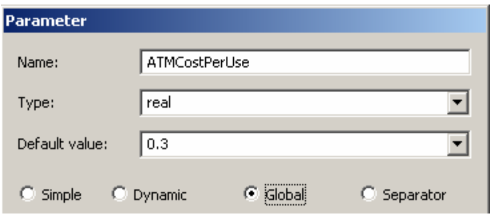
Name: idleCostRate

Type: real

Default value: toMinute(4.0)

○ Simple ○ Dynamic ● Global ○ Separator

Create ATM cost per use parameter. Define the parameter as following:



Parameter

Name: ATMCostPerUse

Type: real

Default value: 0.3

○ Simple ○ Dynamic ● Global ○ Separator

Serving a customer at ATM costs $0.30.

Note that it is possible to associate resource cost with a resource message by adding message class parameters and assessing resource costs in onSeizeUnit and onReleaseUnit actions of the ProcessQ object.

❷ Create `timeUpdateCosts` variable by clicking the *Variable* toolbar button and clicking the structure diagram. Create some more variables named as following:

- `tellersIdleTime`
- `tellersBusyTime`
- `tellersIdleCost`
- `tellersBusyCost`

These variables will accumulate time and cost statistics for tellers.

❸ Create datasets named as following:

- `waitCost`
- `serviceCost`

Make sure the datasets created are not timed; that is, *Timed* option is not selected for all datasets. These datasets will store wait and service costs for each served customer.

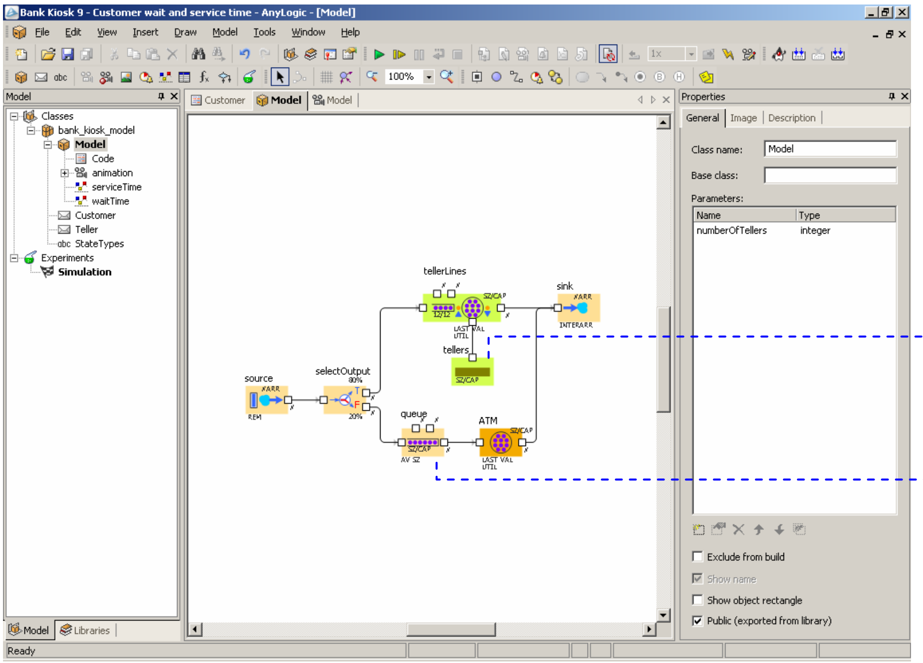
❹ Create an algorithmic function to update tellers cost and time statistics. Do it by clicking the *New Algorithmic Function* button. Name it `UpdateCosts` and change its *Function type* to `void`.

```
double dt = getTime() - timeUpdateCosts;
  tellersIdleTime += tellers.size()*dt;
  tellersBusyTime += (tellers.capacity-
tellers.size())*dt;
  tellersIdleCost += tellersIdleTime*idleCostRate;
  tellersBusyCost += tellersBusyTime*busyCostRate;
```

```
timeUpdateCosts = getTime();
```

Now we can calculate the total customer service cost, as well as the customer wait cost.

► Calculate customer service and wait cost



Class name: Model

Base class:

Parameters:

Name	Type
numberOfTellers	integer

□ Exclude from build

☑ Show name

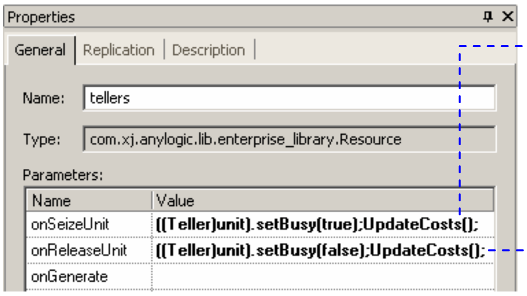
□ Show object rectangle

☑ Public (exported from library)

❶ Modify `tellers` to incur teller costs

❷ Modify `sink`

❶ Set the following properties:



Properties

General | Replication | Description

Name: tellers

Type: com.xj.anylogic.lib.enterprise_library.Resource

Parameters:

Name	Value
onSeizeUnit	((Teller)unit).setBusy(true);UpdateCosts();
onReleaseUnit	((Teller)unit).setBusy(false);UpdateCosts();
onGenerate	

Ⓐ Add some code to incur teller costs

Ⓑ Add some code to incur teller costs

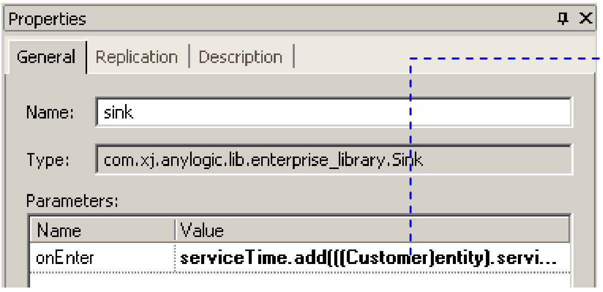
Ⓐ Add the code shown in bold:

```
((Teller)unit).setBusy(true); UpdateCosts();
```

Ⓑ Add the code shown in bold:

```
((Teller)unit).setBusy(true); UpdateCosts();
```

❷ Set the following properties:



Properties

General | Replication | Description

Name: sink

Type: com.xj.anylogic.lib.enterprise_library.Sink

Parameters:

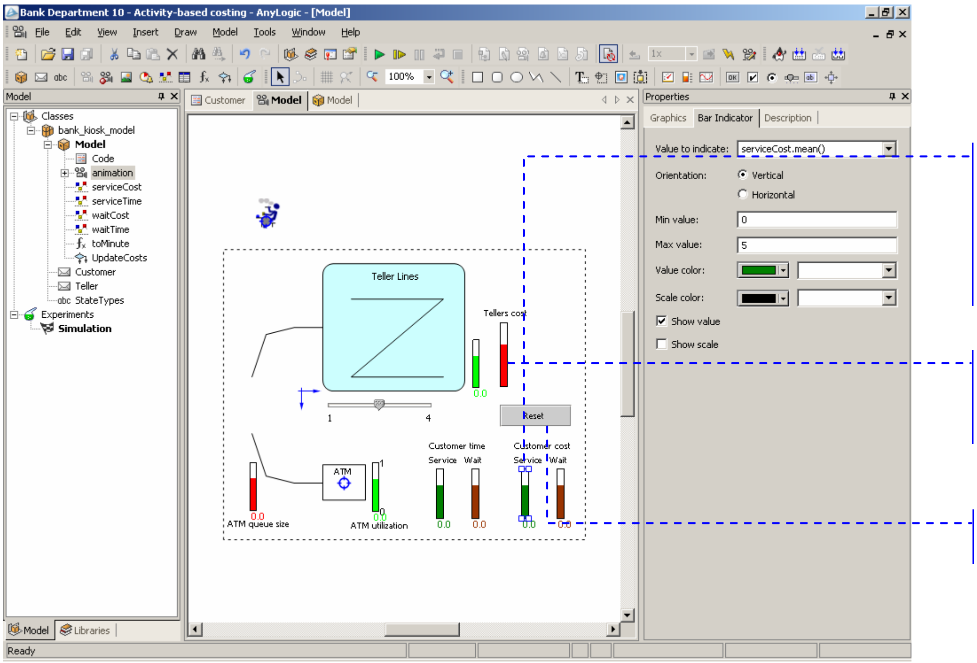
Name	Value
onEnter	serviceTime.add(((Customer)entity).servi...

Ⓐ Add some code

Ⓐ Add the lines shown in bold to add exiting customer cost statistics to the cost datasets:

```
serviceTime.add(((Customer)entity).serviceTime);
waitTime.add(((Customer)entity).waitTime);
serviceCost.add(((Customer)entity).serviceCost);
waitCost.add(((Customer)entity).waitCost);
```

▶ Add cost indicators onto the animation



Value to indicate: serviceCost.mean()

Orientation: ● Vertical ○ Horizontal

Min value: 0

Max value: 5

Value color: 🟩

Scale color: ■

☑ Show value

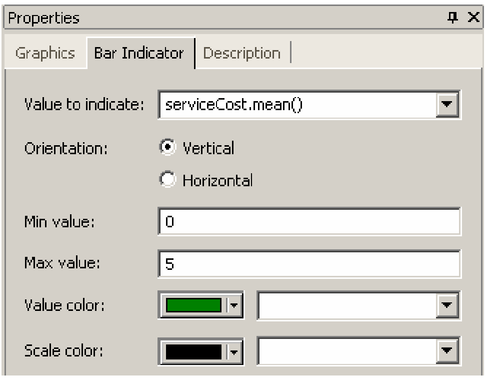
□ Show scale

❶ Add customer wait and service cost indicators

❷ Add tellers cost indicator

❸ Reset statistics

❶ Set up the following options for the customer service cost indicator:



Properties

Graphics | Bar Indicator | Description

Value to indicate: serviceCost.mean()

Orientation: ● Vertical ○ Horizontal

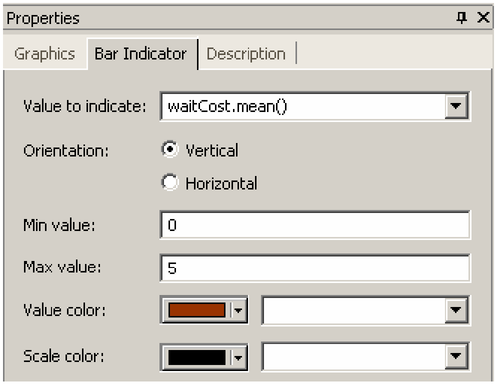
Min value: 0

Max value: 5

Value color: 🟩

Scale color: ■

Set up the following options for the customer wait cost indicator:



Value to indicate: waitCost.mean()

Orientation: ● Vertical ○ Horizontal

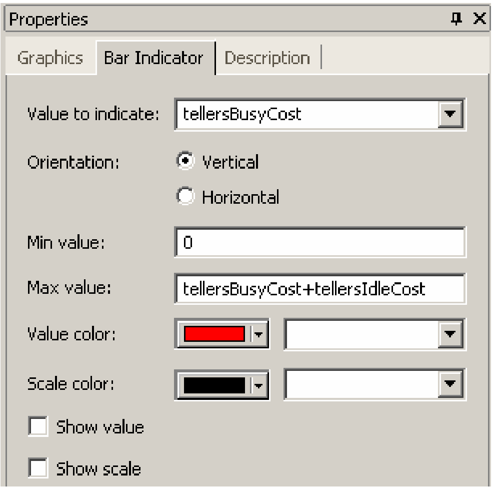
Min value: 0

Max value: 5

Value color: 🟫

Scale color: ⬛

❷ Add indicator to show the idle-busy teller costs relationship. Set up the following options:



Value to indicate: tellersBusyCost

Orientation: ● Vertical ○ Horizontal

Min value: 0

Max value: tellersBusyCost+tellersIdleCost

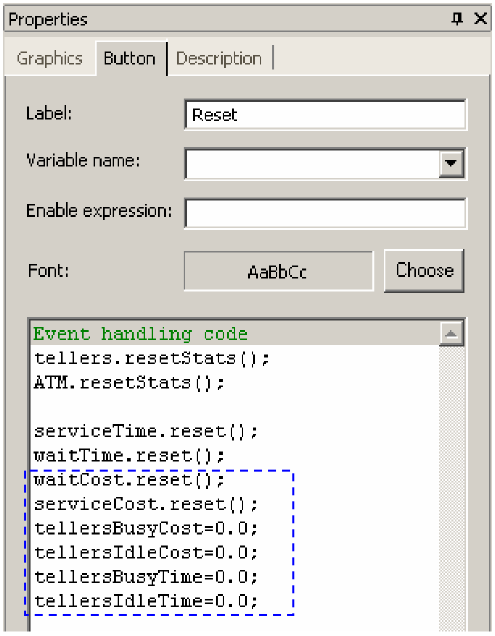
Value color: 🟥

Scale color: ⬛

□ Show value

□ Show scale

❸ Add the following lines to reset statistics:



Label: Reset

Variable name:

Enable expression:

```
tellers.resetStats();
ATM.resetStats();

serviceTime.reset();
waitTime.reset();
waitCost.reset();
serviceCost.reset();
tellersBusyCost=0.0;
tellersIdleCost=0.0;
tellersBusyTime=0.0;
tellersIdleTime=0.0;
```

Run the model by clicking the *Run* ▶ button. You will see statistics about customer service and wait cost. Now you can see where the cost comes from and discover how to improve efficiency and provide better customer service.

↘ The reference model for this point is Examples \ Enterprise Library Tutorial Models \ Bank Department 9 - Activity-based costing.alp.

3. Shop Floor Model

In this section, we will consider the model of a factory. The factory processes parts at a processing station and then assembles a product from two parts of different types at assembly machine. The part supply provides parts of different types independently—each part interarrival mean time is 10 minutes. Parts are moved by a conveyor to the processing station. The conveyor moves a part with the speed of 200 feet per minute; each part occupies 10 feet on the conveyor. The processing station processes one part at a time, and the processing rate is 20 parts per minute. Having been processed, the part is moved by conveyors to the assembly station to finish processing. Two conveyors leading to the assembly station, each moving red or blue parts only, have the speed of 300 feet per minute. The assembly station processing rate is 20 assemblies per minute. Finally, finished parts are moved by a conveyor, which has the speed of 300 feet per minute.

3.1 Creating a new project

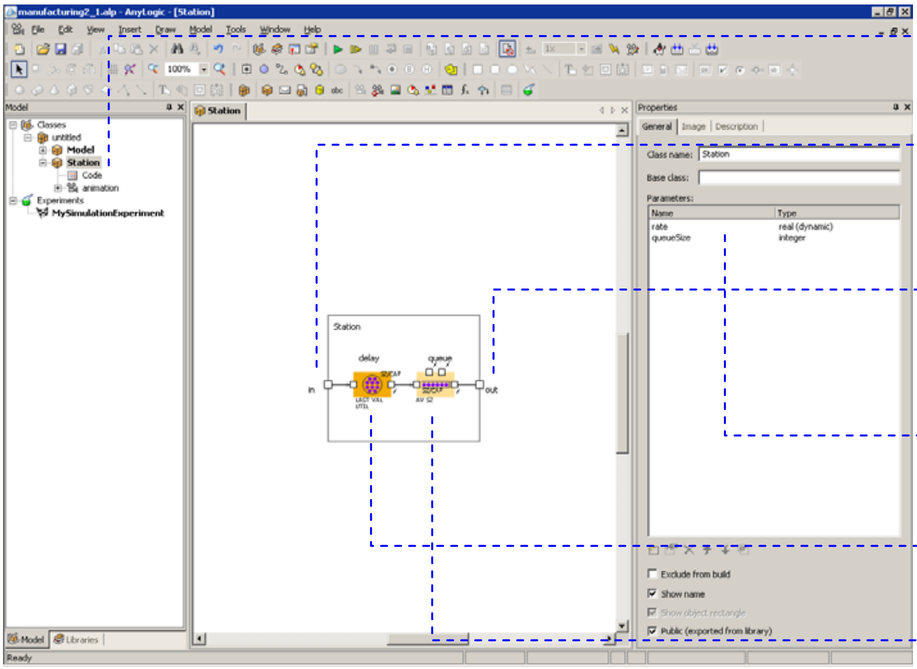
Create a new AnyLogic™ project as described in Section 1.1, "How to create a new AnyLogic™ model". Rename `Main` class to `Model`. We will consider each model time unit as a minute. Specify that the model is executed in *Real time* mode, and *Model time units per second* is 1, that means that one minute (one model time unit) is simulated in one second in real time mode (you can also execute the model in virtual time with the fastest speed).

3.2 Basic shop floor model

Now we will create a basic model, consisting of a part source, a conveyor and a processing station. Later on, we will add an assembly station with upstream and downstream conveyors.

Now we will create a class representing a processing station.

► Create a processing station



Class name: Station

Base class:

Parameters:

Name	Type
rate	real (dynamic)
queueSize	integer

☐ Exclude from build

☑ Show name

☑ Show object rectangle

☑ Public (exported from library)

❶ Create new class `Station`

❷ Click *Port* and create a port named `in`

❸ Click *Port* and create a port named `out`

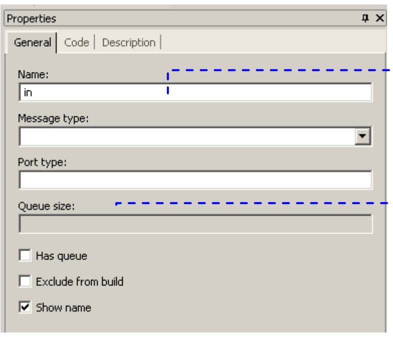
❹ Create class parameters

❺ Delay

❻ Queue

❶ To create a new class, click the *New Active Object Class* toolbar button. Name the new class `Station`. Class structure diagram opens automatically.

❷ In the structure diagram, create `in` port.



Name: in

Message type:

Port type:

Queue size:

☐ Has queue

☐ Exclude from build

☑ Show name

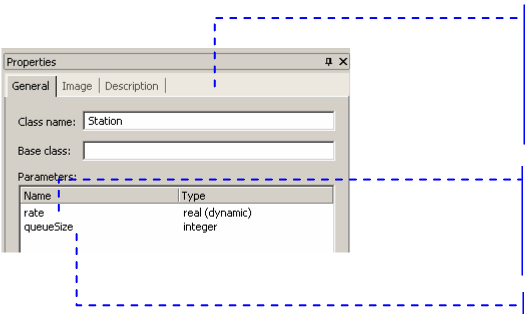
Ⓐ Name the port `in`

Ⓑ Leave all properties by default

Ⓐ To edit the port name, you can click the port and press F2.

❸ Create `out` port. Leave all properties by default.

❹ Parameters allow us to define processing time and queue size individually for each station.



Class name: Station

Base class:

Parameters:

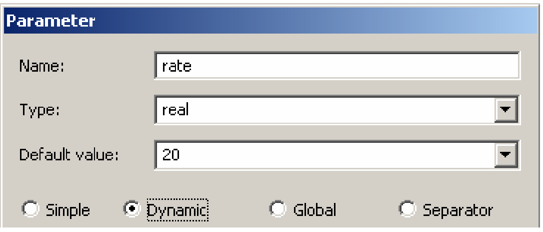
Name	Type
rate	real (dynamic)
queueSize	integer

Ⓐ Click `Station` in the project tree. The Properties window will show the class properties.

Ⓑ Define `rate` parameter for station rate (items per minute)

Ⓒ Define `queueSize` parameter for output queue size

Ⓑ Select `real` as parameter type, and specify this parameter as *Dynamic* in the parameter creation dialog. Type the default value of 20.



Name: rate

Type: real

Default value: 20

○ Simple ● Dynamic ○ Global ○ Separator

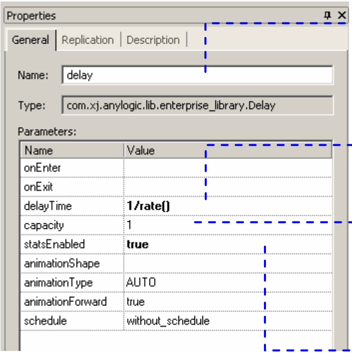
By selecting *Dynamic* we specify that the parameter is dynamic; that is, the expression you type for the parameter value for an object of the Station class is reevaluated each time the parameter is assessed. Therefore, you can specify dynamically changing expressions for the parameter value, like random numbers, and each parameter assessment will return the current expression value; for example, next random number.

In our case, we define station rate to be a dynamic parameter, because we suppose that rate can be changing over time.

Ⓒ Queue size should be defined and used only once and need not be reevaluated. Therefore, we use *Simple* parameter with the type `integer`. Specify the default value of 2.

➡ To assess dynamic parameters in your model, use parentheses that depict the function call; for instance, `rate()`. To assess simple parameters, just type a parameter name; for instance, `queueSize`.

❺ Create a Delay object and set the following properties:



Properties

General | Replication | Description

Name: delay

Type: com.xj.anylogic.lib.enterprise_library.Delay

Parameters:

Name	Value
onEnter	
onExit	
delayTime	**1/rate()**
capacity	1
statsEnabled	**true**
animationShape	
animationType	AUTO
animationForward	true
schedule	without_schedule

Ⓐ Leave default `delay` name

Ⓑ Type `1/rate()` for delay time

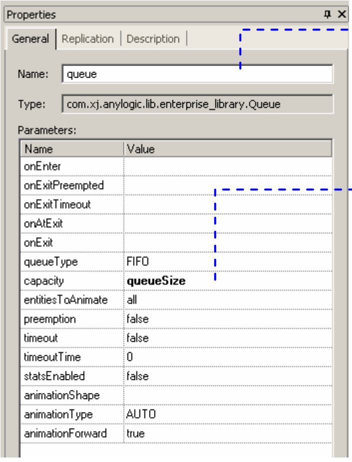
Ⓒ Capacity is 1 by default

Ⓓ Enable statistics collection

Ⓑ Since station rate represents how many items are processed in a time unit, this expression will evaluate processing time.

❻ We need the buffer object representing a "heap" of parts because a situation when the part cannot be placed on the conveyor immediately can occur.

Set up the following object properties:



Properties

General | Replication | Description

Name: queue

Type: com.xj.anylogic.lib.enterprise_library.Queue

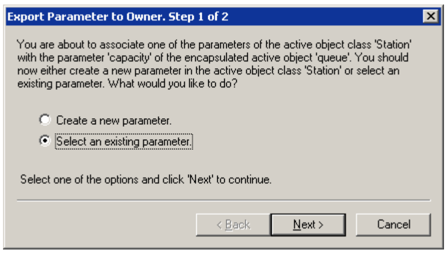
Parameters:

Name	Value
onEnter	
onExitPreempted	
onExitTimeout	
onAtExit	
onExit	
queueType	FIFO
capacity	**queueSize**
entitiesToAnimate	all
preemption	false
timeout	false
timeoutTime	0
statsEnabled	false
animationShape	
animationType	AUTO
animationForward	true

Ⓐ Leave default `queue` name

Ⓑ Select `queueSize` parameter for queue size

❸ To link queue size parameter to class parameter, right-click the parameter row, choose *Export to Owner* from the shortcut menu, then *Select existing parameter*.



Export Parameter to Owner. Step 1 of 2

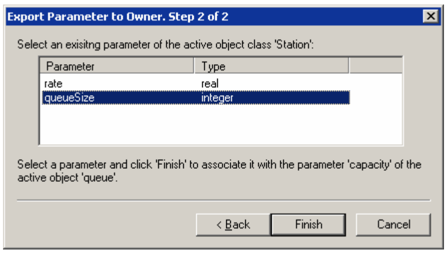
You are about to associate one of the parameters of the active object class 'Station' with the parameter 'capacity' of the encapsulated active object 'queue'. You should now either create a new parameter in the active object class 'Station' or select an existing parameter. What would you like to do?

○ Create a new parameter.

● Select an existing parameter.

Select one of the options and click 'Next' to continue.

Then select `queueSize` class parameter from the list.



Export Parameter to Owner. Step 2 of 2

Select an exisitng parameter of the active object class 'Station':

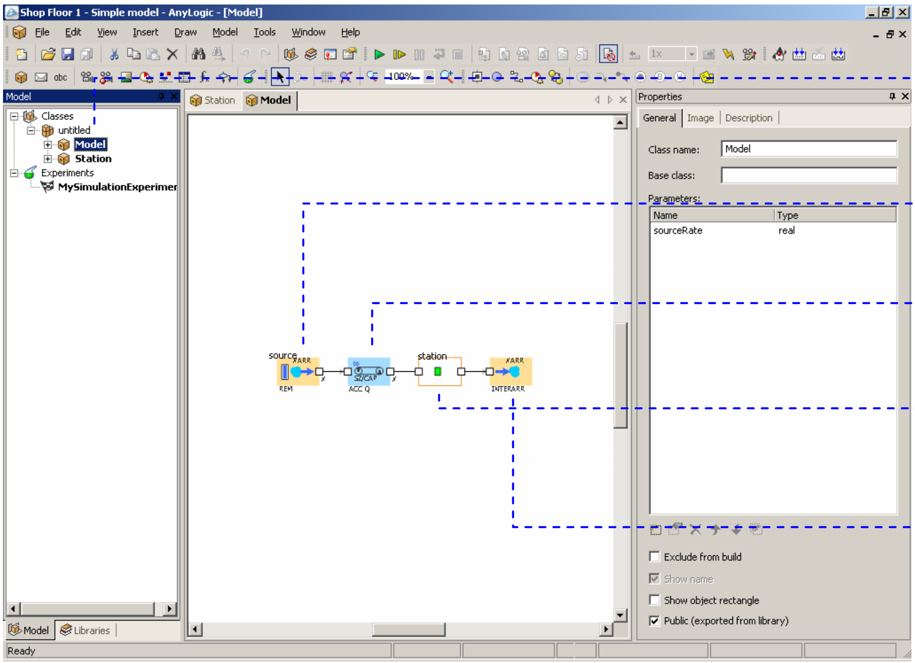
Parameter	Type
rate	real
queueSize	integer

Select a parameter and click 'Finish' to associate it with the parameter 'capacity' of the active object 'queue'.

❼ All the objects in the Enterprise Library have a unique appearance in the diagram due to the distinguishing picture. You can create an icon for your classes as well. To create an icon, right-click `Station` class in the tree and select *New Icon*. You can draw an icon in the diagram appeared.

Now we will create a flowchart of our basic shop floor model.

► Create the flowchart



Properties

General | Image | Description

Class name: Model

Base class:

Parameters:

Name	Type
sourceRate	real

□ Exclude from build

☑ Show name

□ Show object rectangle

☑ Public (exported from library)

❶ Double-click Model

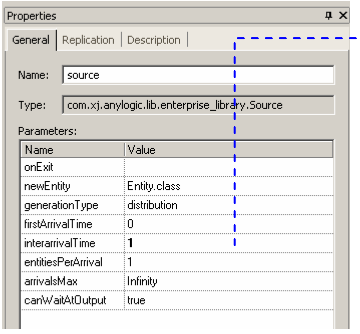
❷ Source

❸ Conveyor

❹ Station

❺ Sink

❷ Set up the following properties for the source object:



Properties

General | Replication | Description

Name: source

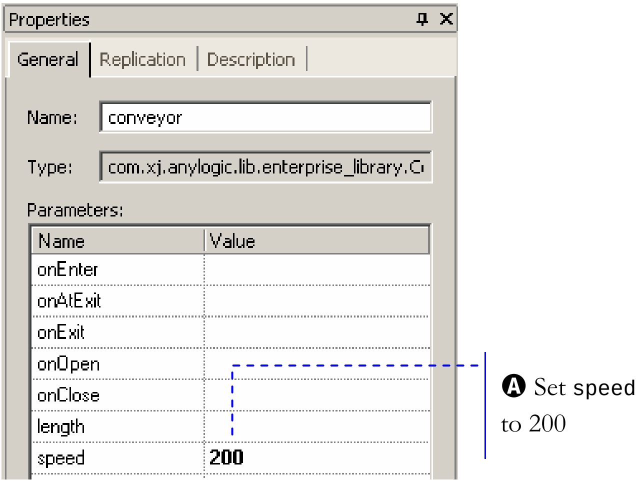
Type: com.xj.anylogic.lib.enterprise_library.Source

Parameters:

Name	Value
onExit	
newEntity	Entity.class
generationType	distribution
firstArrivalTime	0
interarrivalTime	**1**
entitiesPerArrival	1
arrivalsMax	Infinity
canWaitAtOutput	true

Ⓐ Set interarrival time to 1

❸ Set up the following properties for the conveyor:



Name: conveyor

Type: com.xj.anylogic.lib.enterprise_library.C

Parameters:

Name	Value
onEnter	
onAtExit	
onExit	
onOpen	
onClose	
length	
speed	**200**

Ⓐ Set speed to 200

❹ Leave all properties of the station object by default.

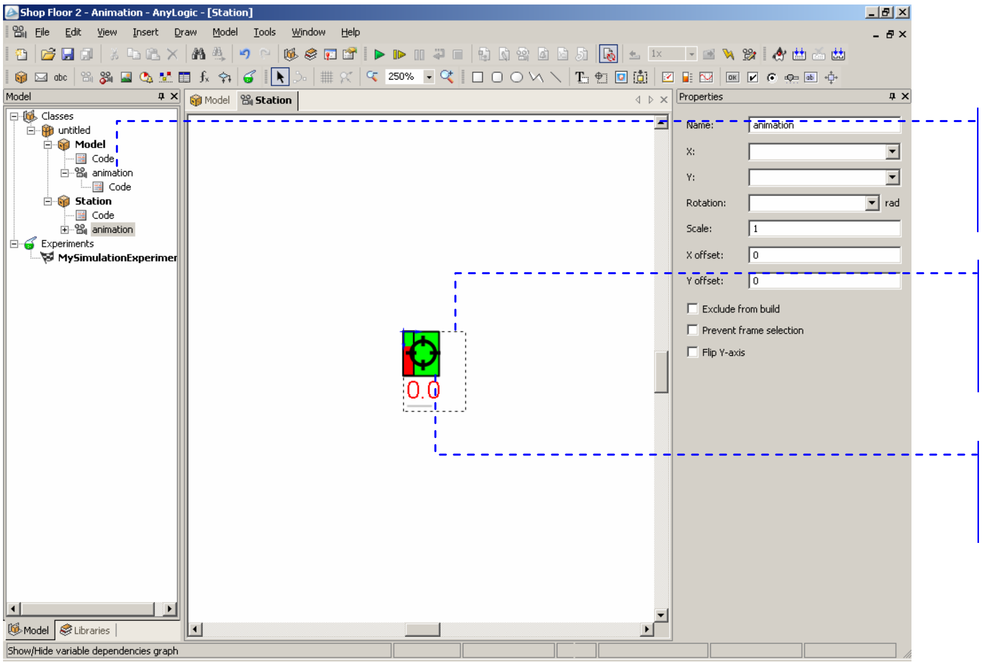
❺ Leave all properties of the Sink object by default.

Click *Run* ▶ to start the model.

The reference model for this point is Examples \ Enterprise Library Tutorial Models \ Shop Floor 1 - Simple model.alp.

3.3 Animating the model

► Create station animation


Shop Floor 2 - Animation - AnyLogic - [Station]
File Edit View Insert Draw Model Tools Window Help
250%
Model
Classes
untitled
Model
Code
animation
Code
Station
Code
animation
Experiments
MySimulationExperiment
Model Libraries
Model Station
0.0
Show/Hide variable dependencies graph

Properties

Name: animation

X:

Y:

Rotation: rad

Scale: 1

X offset: 0

Y offset: 0

□ Exclude from build

□ Prevent frame selection

□ Flip Y-axis

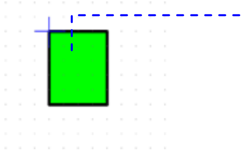
❶Create animation diagram

❷ Resize and move dashed frame

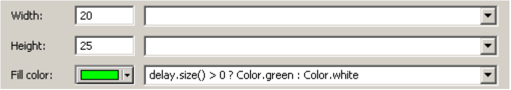
❸ Create station picture

❷ Resize the frame to be about 30 by 30 pixels and put the left top frame corner onto the crosshair origin.

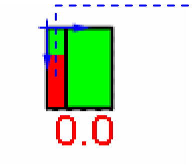
❸ Draw the station picture.



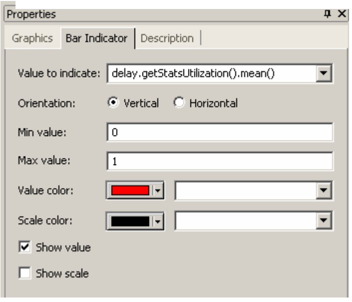
Ⓐ Draw a rectangle about 20 by 25 pixels and set its run-time color



Width:	20	
Height:	25	
Fill color:	🟩	delay.size() > 0 ? Color.green : Color.white



Ⓑ Draw a utilization indicator using a bar indicator. Set transparent fill color, turn off scale, set max value to 1 and min value to 0, and set value to indicate `delay.getStatsUtilization().mean()`



Properties

Graphics | Bar Indicator | Description

Value to indicate: delay.getStatsUtilization().mean()

Orientation: ● Vertical ○ Horizontal

Min value: 0

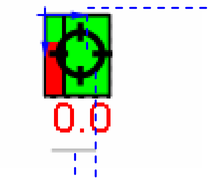
Max value: 1

Value color: 🟥

Scale color: ⬛

☑ Show value

☐ Show scale

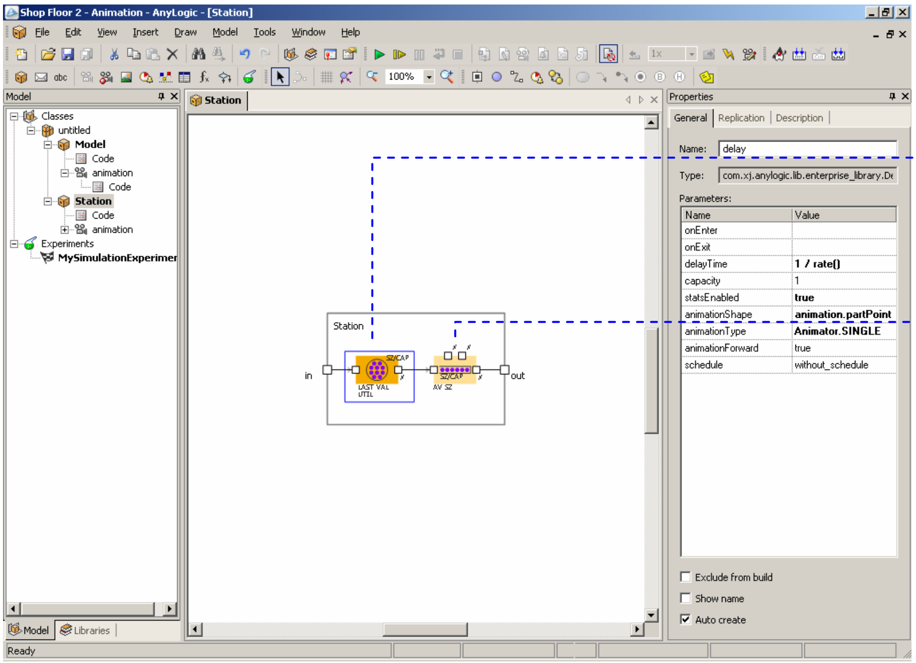


Ⓒ Click *Pivot* and place a point to indicate the position of the part being processed

Ⓓ Name the point `partPoint`

Ⓔ Create a polyline to display parts in the output queue. Name polyline `queuePath` and hide it at run time by setting *Visible* to `false`

► Animate the station



Properties

General | Replication | Description

Name: delay

Type: com.xj.anylogic.lib.enterprise_library.De

Parameters:

Name	Value
onEnter	
onExit	
delayTime	1 / rate()
capacity	1
statsEnabled	true
animationShape	animation.partPoint
animationType	Animator.SINGLE
animationForward	true
schedule	without_schedule

☐ Exclude from build

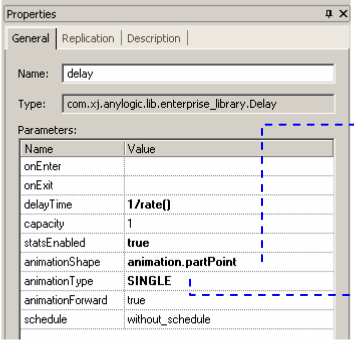
☐ Show name

☑ Auto create

❶ Animate the Delay object

❷ Animate the Queue object

❶ Set up the following animation properties for the Delay object:



Properties

General | Replication | Description

Name: delay

Type: com.xj.anylogic.lib.enterprise_library.Delay

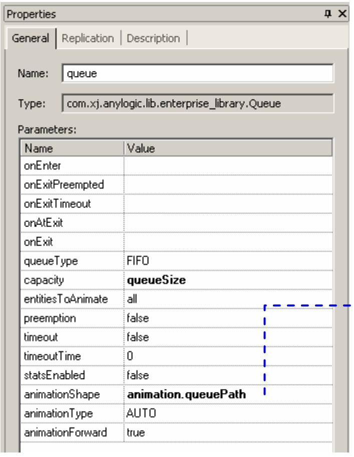
Parameters:

Name	Value
onEnter	
onExit	
delayTime	1/rate()
capacity	1
statsEnabled	true
animationShape	animation.partPoint
animationType	SINGLE
animationForward	true
schedule	without_schedule

Ⓐ Select `animation.partPoint` animation shape

Ⓑ Set animation style to SINGLE

❷ Set up the following animation property for the Queue object:



Properties

General | Replication | Description

Name: queue

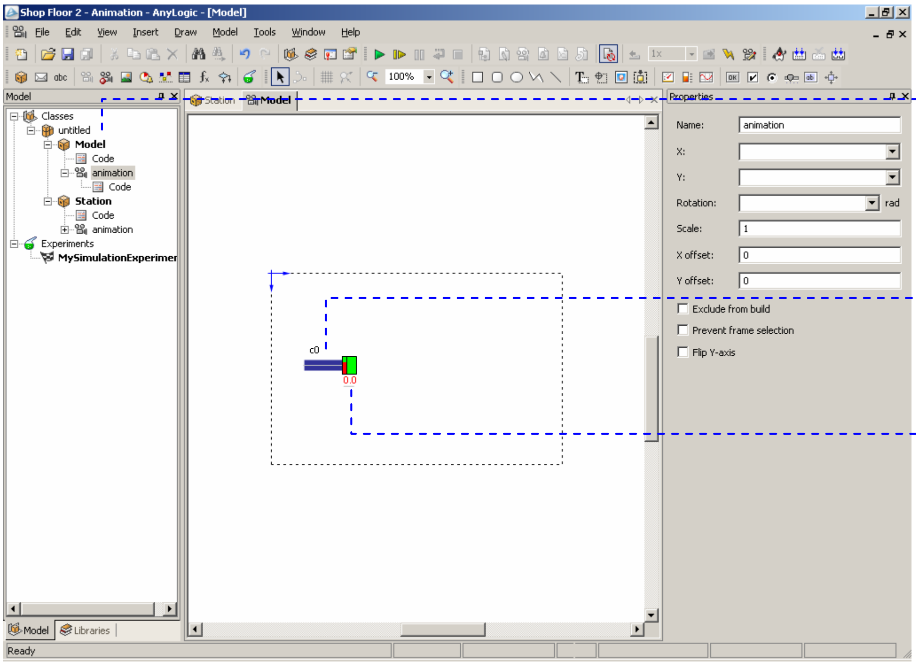
Type: com.xj.anylogic.lib.enterprise_library.Queue

Parameters:

Name	Value
onEnter	
onExitPreempted	
onExitTimeout	
onAtExit	
onExit	
queueType	FIFO
capacity	**queueSize**
entitiesToAnimate	all
preemption	false
timeout	false
timeoutTime	0
statsEnabled	false
animationShape	**animation.queuePath**
animationType	AUTO
animationForward	true

Ⓐ Set animation shape to `animation.queuePath`

► Create the model animation



Properties

Name: animation

X:

Y:

Rotation: rad

Scale: 1

X offset: 0

Y offset: 0

☐ Exclude from build

☐ Prevent frame selection

☐ Flip Y-axis

❶ Right-click Model and create new animation

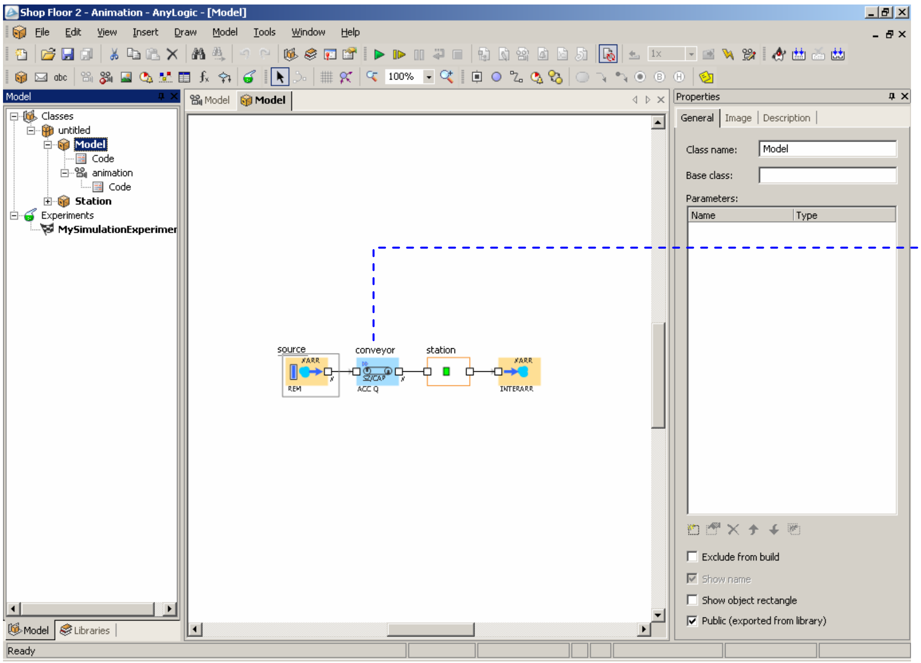
❷ Draw a conveyor

❸ Move the station animation here

❷ Draw an upstream conveyor. First, draw a thick blue polyline as the conveyor background, set the width of 15. Second, draw a thin white polyline (set the width of 1) over the blue one; that will represent the part movement path. Name the second (thin and white) polyline `c0`.

❸ The encapsulated animation of the station should automatically appear on the model animation. If not, then you should add encapsulated animation on the animation diagram by yourself and choose `Station` in its *Object* property.

▶ Animate the model



Class name: Model

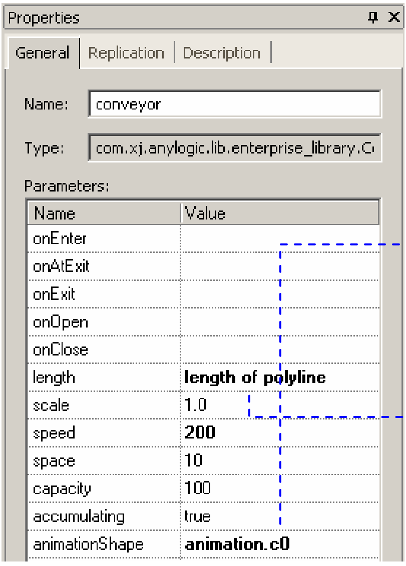
Base class:

Parameters:

Name	Type

❶ Set up conveyor animation

❶ Set up the conveyor properties:



Name: conveyor

Type: com.xj.anylogic.lib.enterprise_library.C

Parameters:

Name	Value
onEnter	
onAtExit	
onExit	
onOpen	
onClose	
length	**length of polyline**
scale	1.0
speed	**200**
space	10
capacity	100
accumulating	true
animationShape	**animation.c0**

🅐 Set up the conveyor animation path to `c0`

🅑 Select ”length of polyline” from the drop-down list of `length` parameter

🅑 Polyline length will be measured (in pixels) and set up as the conveyor length. Note that `scale` parameter can be used to convert pixel length to, for example, feet.

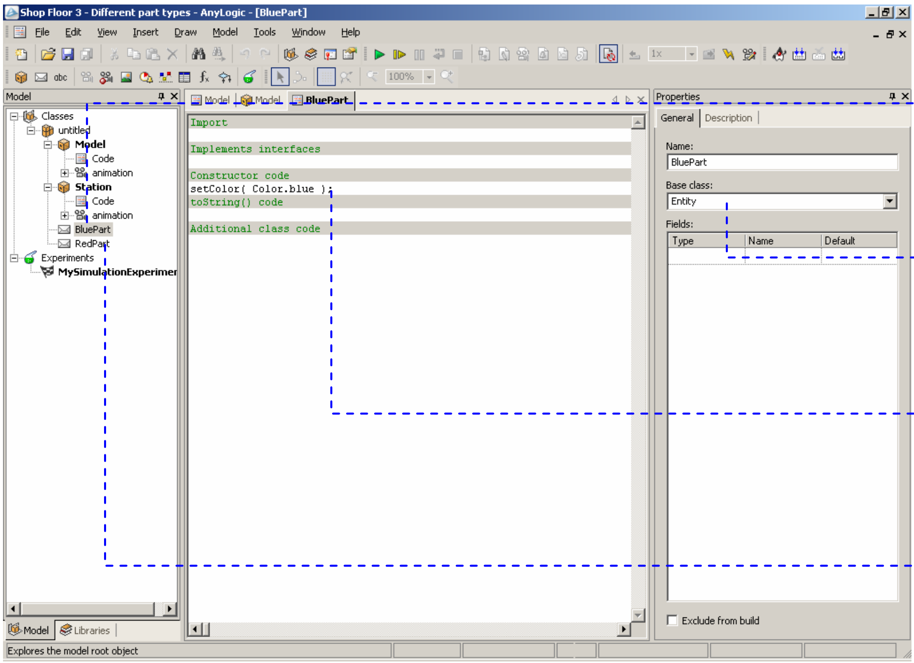
Click *Run* ▶ to start the model.

↘ The reference model for this point is Examples \ Enterprise Library Tutorial Models \ Shop Floor 2 - Animation.alp.

3.4 Different entity types

The factory works with two part types. We will visually depict them as red and blue parts. Now we will create different entity types for each kind of parts, and set up the visual appearance.

► Create classes for different part types



```
Import

Implements interfaces

Constructor code
setColor( Color.blue );
toString() code

Additional class code
```

Name: BluePart

Base class: Entity

Fields:

Type	Name	Default

❶ Create `BluePart` message class

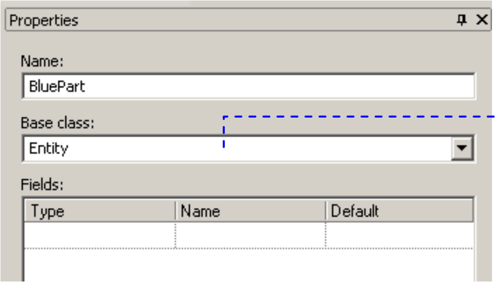
❷ Derive the message from Entity

❸ Set up entity appearance in the constructor code

❹ The same applies to the second message class

❶ To create a new message class, click the *New Message Class* ✉ toolbar button. Name the message class `BluePart`.

❷



Name: BluePart

Base class: Entity

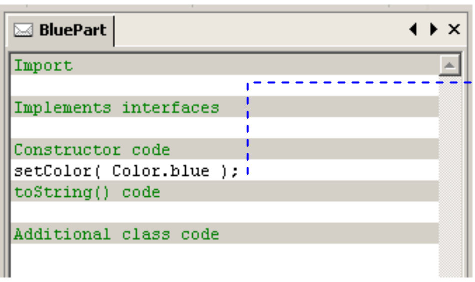
Fields:

Type	Name	Default

Ⓐ Select `Entity` as the base class for the message created

Ⓐ `Entity` class is defined in the Enterprise Library.

❸ Change appearance using constructor code.



BluePart

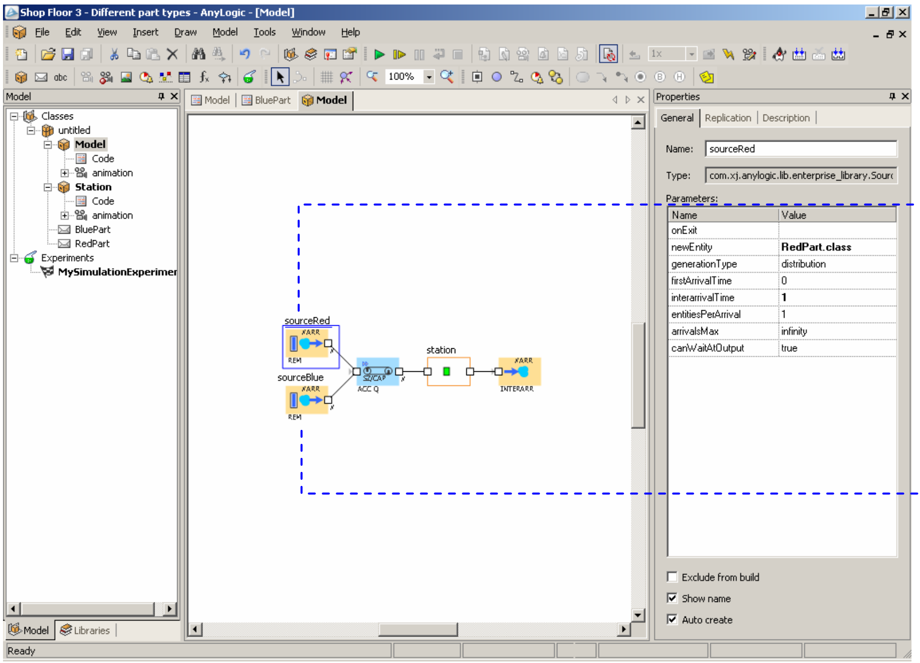
```
Import

Implements interfaces

Constructor code
setColor( Color.blue );
toString() code

Additional class code
```

Ⓐ Set blue entity color

Ⓐ To set the entity color, we use the `setColor()` function of the `Entity` class. `Entity` message class has certain functions and certain fields, like `priority` field. For more information about `Entity` class, please refer to *Enterprise Library Reference Guide*.

❹ Create `RedPart` message doing the same as for `BluePart` except setting color to red by typing `setColor(Color.red);`.

► Create the sources of different part types



Shop Floor 3 - Different part types - AnyLogic - [Model]

Name: sourceRed

Type: com.xj.anylogic.lib.enterprise_library.Sourc

Parameters:

Name	Value
onExit	
newEntity	**RedPart.class**
generationType	distribution
firstArrivalTime	0
interarrivalTime	**1**
entitiesPerArrival	1
arrivalsMax	infinity
canWaitAtOutput	true

☐ Exclude from build

☑ Show name

☑ Auto create

❶ Change the existing source type

❷ Add the second source

❶ Select `RedPart` as `newEntity` parameter for this source object. Keep interarrival time of 1.

❷ Select `BluePart` as `newEntity` parameter for this source object.

Set interarrival time of 2.

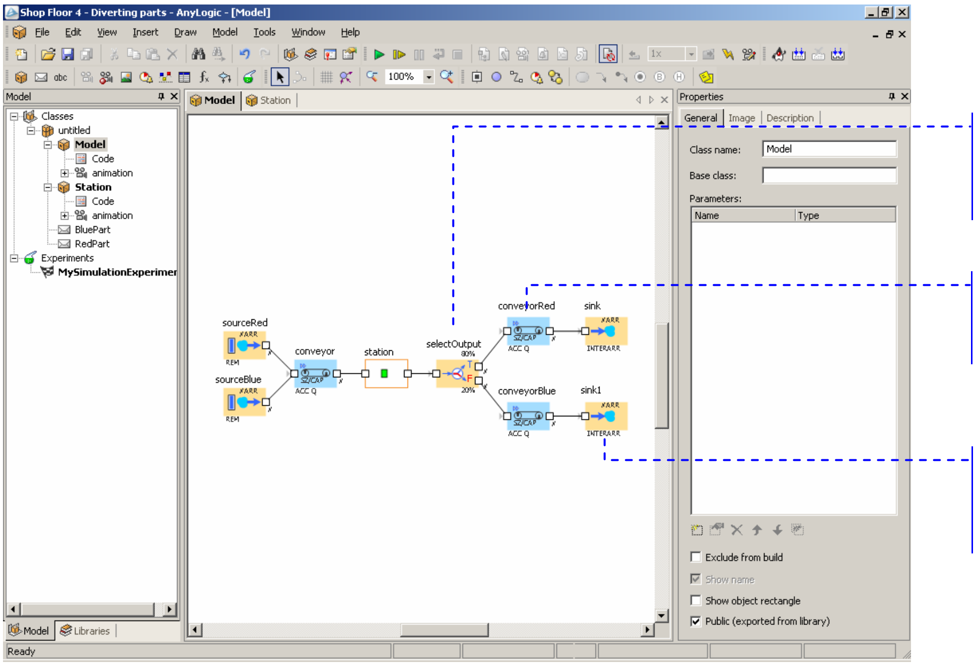
Click *Run* ▶ to start the model. You will see red and blue parts moving along the conveyors.

↘ The reference model for this point is Examples \ Enterprise Library Tutorial Models \ Shop Floor 3 - Different part types.alp.

3.5 Diverting parts based on the part type

The assembly station will produce a product of two parts: red and blue. Therefore, we need to sort the parts first and assemble the product from the parts of different types. Now we will model how the parts of different types are diverted to different conveyors.

▶ Modify the flowchart



Shop Floor 4 - Diverting parts - AnyLogic - [Model]

Class name: Model

Base class:

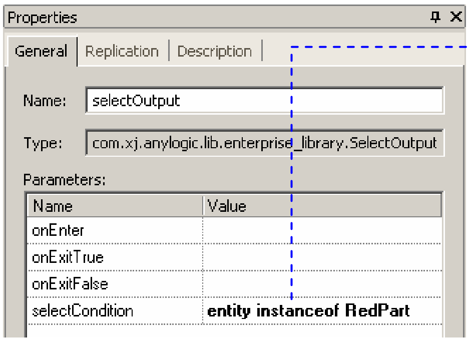
□ Exclude from build
☑ Show name
□ Show object rectangle
☑ Public (exported from library)

❶ Add SelectOutput

❷ Add two conveyors

❷ Add the second Sink

❶ Add SelectOutput object and set up the following properties:



Name: selectOutput

Type: com.xj.anylogic.lib.enterprise_library.SelectOutput

Parameters:

Name	Value
onEnter	
onExitTrue	
onExitFalse	
selectCondition	**entity instanceof RedPart**

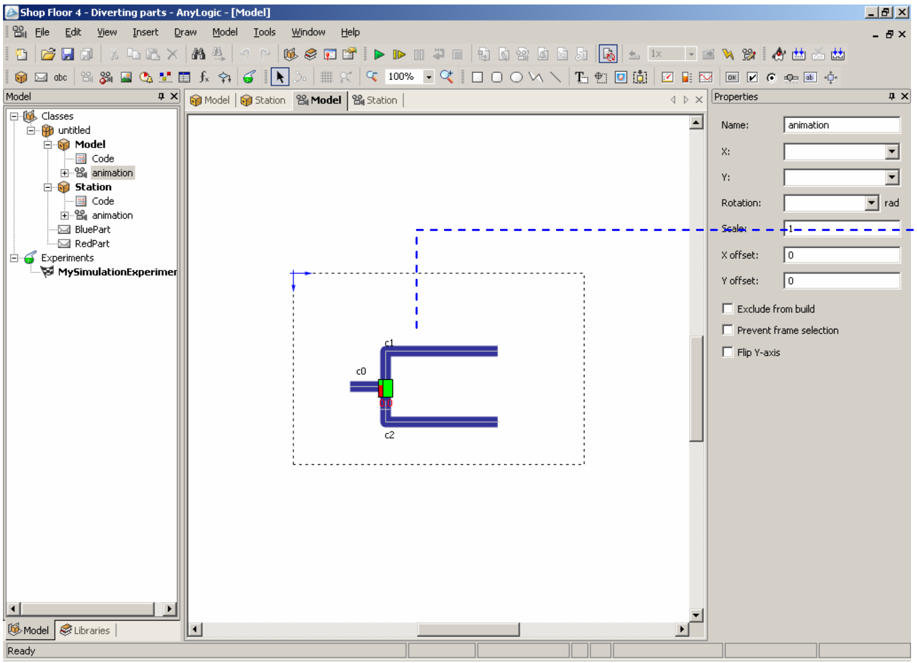
Ⓐ Type `entity instanceof RedPart` as the route select condition

Ⓐ This is a user-defined routing condition. It determines the type of the entity about to exit and diverts red parts to the upper conveyor, and blue parts to the lower. Current entity is accessed as `entity`. Its type is determined by the `instanceof` Java™ operator. If the condition evaluates to `true`, the upper conveyor is taken, otherwise the lower.

❷ Name the upper conveyor `conveyorRed` and the lower conveyor `conveyorBlue`. Set the conveyor speeds of 300.

❷ Add the second Sink object. Leave object properties by default.

► Draw conveyors in the animation



Name: animation

X:

Y:

Rotation: rad

Scale: 1

X offset: 0

Y offset: 0

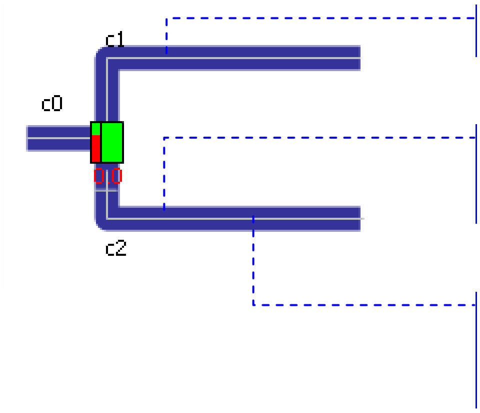
□ Exclude from build

□ Prevent frame selection

□ Flip Y-axis

❶ Draw `c1` and `c2` conveying paths

❷ Draw the animation as shown in the following picture:

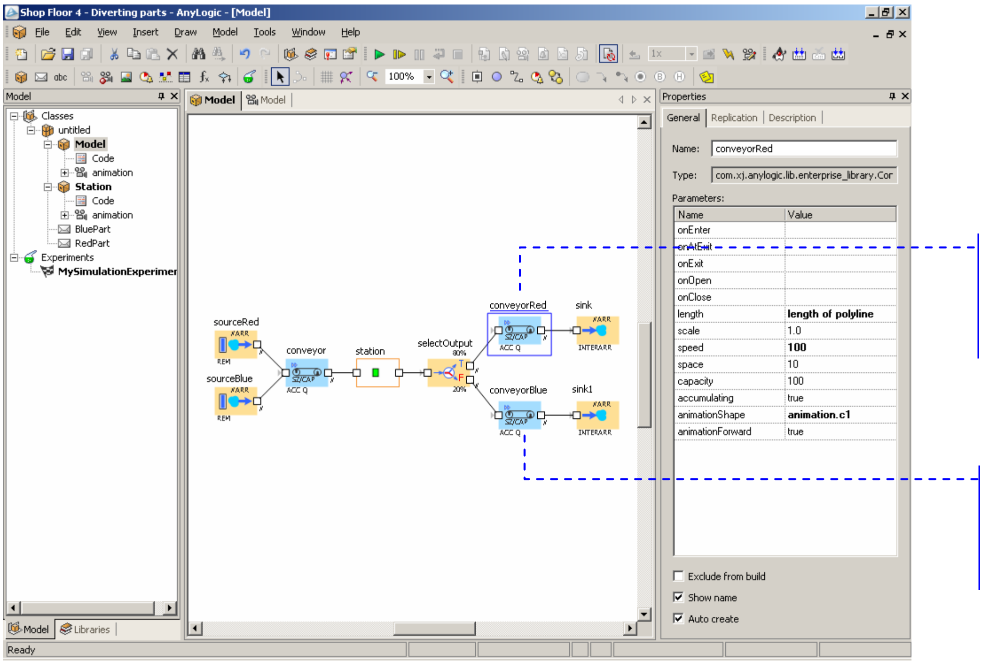


🅐 Draw `c1` conveyor

🅑 Draw c2 conveyor as well

🅒 To create the proper direction, draw the conveyors from left to right

► Animate the conveyors



Properties

General | Replication | Description

Name: conveyorRed

Type: com.xj.anylogic.lib.enterprise_library.Cor

Parameters:

Name	Value
onEnter	
onAtExit	
onExit	
onOpen	
onClose	
length	**length of polyline**
scale	1.0
speed	**100**
space	10
capacity	100
accumulating	true
animationShape	**animation.c1**
animationForward	true

☐ Exclude from build

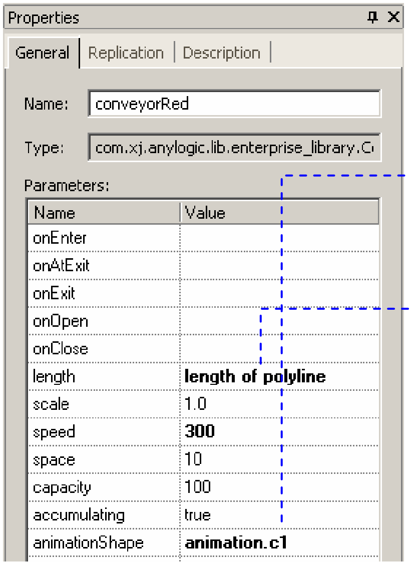
☑ Show name

☑ Auto create

❶ Set up conveyor animation

❷ Set up conveyor animation

❶ Set up the conveyor properties:



Properties

General | Replication | Description

Name: conveyorRed

Type: com.xj.anylogic.lib.enterprise_library.C

Parameters:

Name	Value
onEnter	
onAtExit	
onExit	
onOpen	
onClose	
length	**length of polyline**
scale	1.0
speed	**300**
space	10
capacity	100
accumulating	true
animationShape	**animation.c1**

Ⓐ Set up the conveyor animation path to `c1`

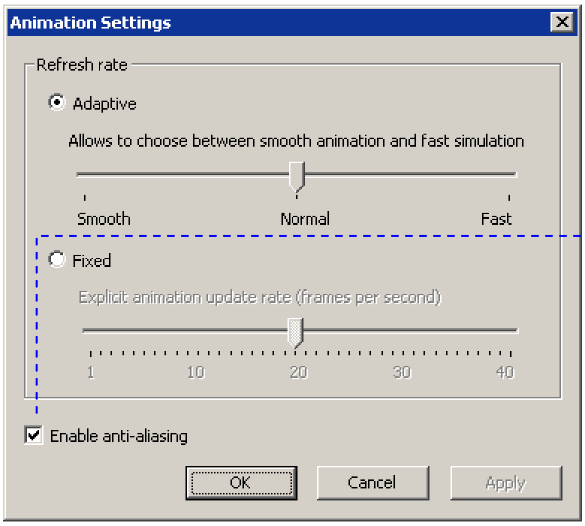
Ⓑ Set up the conveyor length to be measured automatically

❷ Set up the second conveyor in the similar way except set up animation path to `c2`.

Now run the model by clicking *Run* .

➘ The reference model for this point is [Examples \ Enterprise Library Tutorial Models \ Shop Floor 4 - Diverting parts.alp]().

➲ To get a better view at run time, you can click *Animation settings* toolbar button and select *Enable anti-aliasing* option.



Animation Settings

Refresh rate

● Adaptive

Allows to choose between smooth animation and fast simulation

○ Fixed

Explicit animation update rate (frames per second)

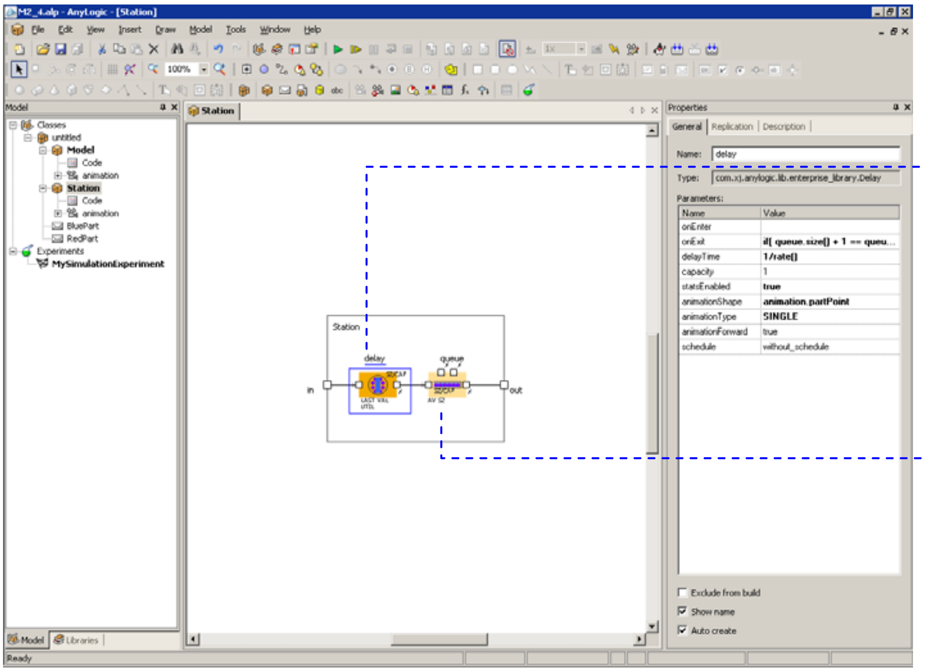
☑ Enable anti-aliasing

Ⓐ Enable anti-aliasing to get a better view

3.6 Blocking the station

Whenever the downstream conveyor cannot accept entities, the station should stop. This is known as a block condition. Now we will introduce station block ability.

▶ Add station block ability



Name: delay

Type: com.xj.anylogic.lib.enterprise_library.Delay

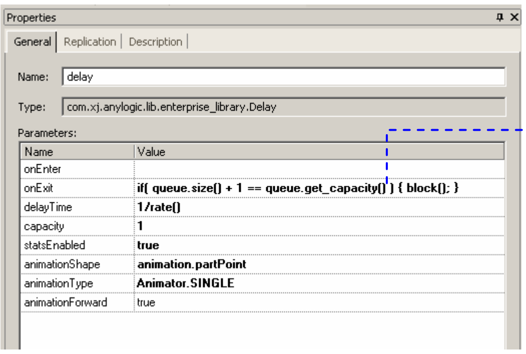
Parameters:

Name	Value
onEnter	
onExit	if(queue.size() + 1 == queu...
delayTime	1/rate()
capacity	1
statsEnabled	true
animationShape	animation.partPoint
animationType	SINGLE
animationForward	true
schedule	without_schedule

❶ Set up the Delay object properties

❷ Set up the Queue object properties

❶ Block the Delay object whenever the queue becomes full.



Name: delay

Type: com.xj.anylogic.lib.enterprise_library.Delay

Parameters:

Name	Value
onEnter	
onExit	if(queue.size() + 1 == queue.get_capacity()) { block(); }
delayTime	1/rate()
capacity	1
statsEnabled	true
animationShape	animation.partPoint
animationType	Animator.SINGLE
animationForward	true

🅐 Type code shown below

🅐 Type the following code as “on exit” reaction:

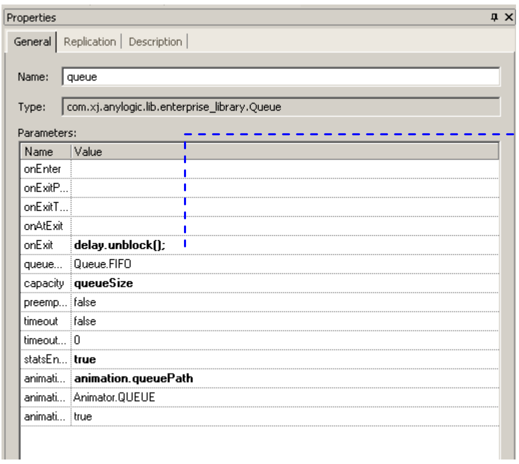
```
if( queue.size() + 1 == queue.get_capacity() ) block();
```

This code compares the current queue size obtained by `queue.size()` function with the capacity of the queue; the capacity property is obtained by `get_capacity()` function. Plus one stands for the entity exiting the Delay object.

➲ In AnyLogic™, you can obtain the property value or set the property value using functions `get_x()` or `set_x()`, where `x` is the name of the property you need.

In our example, if the entity exits the Delay object and the queue will be full with counting entity, the Delay object blocks itself by calling its `block()` function. Blocking means that the Delay object will not accept any entities for processing until its `unblock()` function is called.

❷ Unblock the Delay object once an entity exits the queue.



Name: queue

Type: com.xj.anylogic.lib.enterprise_library.Queue

Parameters:

Name	Value
onEnter	
onExitP...	
onExitT...	
onAtExit	
onExit	**delay.unblock();**
queue...	Queue.FIFO
capacity	**queueSize**
preemp...	false
timeout	false
timeout...	0
statsEn...	**true**
animati...	**animation.queuePath**
animati...	Animator.QUEUE
animati...	true

🅐 Type `delay.unblock()` as “on exit” code

🅐 We unblock the Delay object when an entity exits queue, and the queue now has a free place.

Note that we always call the `unblock()` function, even if the Delay object is not blocked. It is valid to invoke the `unblock()` function for the object that is not blocked. However, if you want to know if an object is blocked, you can call the object’s `blocked()` function.

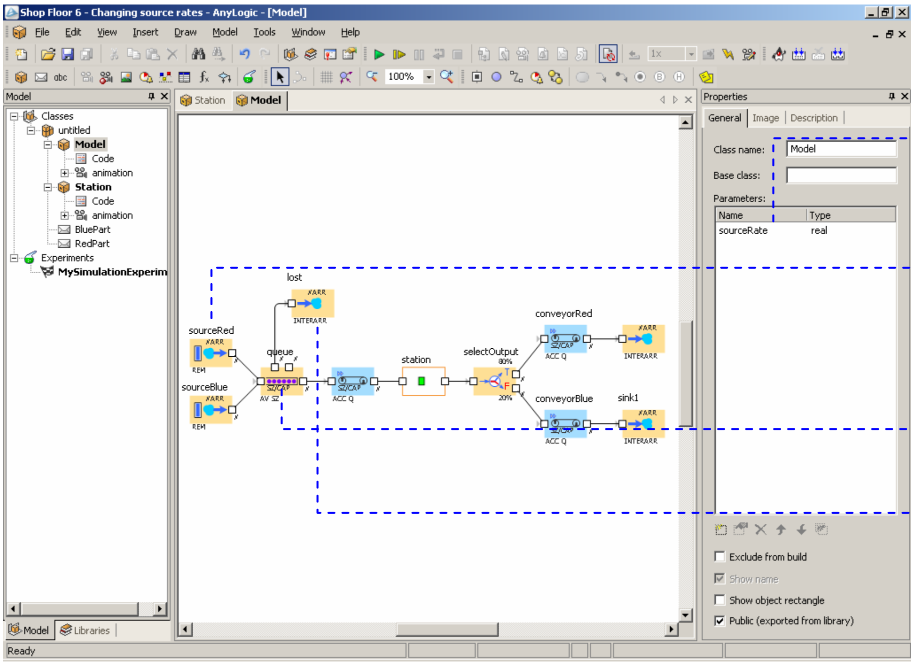
Now, having created the block behavior of stations, we can run the model again. Click *Run* ▶ to start the model. Now the station will be blocked when the downstream conveyor cannot accept entities.

➘ The reference model for this point is Examples \ Enterprise Library Tutorial Models \ Shop Floor 5 - Blocking the station.alp.

3.7 Changing source rates interactively

Now we will make the source rate changeable. We will introduce the corresponding control element in the animation to change the source rate interactively.

► Set source rate



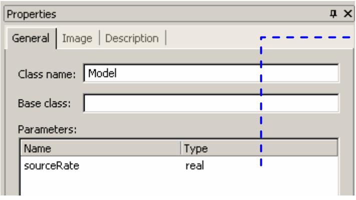
❶ Add a class parameter

❷ Modify source properties

❸ Add Queue

❹ Add Sink to collect lost entities

❶ Add the class parameter for the source rate. Click `Model` item in the tree, and create the following parameter:



Class name: Model

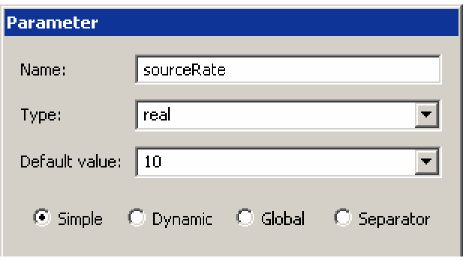
Base class:

Parameters:

Name	Type
sourceRate	real

Ⓐ `sourceRate` is the model parameter and will be changed by the animation slider we will create later on

Ⓐ Select `real` for the parameter type, name the parameter `sourceRate` and set the default value to 10.


Parameter

Name: sourceRate

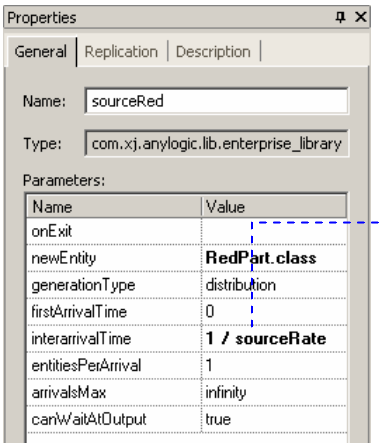
Type: real

Default value: 10

● Simple ○ Dynamic ○ Global ○ Separator

❷ For both the Source objects, set the interarrival time.

For the upper Source object:



Name: sourceRed

Type: com.xj.anylogic.lib.enterprise_library

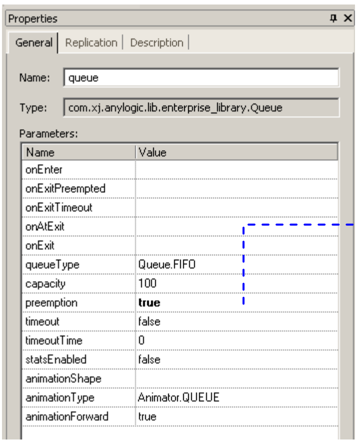
Parameters:

Name	Value
onExit	
newEntity	**RedPart.class**
generationType	distribution
firstArrivalTime	0
interarrivalTime	**1 / sourceRate**
entitiesPerArrival	1
arrivalsMax	infinity
canWaitAtOutput	true

🅐 Set interarrival time to `1/sourceRate`

Set the same property for the lower Source object.

❸ We add a queue to store the entities arrived until they can be taken on the conveyor. Set the following queue properties:



Name: queue

Type: com.xj.anylogic.lib.enterprise_library.Queue

Parameters:

Name	Value
onEnter	
onExitPreempted	
onExitTimeout	
onAtExit	
onExit	
queueType	Queue.FIFO
capacity	100
preemption	**true**
timeout	false
timeoutTime	0
statsEnabled	false
animationShape	
animationType	Animator.QUEUE
animationForward	true

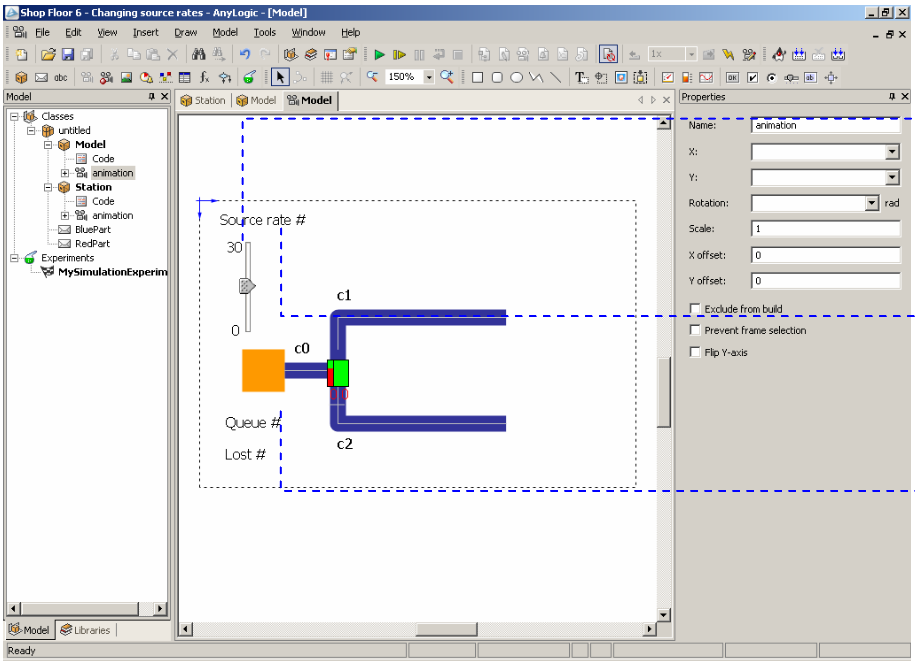
🅐 Set `preemption` to `true`

🅐 By enabling preemption, we specify that if the queue capacity becomes exceeded, the arrived entity exits via `outputPreempted` port on the top of the queue object. To accept those entities, we will place a Sink object. For more information about preemption, please refer to *Enterprise Library Reference Guide.*

❹ Place a Sink object and connect it to the `outputPreempted` queue port; that is, the left port on the top of the Queue object.

Name this Sink object `lost`.

► Add rate controls to the animation



Properties

Name: animation

X:

Y:

Rotation: rad

Scale: 1

X offset: 0

Y offset: 0

☐ Exclude from build

☐ Prevent frame selection

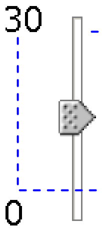
☐ Flip Y-axis

❶ Add source rate slider

❷ Add rate note

❸ Add queue area rectangle, queue size and lost note

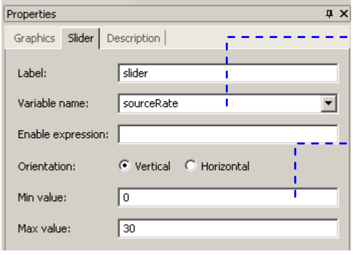
❶



🅐 Place a slider

🅑 Click *Text* T and place min and max notes

For the slider created, set up the following properties:



Label: slider

Variable name: sourceRate

Enable expression:

Orientation: ● Vertical ○ Horizontal

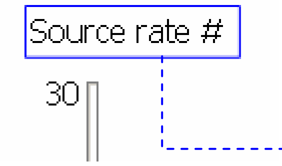
Min value: 0

Max value: 30

🅐 Select `sourceRate` parameter

🅑 Set min value to 0 and max value to 30

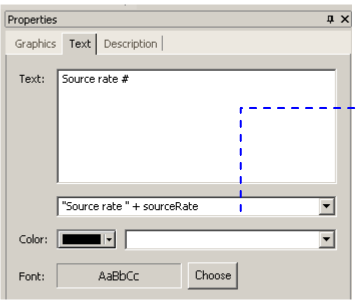
❷



A Click *Text* and add a rate note:

B Type "Source rate #" during text note creation as design-time value

To show the actual source rate value at run time, specify the text note run-time property:

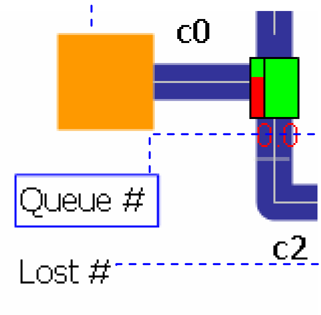


Text: Source rate #

"Source rate " + sourceRate

A Type `"Source rate" + sourceRate` as the design-time value

❸ Using the same approach, add text notes for the current queue size and lost entities counter:



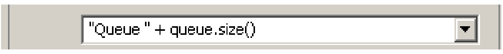
A Add a queue area rectangle

B Add a text note for current queue size

C Add a text note for lost entities counter

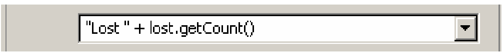
A Name the rectangle `queueArea`. Choose the fill color and line style you like, for example turn off the lines and set orange fill color.

B Type "Queue #" as design-time text, and specify the following run-time text:



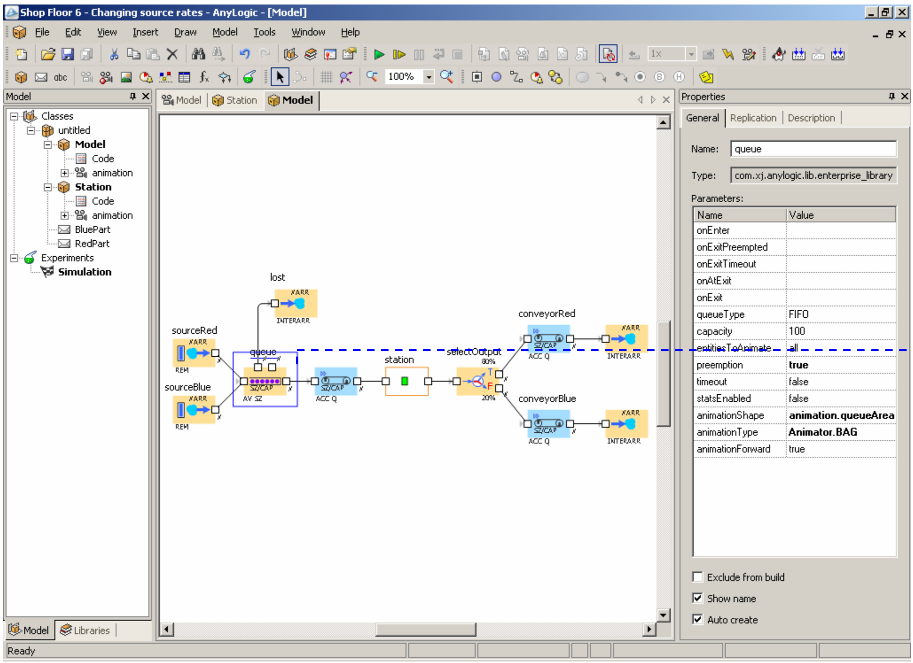
"Queue " + queue.size()

C Type "Lost #" as design-time text, and specify the following run-time text:



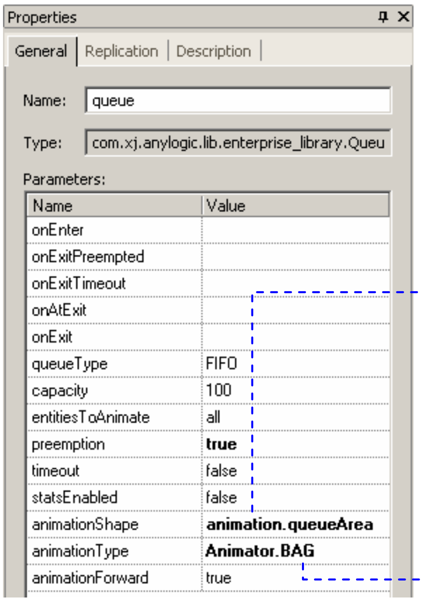
"Lost " + lost.getCount()

▶ Animate the queue



❶ Set up queue animation properties

❶ For the queue, set the following animation properties:



Properties

General | Replication | Description

Name: queue

Type: com.xj.anylogic.lib.enterprise_library.Queu

Parameters:

Name	Value
onEnter	
onExitPreempted	
onExitTimeout	
onAtExit	
onExit	
queueType	FIFO
capacity	100
entitiesToAnimate	all
preemption	**true**
timeout	false
statsEnabled	false
animationShape	**animation.queueArea**
animationType	**Animator.BAG**
animationForward	true

Ⓐ Select `queueArea` as the animation shape

Ⓑ Select BAG animation style

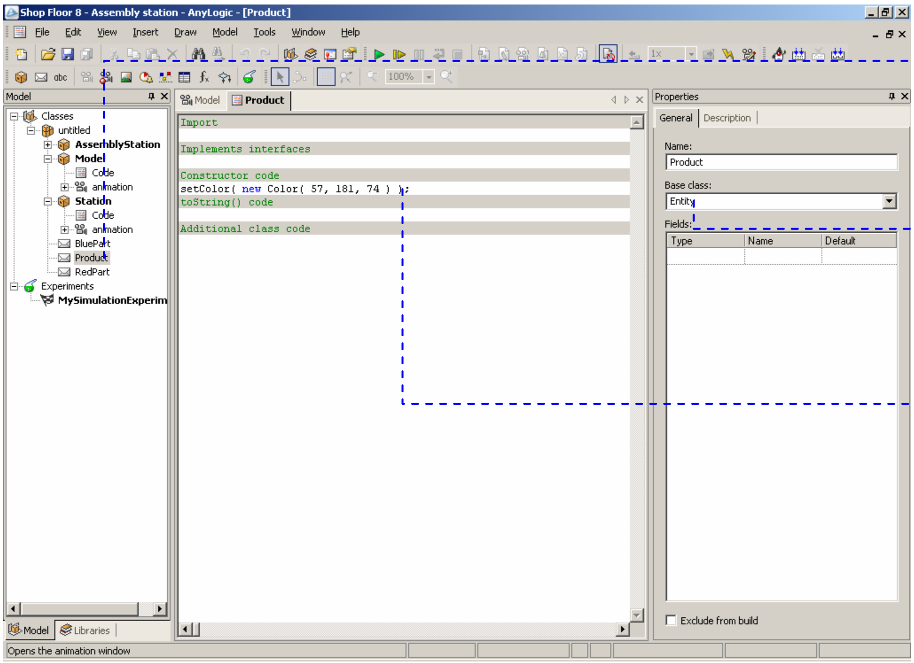
Now we can run the model and observe its behavior under different source rates. Click *Run* ▶ to start the model.

↘ The reference model for this point is Examples \ Enterprise Library Tutorial Models \ Shop Floor 6 - Changing source rates.alp.

3.8 Assembly station

Now we will create an assembly station. Before creating the station, we will create new `Product` entity type.

► Create Product entity



Name: Product

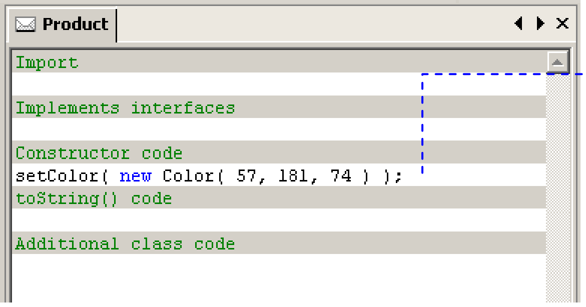
Base class: Entity

❶ Create Product message class

❷ Derive the message from Entity

❸ Change entity appearance

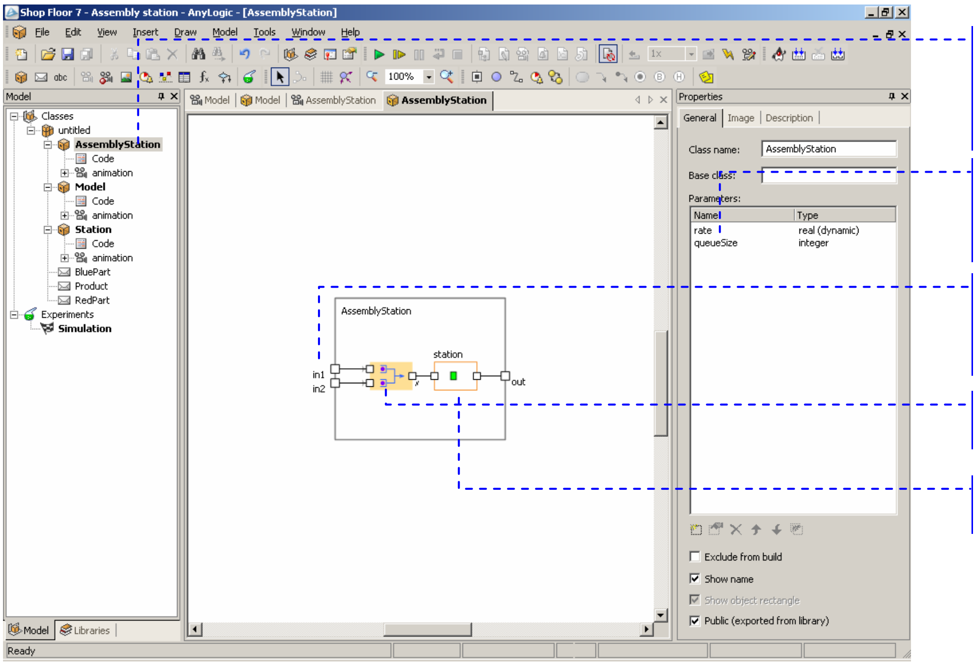
❸ Set the custom product color:



```
Import
Implements interfaces
Constructor code
setColor( new Color( 57, 181, 74 ) );
toString() code
Additional class code
```

Ⓐ Specifies the custom dark green color with red component 57, green component 181, blue component 74

► Create an assembly station



Class name: AssemblyStation

Base class:

Parameters:

Name	Type
rate	real (dynamic)
queueSize	integer

☐ Exclude from build

☑ Show name

☑ Show object rectangle

☑ Public (exported from library)

❶ Create AssemblyStation class

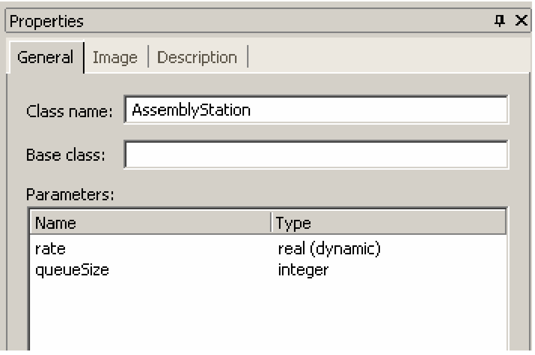
❷ Create parameters

❸ Create three ports

❹ Combine

❺ Station

❷ Define two assembly station parameters. Define a new parameter `rate` of type `real` and specify that it is *Dynamic*. Define also parameter `queueSize` of type `integer` (this will be a simple, not a dynamic parameter).



Class name: AssemblyStation

Base class:

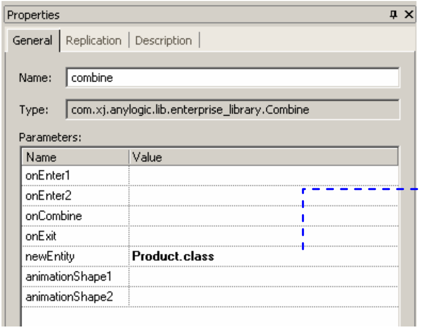
Parameters:

Name	Type
rate	real (dynamic)
queueSize	integer

Set the default value of `rate` of 20 and `queueSize` of 2.

❸ You can name the ports `in1`, `in2`, and `out`. Leave default port properties.

❹ Set up the following properties for the Combine object:



Name: combine

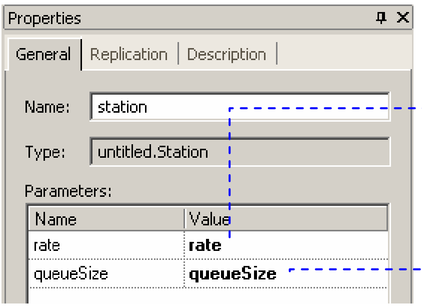
Type: com.xj.anylogic.lib.enterprise_library.Combine

Parameters:

Name	Value
onEnter1	
onEnter2	
onCombine	
onExit	
newEntity	**Product.class**
animationShape1	
animationShape2	

Ⓐ Select `Product` entity type

❻ Set up the following properties for the Station object:



Name: station

Type: untitled.Station

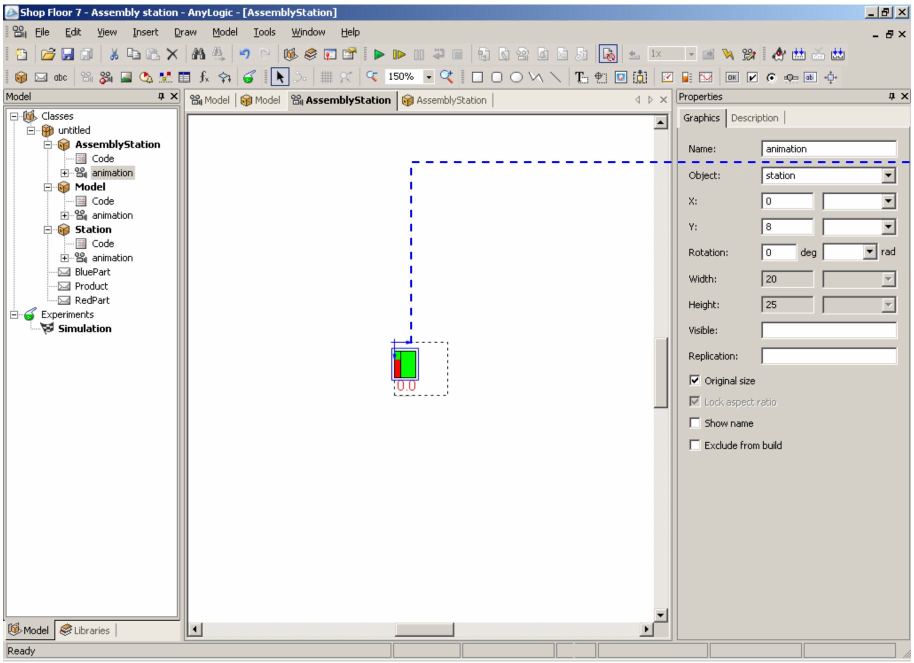
Parameters:

Name	Value
rate	**rate**
queueSize	**queueSize**

Ⓐ Station processing rate is linked to the assembly station parameter

Ⓑ Queue size is linked to the assembly station parameter

► Adjust AssemblyStation animation



Name: animation

Object: station

X: 0

Y: 8

Rotation: 0 deg rad

Width: 20

Height: 25

Visible:

Replication:

☑ Original size

☑ Lock aspect ratio

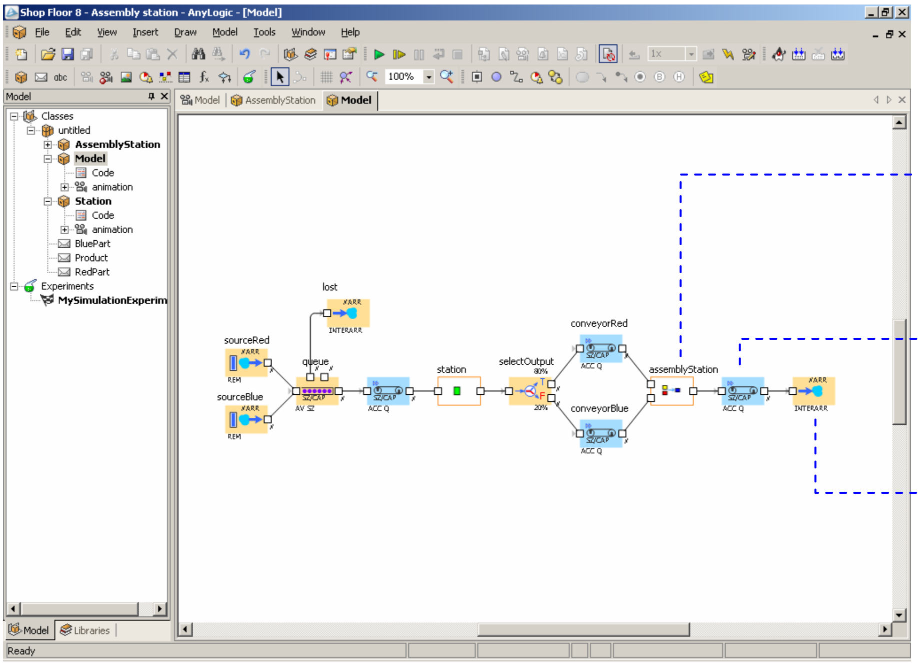
□ Show name

□ Exclude from build

❶ Move the station animation in the animation area rectangle

Now we can test the assembly station created.

▶ Add an assembly station to the flowchart

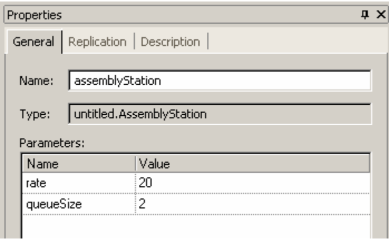


❶ AssemblyStation

❷ Conveyor

❸ One Sink

❶ Add an assembly station and leave all properties by default:



Properties

General | Replication | Description

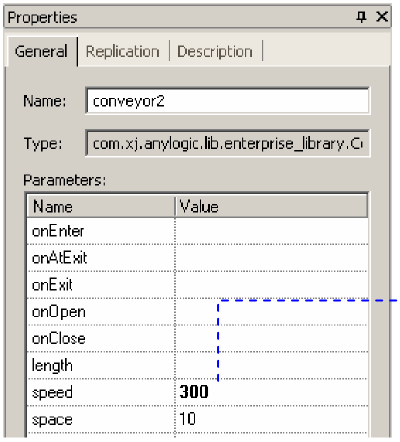
Name: assemblyStation

Type: untitled.AssemblyStation

Parameters:

Name	Value
rate	20
queueSize	2

❷ For the conveyor created, specify the following:



Properties

General | Replication | Description

Name: conveyor2

Type: com.xj.anylogic.lib.enterprise_library.C

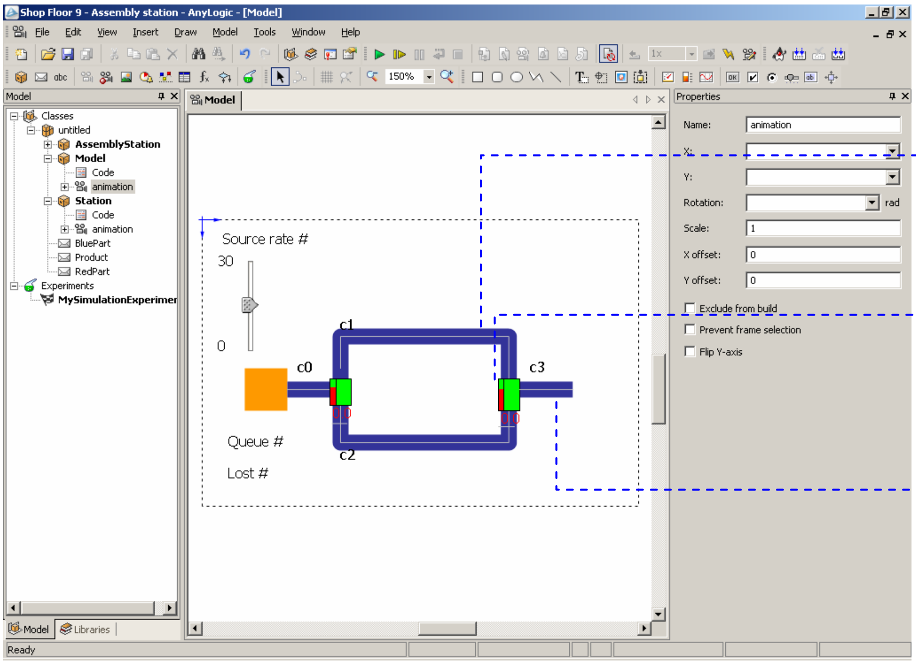
Parameters:

Name	Value
onEnter	
onAtExit	
onExit	
onOpen	
onClose	
length	
speed	**300**
space	10

Ⓐ Set the speed of 300

❸ Instead of two Sink objects, place just one. Check that it is named `sink`.

► Draw animation



Name: animation

X:

Y:

Rotation: rad

Scale: 1

X offset: 0

Y offset: 0

□ Exclude from build

□ Prevent frame selection

□ Flip Y-axis

❶ Change the shape of the conveyors

❷ Move the assembly station animation here

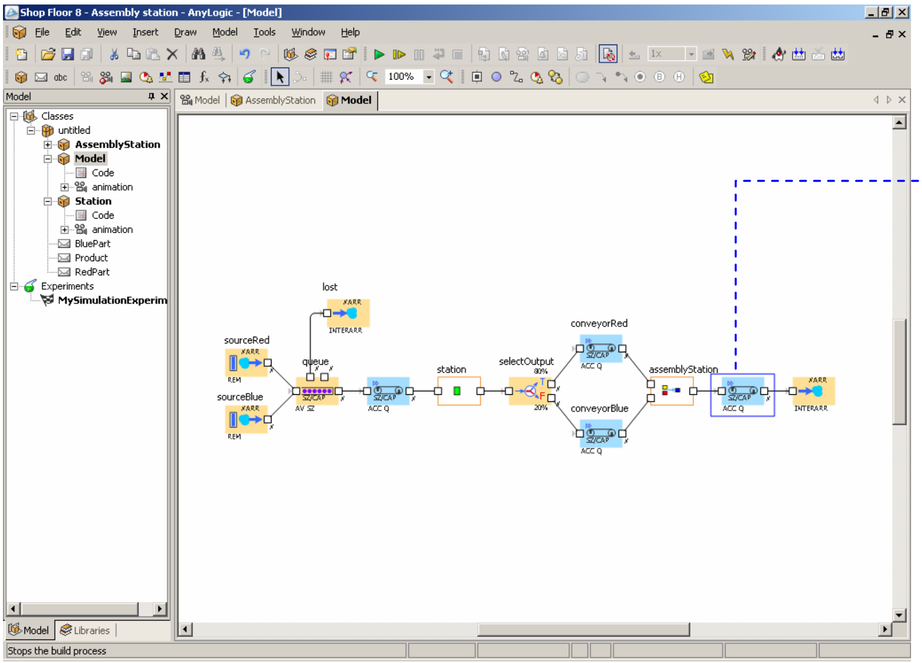
❸ Draw new conveyor segment

❶ Make `c1` and `c2` the conveyors leading to the assembly station.

❷ Place the automatically appearing animation of the assembly station here.

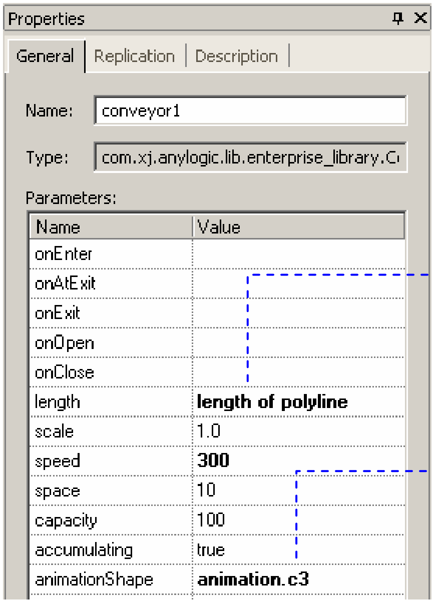
❸ Draw new `c3` conveyor segment leading from the assembly station.

► Set up animation



❶ Set up animation

❶ Specify the following animation properties for the conveyor:



Name: conveyor1

Type: com.xj.anylogic.lib.enterprise_library.C

Parameters:

Name	Value
onEnter	
onAtExit	
onExit	
onOpen	
onClose	
length	**length of polyline**
scale	1.0
speed	**300**
space	10
capacity	100
accumulating	true
animationShape	**animation.c3**

Ⓐ Select to measure the length of the polyline

Ⓑ Select c3 as the animation shape

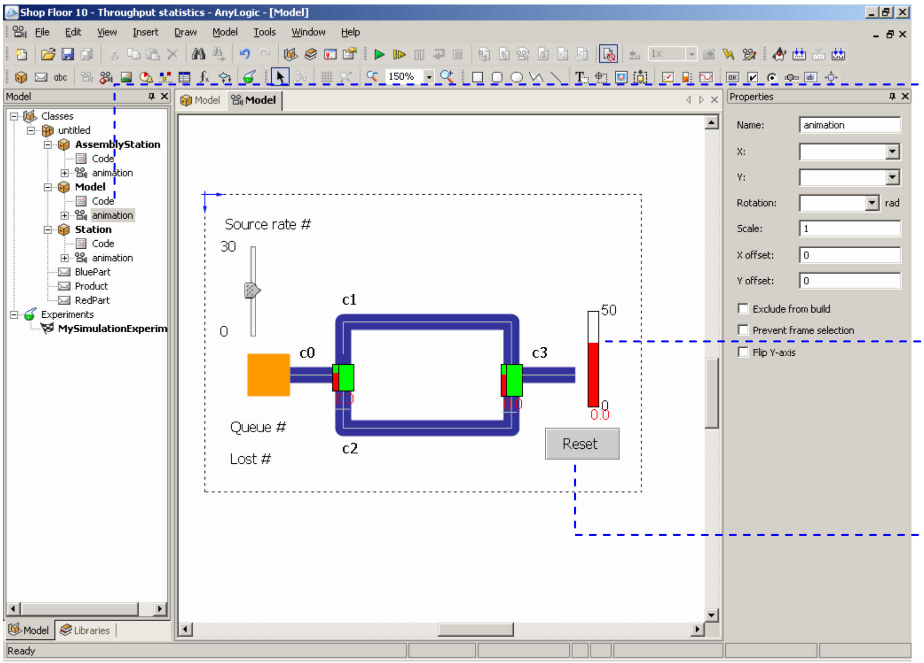
We finished creating our shop floor model. Click *Run* ▶ to start the model.

➘ The reference model for this point is Examples \ Enterprise Library Tutorial Models \ Shop Floor 7 - Assembly station.alp.

3.9 Displaying throughput statistics

Now we will show throughput statistics.

► Show throughput statistics



Properties

Name: animation

X:

Y:

Rotation: rad

Scale: 1

X offset: 0

Y offset: 0

☐ Exclude from build

☐ Prevent frame selection

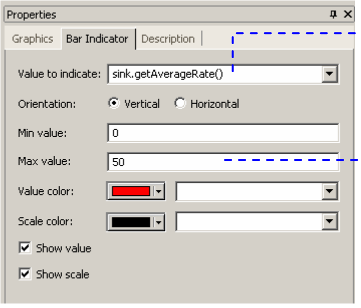
☐ Flip Y-axis

❶ Open Model animation

❷ Draw the throughput indicator

❸ Place a button to reset statistics

❷ For the bar indicator, set up the following properties:



Value to indicate: sink.getAverageRate()

Orientation: ● Vertical ○ Horizontal

Min value: 0

Max value: 50

Value color: 🟥

Scale color: ⬛

☑ Show value

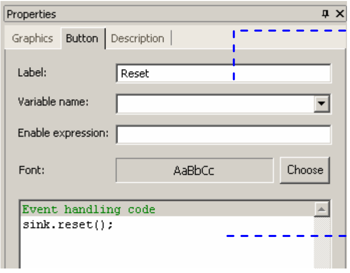
☑ Show scale

Ⓐ Show average sink rate

Ⓑ Set max value to 50

Ⓐ Type `sink.getAverageRate()` to display average exit rate.

❸ Resize the button as necessary and set the button label.



Label: Reset

Variable name:

Enable expression:

Font: AaBbCc

Event handling code

```
sink.reset();
```

🅐 Type Reset as button label

🅑 Type `sink.reset()`

Now we can run the model and see how the throughput statistics is collected. Click *Run* ▶ to start the model.

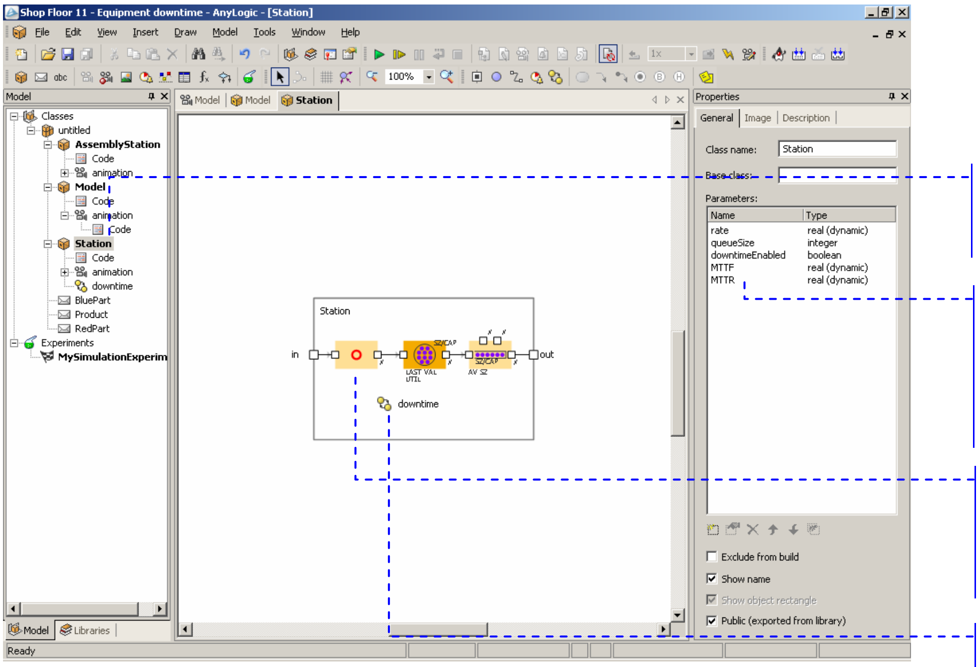
The reference model for this point is Examples \ Enterprise Library Tutorial Models \ Shop Floor 8 - Throughput statistics.alp.

3.10 Equipment Downtime

Equipment downtime and maintenance period can have a great impact on the factory performance. You can simulate and analyze the equipment downtime consequences to improve the performance.

Now we will introduce station downtime.

► Create station downtime



Class name: Station

Base class:

Parameters:

Name	Type
rate	real (dynamic)
queueSize	integer
downtimeEnabled	boolean
MTTF	real (dynamic)
MTTR	real (dynamic)

☐ Exclude from build

☑ Show name

☑ Show object rectangle

☑ Public (exported from library)

❶ Modify Station class

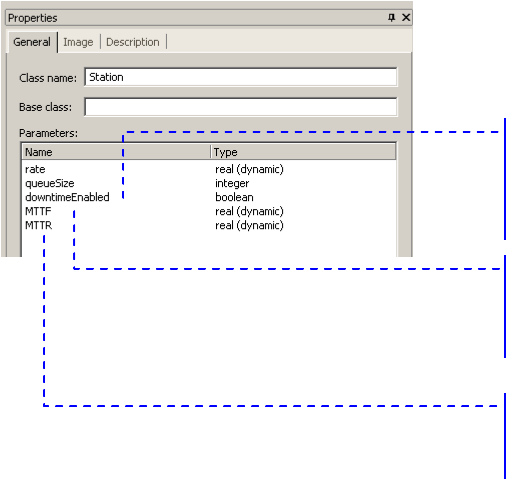
❷ Create MTTF and MTTR parameters

❸ Add a Hold object to model downtimes

❹ Describe downtime behavior with a statechart

❶ Double-click Station in the tree to open the structure diagram.

❷ We create station parameters for mean time to failure (MTTF) and mean time to repair (MTTR).



Class name: Station

Base class:

Parameters:

Name	Type
rate	real (dynamic)
queueSize	integer
downtimeEnabled	boolean
MTTF	real (dynamic)
MTTR	real (dynamic)

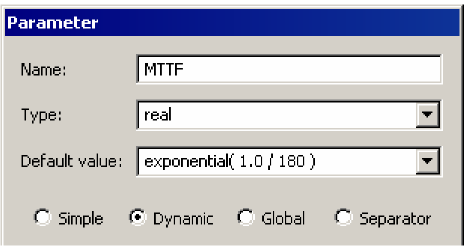
Ⓐ Enables or disables downtimes for the station

Ⓑ Mean time to failure parameter

Ⓒ Mean time to repair parameter

Ⓐ Create `downtimeEnabled` parameter of type `boolean`. Specify `true` as the default value. This parameter will enable or disable downtime for a station.

Ⓑ Create `MTTF` parameter of type `real` and specify that it is *Dynamic*. As a default value, we set up time to failure to be exponentially distributed with the mean of 3 hours (180 minutes). Note that we should specify rate for `exponential()` function, and to obtain rate we use the inverse value (note that we use 1.0 instead of 1 because 1/180 will return 0 as both numbers are integer).



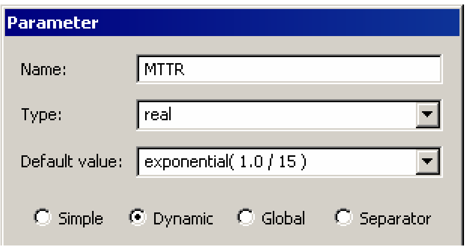
Name: MTTF

Type: real

Default value: exponential(1.0 / 180)

○ Simple ● Dynamic ○ Global ○ Separator

C Create `MTTR` parameter of type `real` and specify that it is *Dynamic*. As a default value, we set up time to repair to be exponentially distributed with the mean of a quarter of an hour.



Parameter

Name: MTTR

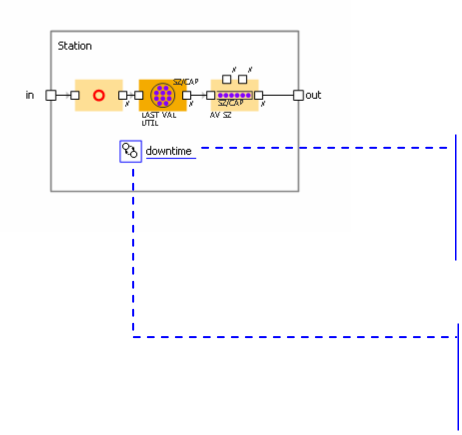
Type: real

Default value: exponential(1.0 / 15)

○ Simple ● Dynamic ○ Global ○ Separator

❸ The Hold object will block the input during downtime. Object is named `hold` by default; keep this name as we will use it later on.

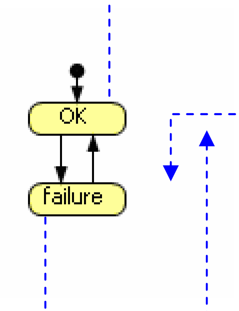
❹ We define downtime behavior visually with a statechart:



A Click *Statechart* and then create a new statechart

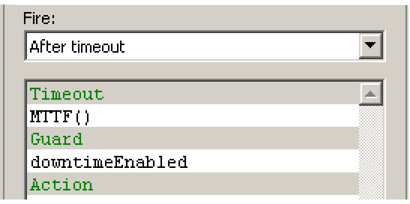
B Double-click to open the statechart diagram

Draw the following statechart:



Ⓐ Press F2 and rename the default state to OK

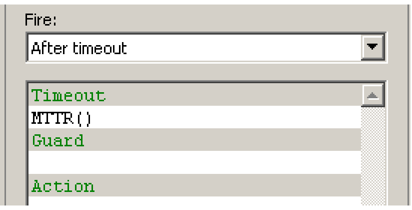
Ⓑ Set up that `OK→failure` transition will be taken after MTTF time, if downtime is enabled for the station



Fire: After timeout

```
Timeout
MTTF()
Guard
downtimeEnabled
Action
```

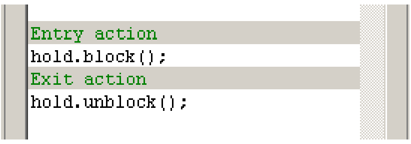
Ⓒ Set up that `failure→OK` transition should be taken after repair time



Fire: After timeout

```
Timeout
MTTR()
Guard

Action
```

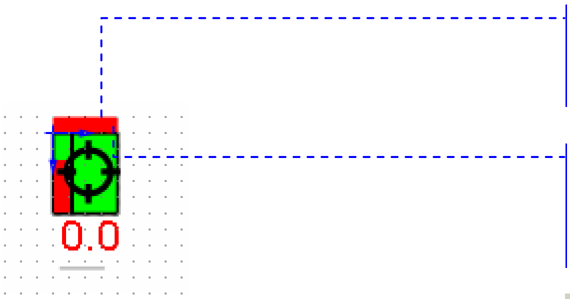
Ⓓ Once the statechart enters `failure` state, the station input will be blocked. Once the statechart exits `failure` state, the station input will be unblocked



```
Entry action
hold.block();
Exit action
hold.unblock();
```

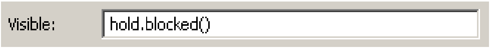
Now the station will go down in a random time. To indicate downtime, in the animation diagram we will draw a downtime indicator next to the station. The alternative approach that can be easily implemented as well, is to change station run-time color to be red during downtime.

▶ Show downtime in animation



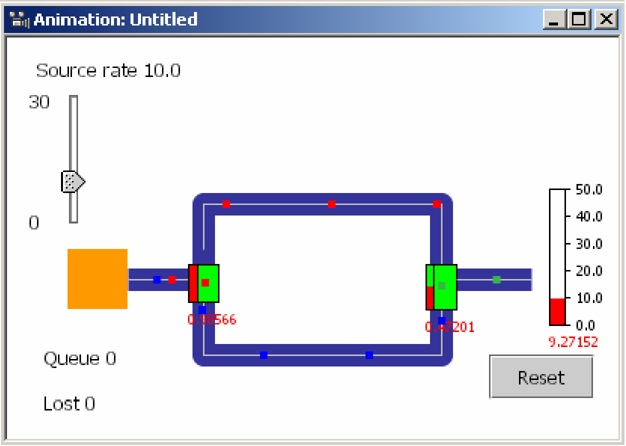
❶ Add a downtime indicator as a red rectangle

❷ Set up to show the indicator during downtime only



Visible: hold.blocked()

Now you can click *Run* ▶ to start the model execution. You will see the picture like the one shown below:



The reference model for this point is Examples \ Enterprise Library Tutorial Models \ Shop Floor 9 - Equipment downtime.alp.

4. Ophthalmology Department Model

In this section of the tutorial we will build a model of a typical ophthalmology unit. Patients arrive to the department to undergo the ophthalmoscopy procedure. They are held in the waiting room until some procedure room becomes available. Then patient moves to this procedure room escorted by a nurse and waits for the ophthalmologist to come and make an examination. The procedure is performed using an ophthalmoscope. Ophthalmoscopes are stored in the storage room and brought by doctors to the procedure room just before the procedure begins. Following the procedure, the doctor transports the ophthalmoscope back to the storage room and returns to the staff lounge, and the patient leaves the ophthalmology department.

We will construct our model using Enterprise Library 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.

4.1 Creating a new project

Create new model as described in Section 1.1, "How to create a new AnyLogic™ model". Rename `Main` class to `Model`. Using *Simulation* in the *Experiments* tree, specify that the model is executed in real time mode and one model time unit will be executed in one second.

4.2 Creating a model animation

Now we will draw the model animation since the logical structure of the model is generated from the animation you draw. To simplify drawing, we will add the model layout first.

You can draw the department layout in AnyLogic™ using the animation editor, or you can import any picture as the layout. The drawing approach is good for prototyping, when you want to experiment with draft layouts. The image approach is excellent for working with an existing layout.

Now we will add the department layout.

► Add department layout

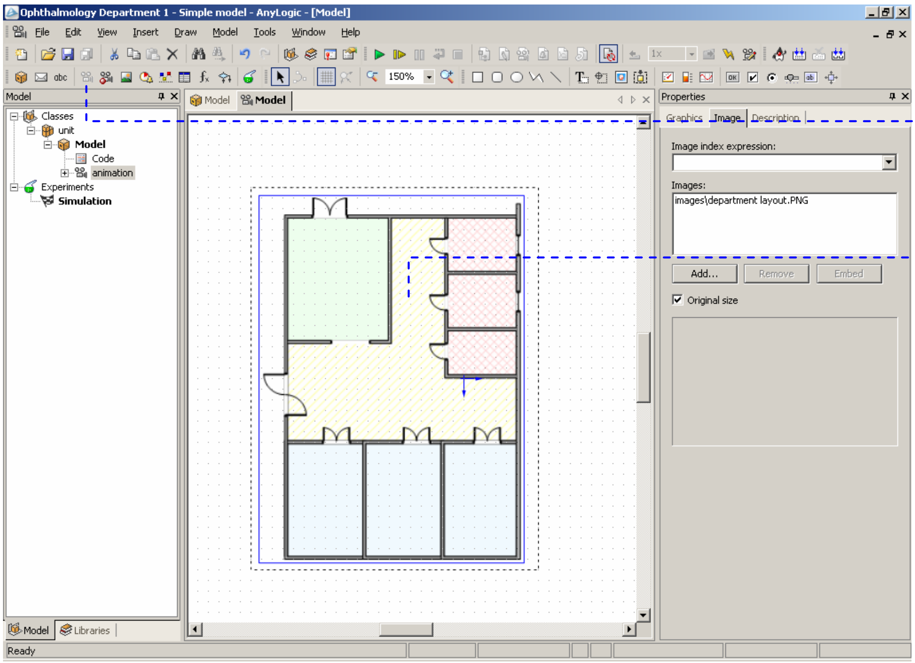

Ophthalmology Department 1 - Simple model - AnyLogic - [Model]
File Edit View Insert Draw Model Tools Window Help
150%
Model
Classes
unit
Model
Code
animation
Experiments
Simulation
Model
Libraries
Ready
Properties
Graphics Image Description
Add... Remove Embed

Image index expression:

Images: images\department layout.PNG

☑ Original size

❶ Create new animation

❷ Add layout

❶ To create a new animation, click the *New Animation* toolbar button.

❷ Add the department layout. Click the *Image* toolbar button, and click the animation diagram to place a new image. Set up the following properties:

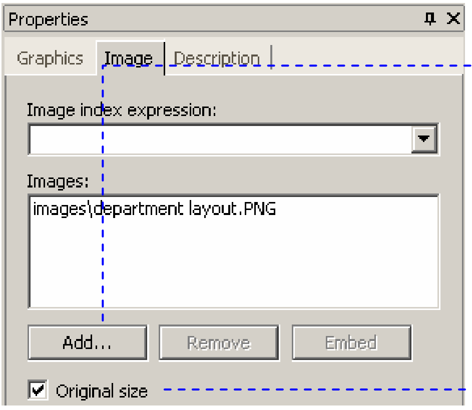


Image index expression:

Images: images\department layout.PNG

☑ Original size

Ⓐ Click *Add…* button and navigate to layout picture

Ⓑ Select *Original size* option

Ⓐ Navigate to the layout picture of our system.

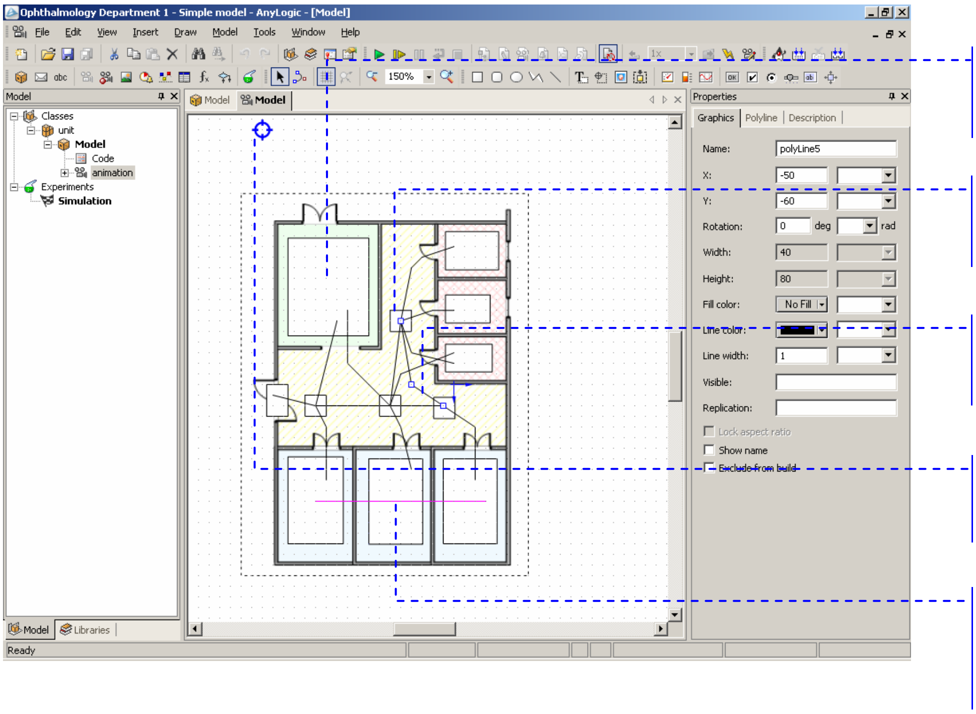
➘ The layout picture we will use is Examples \ Enterprise Library Tutorial Models \ images \ department layout.png

🅑 To preserve necessary picture size, select the *Original size* option.

Now we will draw the model animation. Drawing an animation, we define the network of the advanced transportation model, where rectangles represent network nodes and polylines— network links.

We will draw a rectangle for each department room and connect rectangles with polylines playing the role of moving paths in our model.

▶ Draw model animation



Properties

Graphics | Polyline | Description

Name: polyLine5

X: -50

Y: -60

Rotation: 0 deg rad

Width: 40

Height: 80

Fill color: No Fill

Line color:

Line width: 1

Visible:

Replication:

□ Lock aspect ratio

□ Show name

□ Exclude from build

❶ Draw room locations (rectangles)

❷ Draw additional nodes (rectangles)

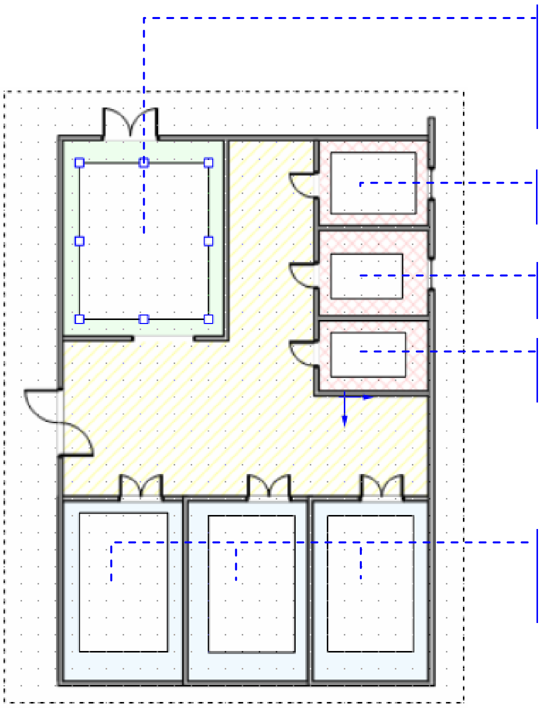
❸ Draw moving paths (polylines)

❹ Create network pivot group

❺ Draw procedure rooms location path (polyline)

❶ Ophthalmology department has the following functional areas: waiting room, three procedure rooms, storage room for ophthalmoscopes and private office spaces for medical staff including staffroom and staff lounge.

Draw rectangles representing department rooms using the *Rectangle* □ drawing tool. Resize rectangles to fit into the corresponding areas of the layout as shown in the figure below:



❶ Draw a rectangle representing the waiting room

❷ Draw the staffroom

❸ Draw the staff lounge

❹ Draw the storage room

❺ Draw the procedure rooms

❶ Draw a rectangle representing the waiting room. Name it `waitingRoom`.

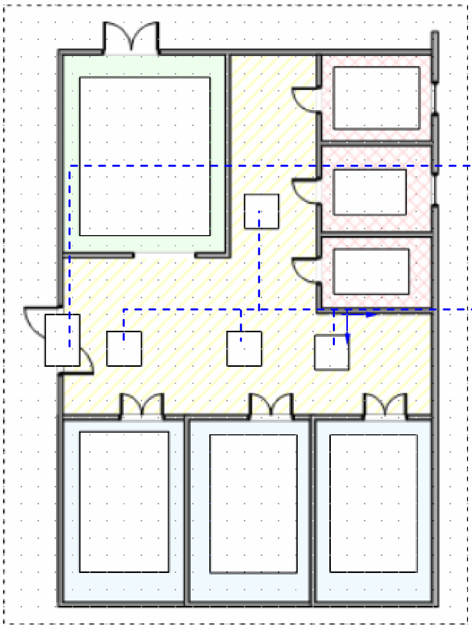
❷ Name the rectangle `staffroom`.

❸ Name the rectangle `staffLounge`.

❹ Name the rectangle `storageRoom`.

❺ Draw three rectangles named `procRoom1`, `procRoom2` and `procRoom3` representing the procedure rooms.

❷ To make moving in our model more sophisticated, we will also add some additional rectangles to represent intermediate moving destinations.



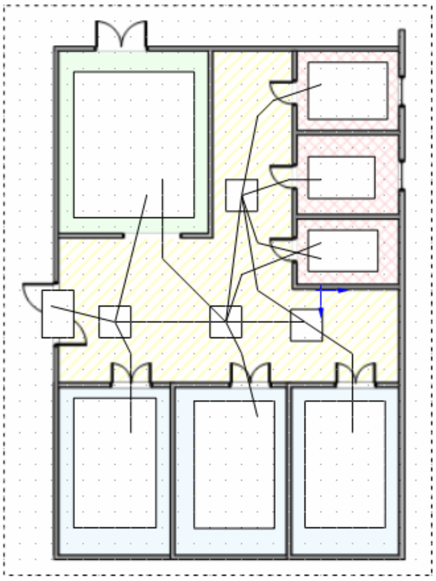
Ⓐ Draw department exit

Ⓑ Draw intermediate nodes

Ⓐ Draw a rectangle representing the department exit. Name it `exit`.

Ⓑ Draw four additional rectangles standing for intermediate network nodes and place them as shown in the figure. Name rectangles `upNode`, `leftNode`, `middleNode` and `rightNode` according to their location on animation.

❸ Using the *Polyline* drawing tool, draw polylines as shown in the following figure.



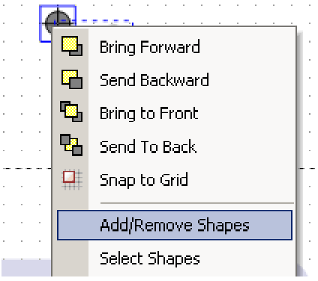
Polylines represent moving paths in the model. To define a correct model network, connect adjacent nodes, namely connect:

- `leftNode` with `exit`, `waitingRoom`, `procRoom1` and `middleNode`;
- `middleNode` – with `waitingRoom`, `procRoom2`, `storageRoom`, `rightNode` and `upNode`;
- `rightNode` – with `procRoom3` and `upNode`;
- `upNode` – with `storageRoom`, `staffLounge` and `staffroom`.

➲ Note that all end points of polylines should necessarily be inside the connected rectangles.

❹ Create pivot group and name it `networkPivot`. The logical structure of the network will be constructed from the animation elements added to the pivot group only.

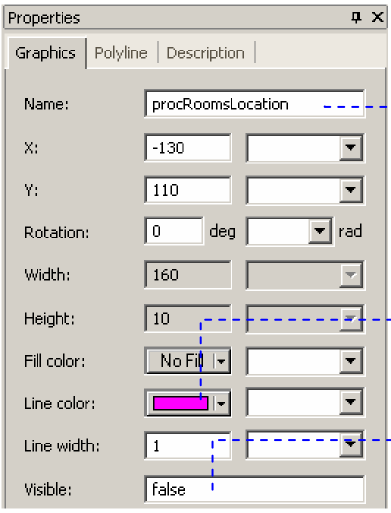
Add all created shapes to the pivot group. Right-click the pivot shape and choose *Add/Remove shapes* from the popup menu.



Then select all shapes, and click on the diagram to add them to the pivot.

❺ Draw a polyline connecting rectangles representing the procedure rooms. We will use it later on.

Place polyline points inside the `procRoom1`, `procRoom2` and `procRoom3` rectangles. Set the following properties:



Name: procRoomsLocation
X: -130
Y: 110
Rotation: 0 deg rad
Width: 160
Height: 10
Fill color: No Fill
Line color:
Line width: 1
Visible: false

A Name it `procRoomsLocation`

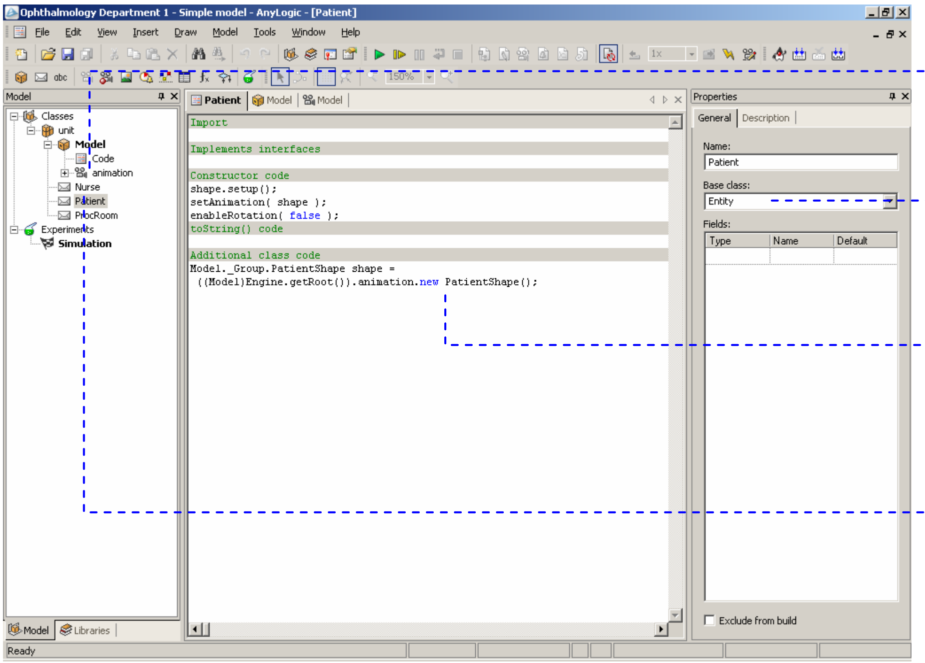
B Set custom line color

C Make polyline invisible

4.3 Creating entity message classes

We need to define new message classes for each resource available in our model. Also, we will create a message class to represent patients. We will assign animation to the message classes created to customize entities animation.

► Create message classes



Name: Patient

Base class: Entity

```
Import

Implements interfaces

Constructor code
shape.setup();
setAnimation( shape );
enableRotation( false );
toString() code

Additional class code
Model._Group.PatientShape shape =
 ((Model)Engine.getRoot()).animation.new PatientShape();
```

❶ Create `Patient` message class

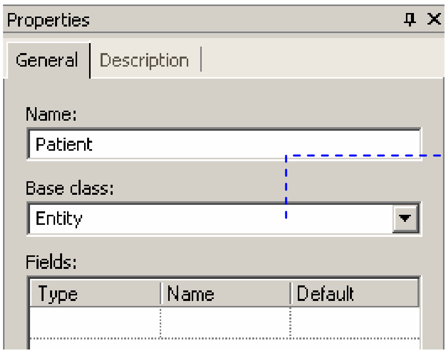
❷ Derive the message from `Entity`

❸ Write the animation code

❹ The same applies to the `Nurse` and `ProcRoom` message classes

❶ Create `Patient` message class by clicking the *New Message Class* ✉ toolbar button. Name the message class `Patient`.

❷



Name: Patient

Base class: Entity

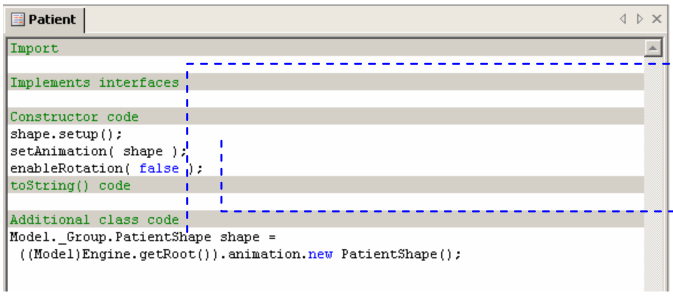
Fields:

Type	Name	Default

🅐 Select `Entity` as the base class for the message created

🅐 You need to derive messages from the `Entity` message class in order to use messages with the Enterprise Library.

❸



```
Import

Implements interfaces

Constructor code
shape.setup();
setAnimation( shape );
enableRotation( false );
toString() code

Additional class code
Model._Group.PatientShape shape =
 ((Model)Engine.getRoot()).animation.new PatientShape();
```

🅐 Create resource animation

🅑 Set up resource animation

🅐 To create a `PatientShape` symbol instance, write the following code in the *Additional class code* section:

```
Model._Group.PatientShape shape =
((Model)Engine.getRoot()).animation.new
PatientShape();
```

This code creates a new dynamic pivot instance.

🅑 To set up the animation, write the following code in the *Constructor code* section:

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

❹ Create `Nurse` message class to represent nurse resource units. Do it in the same way except the code you type in the *Additional class code* section should be the following:

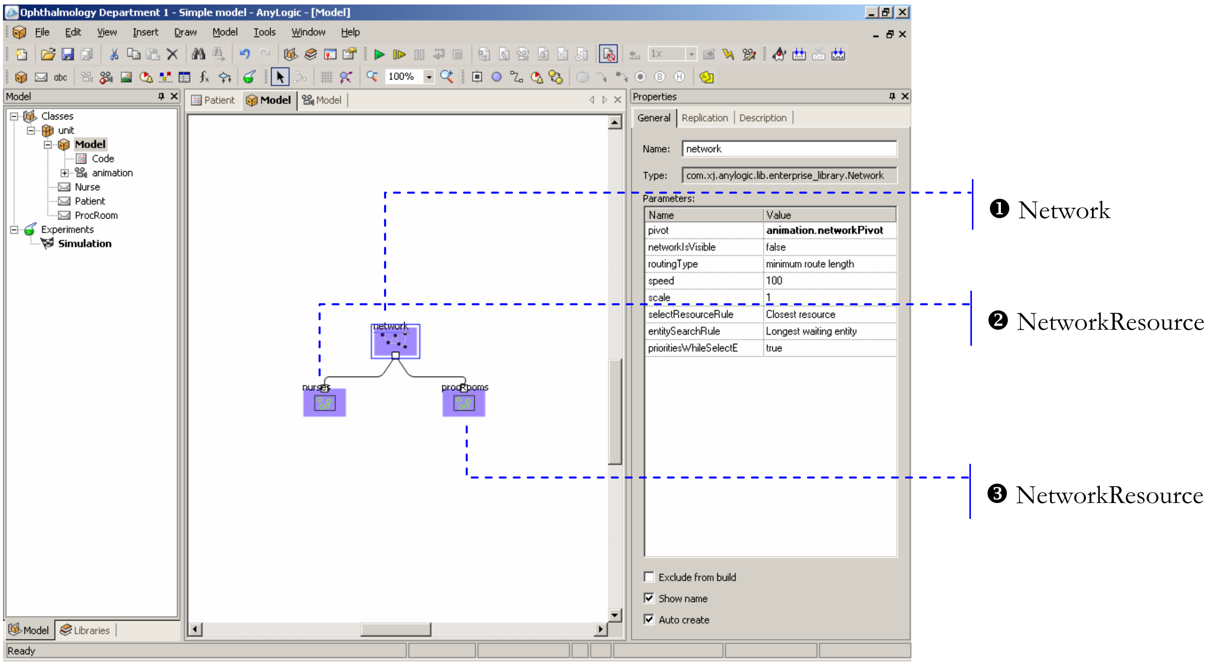
```
Model._Group.NurseShape shape =
((Model)Engine.getRoot()).animation.new NurseShape();
```

Create `ProcRoom` message class to represent procedure rooms resource units. Set Entity as the base class. Do not write any animation code since these resources are static and we do not need to animate them.

4.4 Defining the model network

Now we need to add objects describing the network and resources available in it. The resources may be of three types: staff, portable and static. In our example, nurses and doctors are staff resources, scopes are portable resources, and procedure rooms are the static ones. Now we will define nurse and procedure room resources. We will define doctor and scope resources later on.

► Describe the model network



Name: network

Type: com.xj.anylogic.lib.enterprise_library.Network

Parameters:

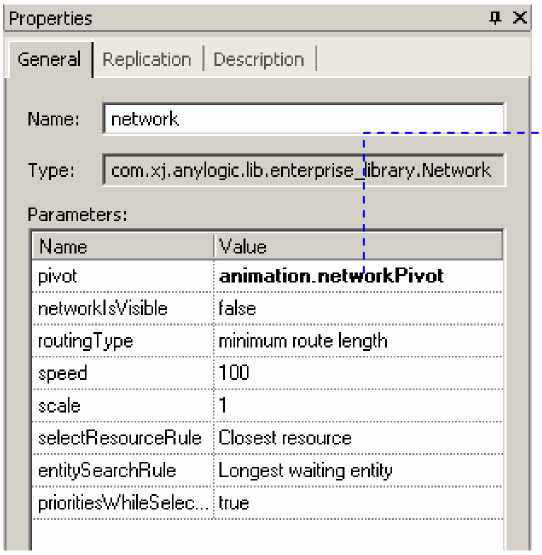
Name	Value
pivot	**animation.networkPivot**
networkIsVisible	false
routingType	minimum route length
speed	100
scale	1
selectResourceRule	Closest resource
entitySearchRule	Longest waiting entity
prioritiesWhileSelectE	true

□ Exclude from build

☑ Show name

☑ Auto create

❶ Create a Network object. Network object defines the network. Set the following properties:



Properties

General | Replication | Description

Name: network

Type: com.xj.anylogic.lib.enterprise_library.Network

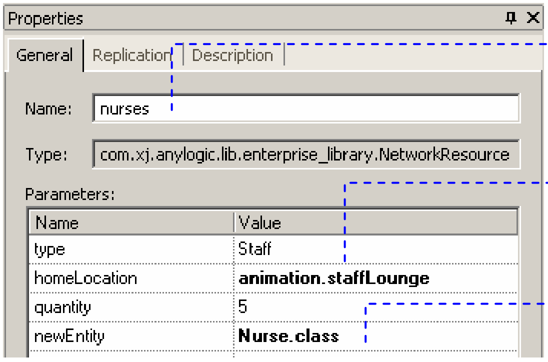
Parameters:

Name	Value
pivot	**animation.networkPivot**
networkIsVisible	false
routingType	minimum route length
speed	100
scale	1
selectResourceRule	Closest resource
entitySearchRule	Longest waiting entity
prioritiesWhileSelec...	true

Ⓐ Choose `animation.networkPivot` as the network pivot group

Ⓐ Choose the pivot group containing shapes standing for network nodes and links. Thus we associate the defined network structure with this Network object.

❷ Create a NetworkResource object and connect it to the Network object. NetworkResource describes resources of a particular type. This one will describe nurse resources. Set the following object properties:



Properties

General | Replication | Description

Name: nurses

Type: com.xj.anylogic.lib.enterprise_library.NetworkResource

Parameters:

Name	Value
type	Staff
homeLocation	**animation.staffLounge**
quantity	5
newEntity	**Nurse.class**

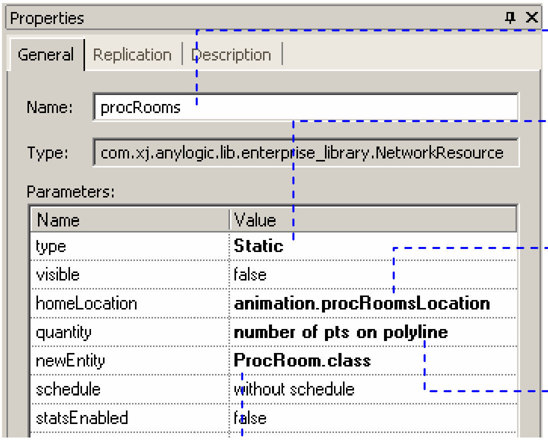
Ⓐ Name the object `nurses`

Ⓑ Set `animation.staffLounge` as the resources home location

Ⓒ Specify that resource units are messages of `Nurse` class

Ⓑ Specify the home location of the resource. A staff resource unit returns there if it gets idle.

❸ Create a NetworkResource object describing procedure room resources. Set the following object properties:



Name: procRooms

Type: com.xj.anylogic.lib.enterprise_library.NetworkResource

Parameters:

Name	Value
type	**Static**
visible	false
homeLocation	**animation.procRoomsLocation**
quantity	**number of pts on polyline**
newEntity	**ProcRoom.class**
schedule	without schedule
statsEnabled	false

Ⓐ Name the object `procRooms`

Ⓑ Set Static resource type

Ⓒ Set the resources location

Ⓓ Specify the number of resource units

Ⓔ Specify that resource units are messages of the class `ProcRoom`

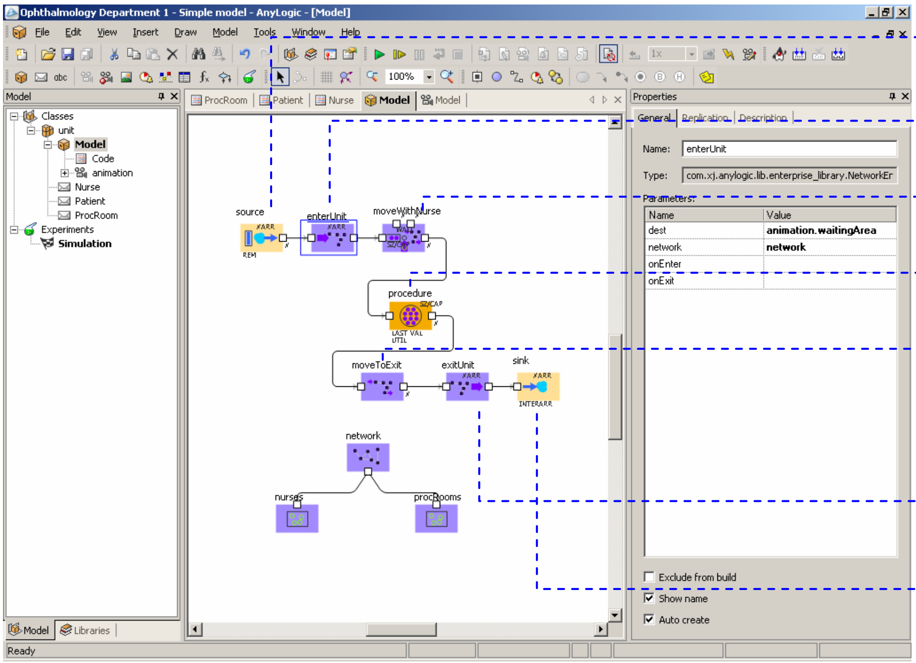
Ⓒ A static resource always resides at its home location. There can be a number of resource home locations. In this case you need to draw a polyline with points lying in the corresponding rectangles and specify it as the object's `homeLocation`. Choose `animation.procRoomsLocation` polyline created beforehand for this purpose.

Ⓓ Specify that the resource object has the number of resource units equal to the number of points of the specified polyline.

4.5 Creating a flowchart

Now we will create a flowchart of our model.

► Create a flowchart



Properties

General | Replication | Description

Name: enterUnit

Type: com.xj.anylogic.lib.enterprise_library.NetworkEr

Parameters:

Name	Value
dest	**animation.waitingArea**
network	**network**
onEnter	
onExit	

☐ Exclude from build

☑ Show name

☑ Auto create

❶ Source

❷ NetworkEnter

❸ NetworkMoveToWithQ

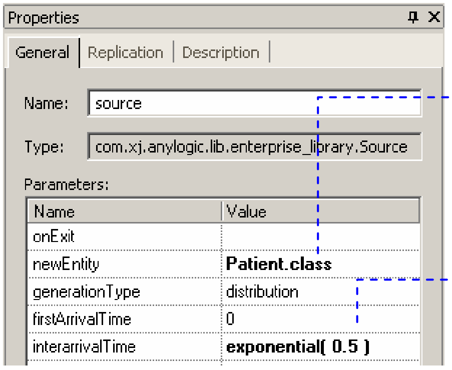
❹ Delay

❺ NetworkMoveTo

❻ NetworkExit

❼ Sink

❶ Add a Source object. Set the following properties:



Properties

General | Replication | Description

Name: source

Type: com.xj.anylogic.lib.enterprise_library.Source

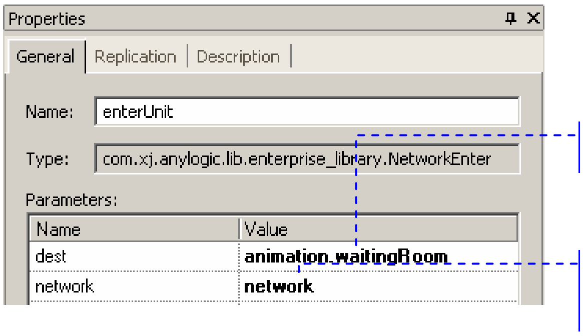
Parameters:

Name	Value
onExit	
newEntity	**Patient.class**
generationType	distribution
firstArrivalTime	0
interarrivalTime	**exponential(0.5)**

Ⓐ Specify Patient as the entity to be generated

Ⓑ Set exponentially distributed interarrival time with 2 mean value

❷ NetworkEnter object adds entities to the specified location in a network. Set the following properties:



Name: enterUnit

Type: com.xj.anylogic.lib.enterprise_library.NetworkEnter

Parameters:

Name	Value
dest	**animation.waitingRoom**
network	**network**

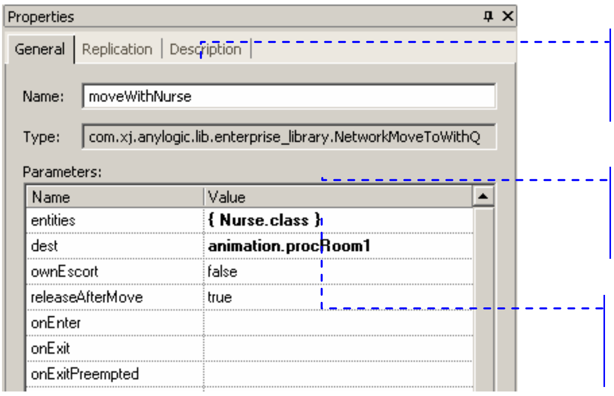
Ⓐ Specify the location

Ⓑ Specify that entities are added to `network`

Ⓐ Choose `animation.waitingRoom` as entities destination location. Patients will arrive at the department waiting room.

Ⓑ Specify the Network object describing the network, to which new entities will be added.

❸ NetworkMoveToWithQ object moves the entity to the specified location escorted by the staff resource unit. We add this object since we need patient to move to a procedure room escorted by a nurse.



Name: moveWithNurse

Type: com.xj.anylogic.lib.enterprise_library.NetworkMoveToWithQ

Parameters:

Name	Value
entities	**{ Nurse.class }**
dest	**animation.procRoom1**
ownEscort	false
releaseAfterMove	true
onEnter	
onExit	
onExitPreempted	

Ⓐ Name the object `moveWithNurse`

Ⓑ Choose the type of the escort resource

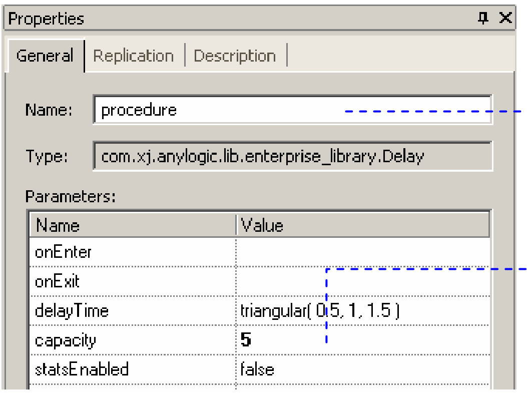
Ⓒ Specify the destination

Ⓑ Choose `Nurse` class as the escort resource type.

Ⓒ Choose `animation.procRoom1`. At this point, we will specify that all patients are examined at the first procedure room only.

❹ Delay object delays entity for the specified time that stands for the ophthalmoscopy.

Set up the following object properties:



Properties

General | Replication | Description

Name: procedure

Type: com.xj.anylogic.lib.enterprise_library.Delay

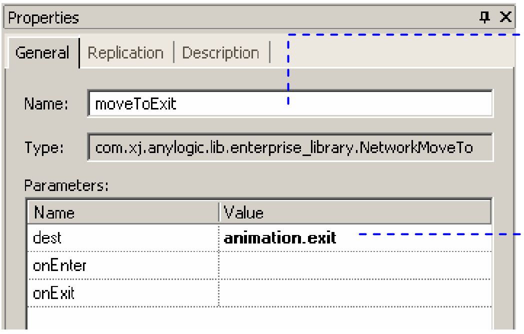
Parameters:

Name	Value
onEnter	
onExit	
delayTime	triangular(0.5, 1, 1.5)
capacity	**5**
statsEnabled	false

🅐 Name the object `procedure`

🅑 Set up processing room capacity of 5 places

❺ NetworkMoveTo object moves entity from the current location to a new one. We add this object to model how patients are leaving the department. Set up the following properties:



Properties

General | Replication | Description

Name: moveToExit

Type: com.xj.anylogic.lib.enterprise_library.NetworkMoveTo

Parameters:

Name	Value
dest	**animation.exit**
onEnter	
onExit	

🅐 Name the object `moveToExit`

🅑 Specify the destination

🅑 Patients will move to the destination you specify. Choose `animation.exit` corresponding to the unit exit.

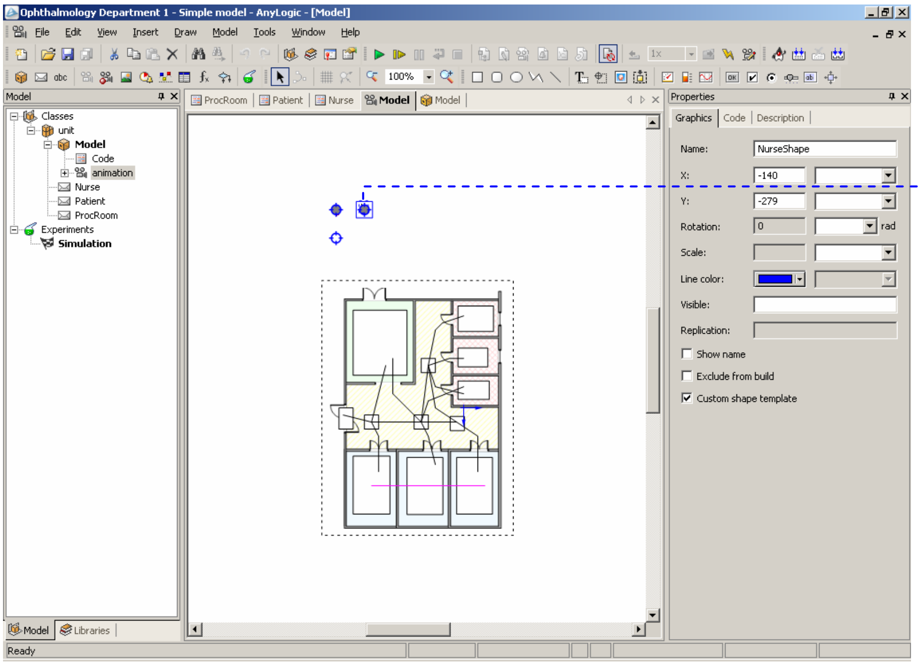
❻ The NetworkExit object removes entities from the network. Leave all properties of the NetworkExit object by default.

❼ Add a Sink object to finish the flowchart. Leave all properties of the Sink object by default.

4.6 Animating nurses and patients

Now we will create the resource symbols to customize the resources appearance.

► Draw resource symbols



Name: NurseShape

X: -140

Y: -279

Rotation: 0 rad

Scale:

Line color:

Visible:

Replication:

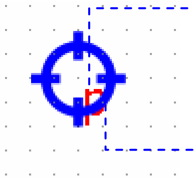
☐ Show name

☐ Exclude from build

☑ Custom shape template

❶ Draw resource symbols

❶ Draw the animation for patient.

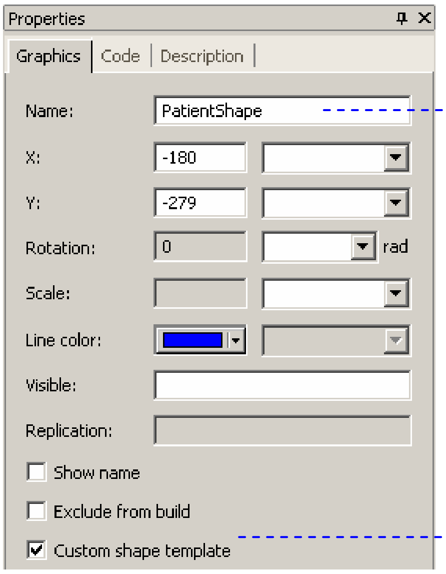


Ⓐ Create p text label to indicate a patient

Ⓑ Create a pivot and add the text label to the pivot group

Ⓐ Create a text label by clicking the *Text* T toolbar button and then clicking the animation diagram. Set red text font of size 7.

🅑 Create a dynamic pivot.



Properties

Graphics | Code | Description

Name: PatientShape

X: -180

Y: -279

Rotation: 0 rad

Scale:

Line color:

Visible:

Replication:

☐ Show name

☐ Exclude from build

☑ Custom shape template

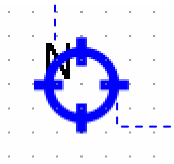
🅒 Name the pivot `PatientShape`

🅓 Enable dynamic creation

🅒 Name all pivots as shown in the pictures, as we will use those names to animate the system.

Add the text label to the pivot group. Right-click the pivot, choose *Add/Remove shapes* from the popup menu, then click the label to add it to the pivot group. Then place the label right on the pivot shape.

Do the same for the nurse symbol.



🅔 Create N text label to indicate nurse resource unit

🅕 Create `NurseShape` dynamic pivot, add the label to the pivot group and place it over the pivot

We have finished creating a simple model using AnyLogic™ advanced transportation. Now you can run the model by clicking the *Run* ▶ button. You can see that the model behaves correct: patients arrive at the waiting room, then go to the first procedure room escorted by a nurse and after some time leave the ophthalmology department.

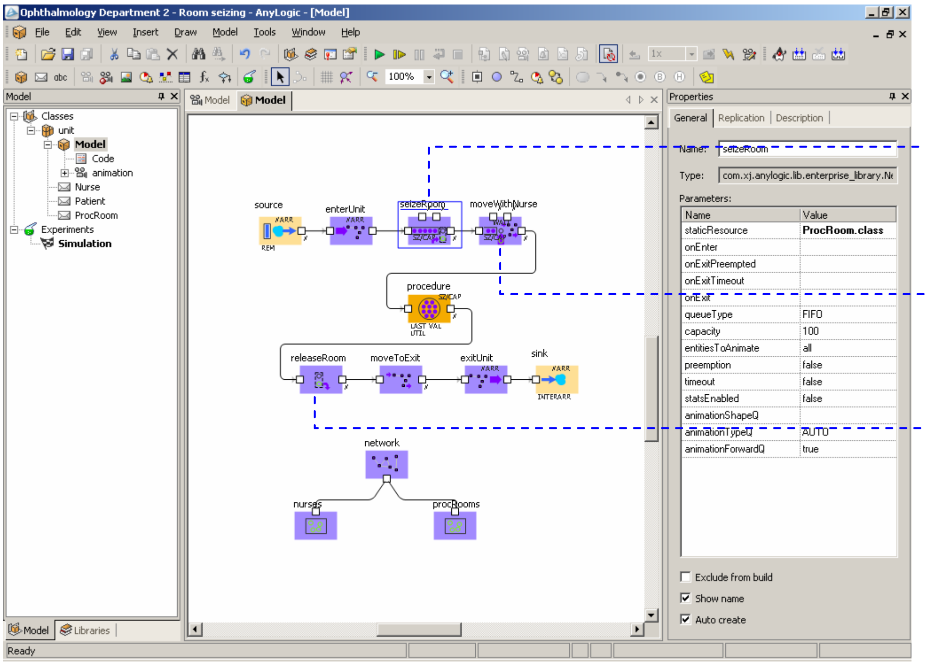
To adjust the execution speed, use *Decrease model speed* and *Increase model speed* toolbar buttons.

➘ The reference model for this point is Examples \ Enterprise Library Tutorial Models \ Ophthalmology Department 1 - Simple model.alp.

4.7 Adding room seizing logic

At this point, all patients are examined in the first procedure room only. Moreover, several patients are examined in the same procedure room at the same time. Now we will improve our model by adding room seizing logic. We will model procedure rooms as static resources. Patients will move now to any of three procedure rooms. The room will be occupied for the examination time so that no one else can be examined there at this time.

▶ Modify the flowchart



Name: seizeRoom

Type: com.xj.anylogic.lib.enterprise_library.Ne

Parameters:

Name	Value
staticResource	**ProcRoom.class**
onEnter	
onExitPreempted	
onExitTimeout	
onExit	
queueType	FIFO
capacity	100
entitiesToAnimate	all
preemption	false
timeout	false
statsEnabled	false
animationShapeQ	
animationTypeQ	AUTO
animationForwardQ	true

□ Exclude from build

☑ Show name

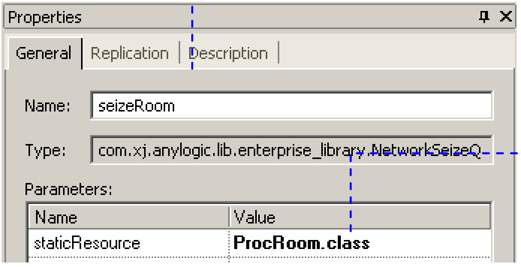
☑ Auto create

❶ NetworkSeizeQ

❷ Modify `movewithNurse` object properties

❸ NetworkRelease

❶ Add a NetworkSeizeQ object. NetworkSeizeQ object seizes static resources. We add this object to model procedure room seizing by a patient. Set the following properties:



Name: seizeRoom

Type: com.xj.anylogic.lib.enterprise_library.NetworkSeizeQ

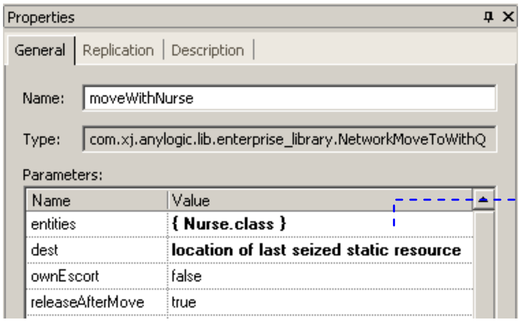
Parameters:

Name	Value
staticResource	**ProcRoom.class**

Ⓐ Name the object `seizeRoom`

Ⓑ Specify that resource units of `ProcRoom` class are to be seized

❷ Set up the following object properties:



Name: moveWithNurse

Type: com.xj.anylogic.lib.enterprise_library.NetworkMoveToWithQ

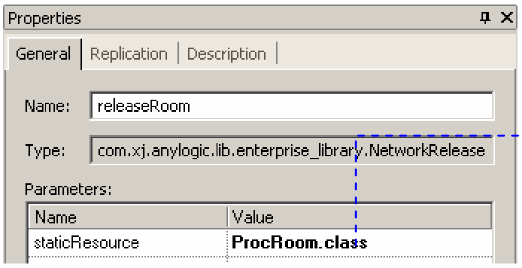
Parameters:

Name	Value
entities	**{ Nurse.class }**
dest	**location of last seized static resource**
ownEscort	false
releaseAfterMove	true

Ⓐ Specify the destination

Ⓐ Choose `location of last seized static resource`. Now patients will move for the examination to the recently seized procedure room.

❸ NetworkRelease object releases previously seized static resources. We need this object to release the procedure room after the procedure. Set the following properties:



Name: releaseRoom

Type: com.xj.anylogic.lib.enterprise_library.NetworkRelease

Parameters:

Name	Value
staticResource	**ProcRoom.class**

Ⓐ Specify `ProcRoom` as the resource to be released

Run the model by clicking the *Run* ▶ button.

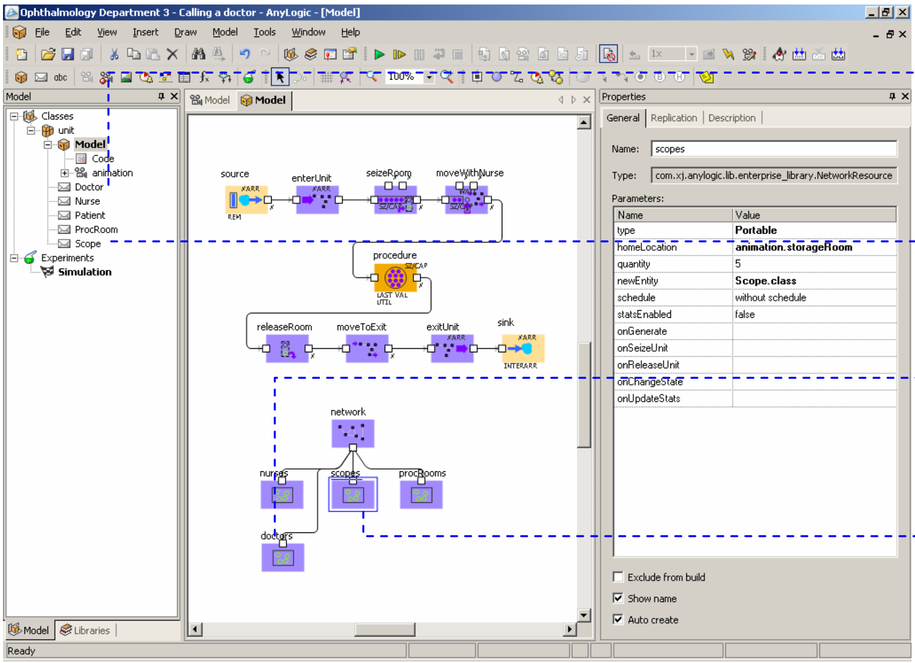
The reference model for this point is Examples \ Enterprise Library Tutorial Models \ Ophthalmology Department 2 - Room seizing.alp.

4.8 Adding doctor call logic

Now we will complete our model definition by modeling how ophthalmologist is called to the procedure room to perform the examination. Ophthalmologist picks an ophthalmoscope from the storage room on his way to the procedure room, and returns it back after the procedure.

First, we will create special message classes for the new resources and describe these resources using NetworkResource objects.

► Define doctor and scope resources



Name: scopes

Type: com.xj.anylogic.lib.enterprise_library.NetworkResource

Parameters:

Name	Value
type	Portable
homeLocation	animation.storageRoom
quantity	5
newEntity	Scope.class
schedule	without schedule
statsEnabled	false
onGenerate	
onSeizeUnit	
onReleaseUnit	
onChangeState	
onUpdateStats	

☐ Exclude from build

☑ Show name

☑ Auto create

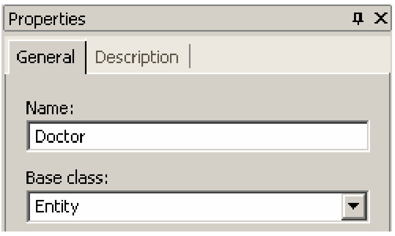
❶ Create Doctor message class

❷ Create Scope message class

❸ NetworkResource

❹ NetworkResource

❶ Create new `Doctor` message class – an entity representing an ophthalmologist. Derive new message class from Entity:



Name:
Doctor

Base class:
Entity

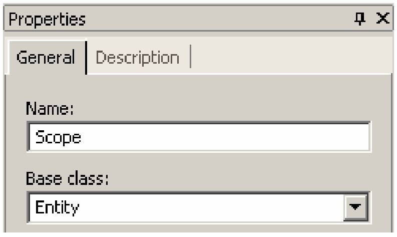
Ⓐ To create a `DoctorShape` symbol instance, write the following code in the *Additional class code* section:

```
Model._Group.DoctorShape shape =
((Model)Engine.getRoot()).animation.new
DoctorShape();
```

Ⓑ To set up the entity animation, write the following code in the *Constructor code* section:

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

❷ Create `Scope` message class – an entity representing an ophthalmoscope.

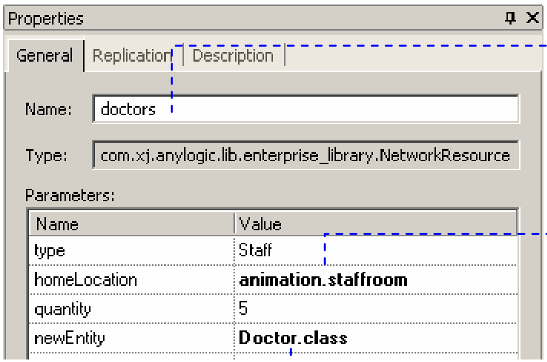


Name:
Scope

Base class:
Entity

Do not write any animation code for the `Scope` class since we do not need to change the default entity appearance.

❸ Add a NetworkResource object to describe resources of the Doctor type. Set the following properties:



Name: doctors

Type: com.xj.anylogic.lib.enterprise_library.NetworkResource

Parameters:

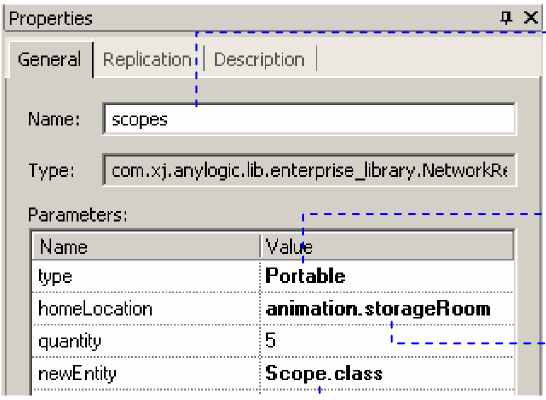
Name	Value
type	Staff
homeLocation	**animation.staffroom**
quantity	5
newEntity	**Doctor.class**

Ⓐ Name the object `doctors`

Ⓑ Set `animation.staffLounge` as the resources home location

Ⓒ Specify that resource units are messages of the class `Doctor`

❹ Add a NetworkResource object to describe a set of scope resource units. Set the following properties:



Name: scopes

Type: com.xj.anylogic.lib.enterprise_library.NetworkRe

Parameters:

Name	Value
type	**Portable**
homeLocation	**animation.storageRoom**
quantity	5
newEntity	**Scope.class**

Ⓐ Name the object `scopes`

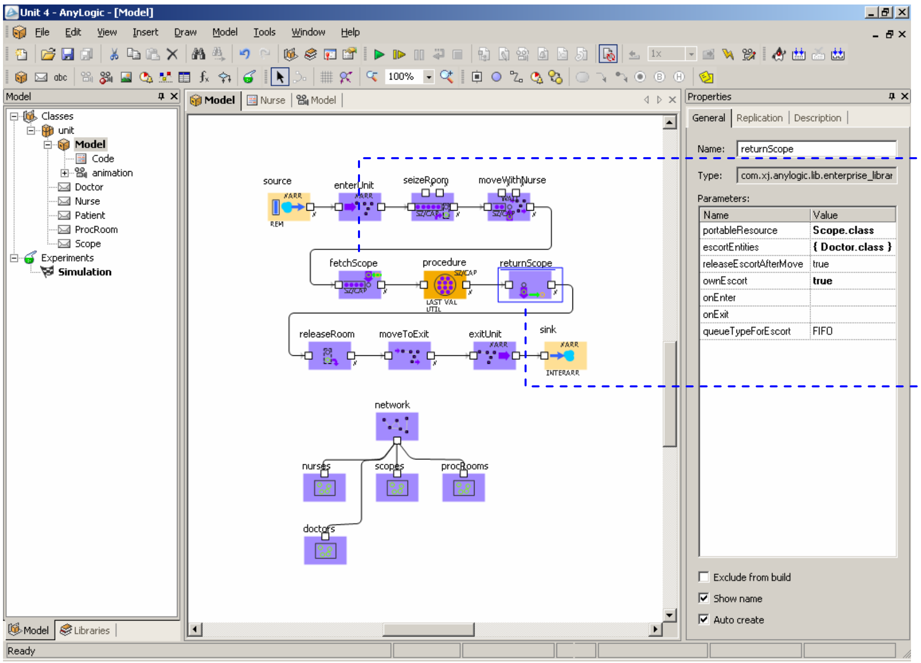
Ⓑ Specify that resources are portable

Ⓒ Choose `animation.storageRoom` as the resources home location

Ⓓ Specify that resource units are messages of the class `Scope`

Now we will add some objects to the flowchart to model doctor call.

► Modify the flowchart



Properties

General | Replication | Description

Name: returnScope

Type: com.xj.anylogic.lib.enterprise_librar

Parameters:

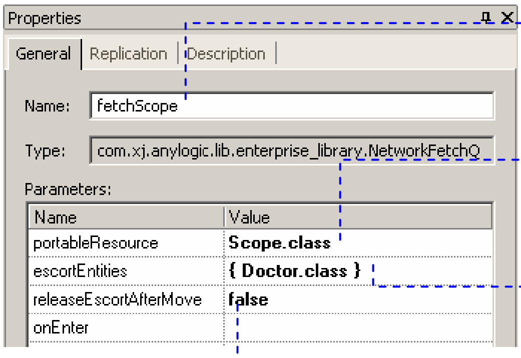
Name	Value
portableResource	**Scope.class**
escortEntities	**{ Doctor.class }**
releaseEscortAfterMove	true
ownEscort	**true**
onEnter	
onExit	
queueTypeForEscort	FIFO

☐ Exclude from build
☑ Show name
☑ Auto create

❶ NetworkFetchQ

❷ NetworkReturn

❶ Create a NetworkFetchQ object. The object allows fetching the portable resources by staff to a current location of the entity. Set the following properties:



Properties

General | Replication | Description

Name: fetchScope

Type: com.xj.anylogic.lib.enterprise_library.NetworkFetchQ

Parameters:

Name	Value
portableResource	**Scope.class**
escortEntities	**{ Doctor.class }**
releaseEscortAfterMove	**false**
onEnter	

Ⓐ Name the object `fetchScope`

Ⓑ Choose `Scope` as the resource to be fetched

Ⓒ Set `Doctor` as the staff resource to fetch the scope

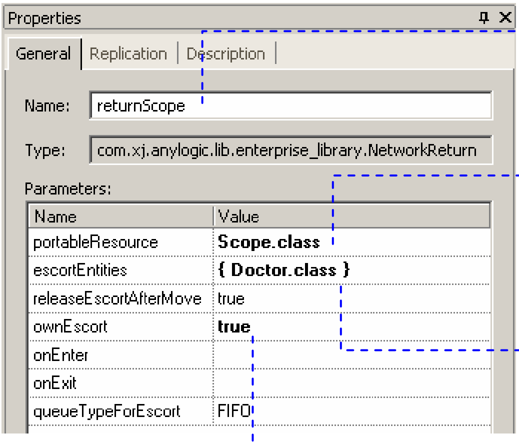
Ⓓ Specify that escort remains seized after fetching the scope

Ⓒ Choose Doctor as the type of the escort resource unit. If needed, portable resource can be fetched by a number of staff resources. See *Enterprise Library Reference Guide* for details.

Ⓓ After resource has been fetched the escort is released or remains seized. Since a patient need to be overhauled by a doctor, the doctor stays to examine the patient.

❷ NetworkReturn object returns the portable resources by staff. We add this object since ophthalmoscopes should be returned by doctors to the storage room after the procedure.

Set the following properties:



Properties

General | Replication | Description

Name: returnScope

Type: com.xj.anylogic.lib.enterprise_library.NetworkReturn

Parameters:

Name	Value
portableResource	**Scope.class**
escortEntities	**{ Doctor.class }**
releaseEscortAfterMove	true
ownEscort	**true**
onEnter	
onExit	
queueTypeForEscort	FIFO

Ⓐ Name the object `returnScope`

Ⓑ Choose `Scope` as the resource to be returned

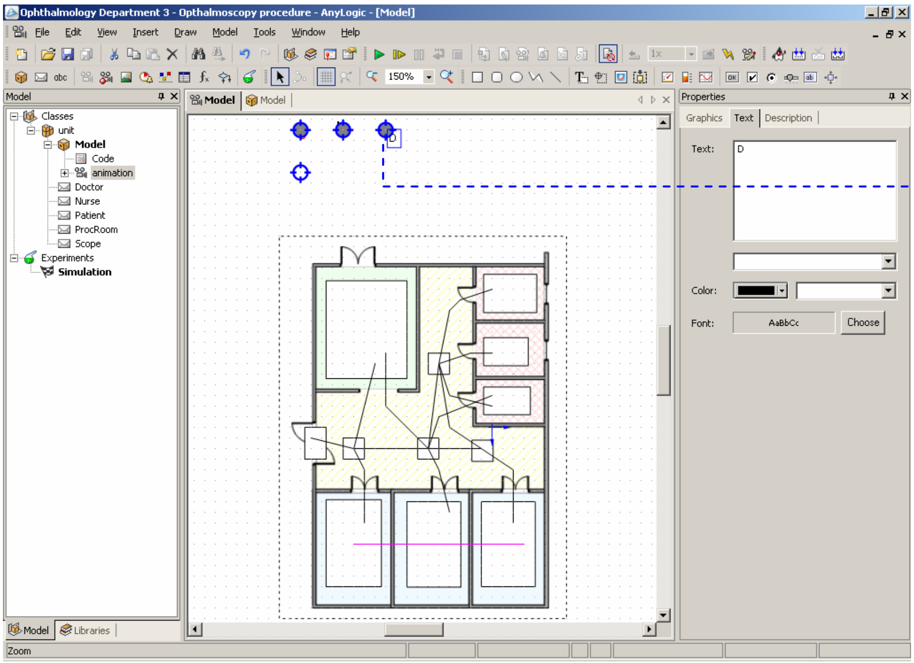
Ⓒ Set `Doctor` as the staff resource to return the scope

Ⓓ Use previously used staff to return the scope

Ⓒ An entity either can call a new staff to return the resource or can use a previously seized staff. We specify that the same doctor that has fetched the ophthalmoscope returns it to the storage room.

Now we will animate the doctors in the same way as we have animated nurses and patients before.

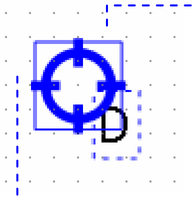
► Draw doctor animation



Text: D

❶ Draw doctor resource symbol

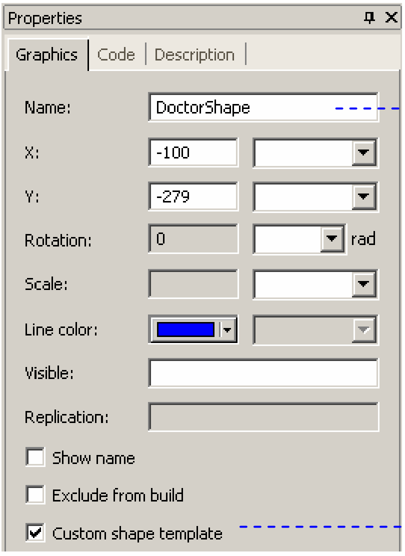
❶ Draw the animation for doctor.



Ⓐ Create D text label to indicate a doctor

Ⓑ Create a pivot and add the text label to the pivot group

🅑 Create a dynamic pivot.



Properties

Name: DoctorShape

X: -100

Y: -279

Rotation: 0 rad

Scale:

Line color:

Visible:

Replication:

□ Show name

□ Exclude from build

☑ Custom shape template

🅐 Name the pivot `DoctorShape`

🅑 Enable dynamic creation

We have finished creating our model of the ophthalmology department. Run the model by clicking the *Run* ▶ button. Now you can see the doctors coming to the procedure room with scopes to perform the patient examination.

The reference model for this point is Examples \ Enterprise Library Tutorial Models \ Ophthalmology Department 3 - Calling a doctor.alp

5. Conclusion

This tutorial has shown you the basic steps of creating models of different natures – in the fields of manufacturing, service and business processes. The Enterprise Library allows you to create models of different natures where discrete events happen – for example, parts processing during manufacturing or document processing during document circulation. There are many other spheres where discrete event simulation can be successfully applied. Moreover, in case you need to extend your model and go beyond pure discrete event simulation, you can seamlessly use any other AnyLogic™ modeling techniques in your model. For instance, you can use statecharts to describe complex, non-trivial behavior. For more information about the modeling techniques and ways AnyLogic™ supports, please refer to *User's Manual*.