A GENTLE INTRODUCTION TO LABELS

Simulation Software / TUTORIAL

A GENTLE INTRODUCTION TO LABELS

1.1 Introduction

Labels can be very useful when you start building more complex models in Enterprise Dynamics (ED). Labels are tags that can be attached to products in the model. They can represent properties such as color, weight, a bar code, service time and many other things. For instance, you could build a model where every product carries a label named *weight*. Besides a name, a label also has a value, which can either be a number or a string. The value of the *weight* label on a particular product might be the number `14', representing a weight of 14kg. Another product may also carry a label *weight*, but with value 16, for 16kg. It is possible for a product to carry multiple labels and the number of labels can differ per product.

In many modeling situations labels are used to distinguish between product types, assigning them a different service time or a different route through the system. Consider for example a model of the queues at the check in desk of airport where a distinction is made between economy travelers and business travelers that have access to different check-in desks. To model this, every passenger (product) is assigned a label *Type*. The value of this label could be either `Business' or `Economy', though in real modeling it's often more convenient to use number values such as 1 for business and 2 for economy. Based on the value of the label the traveler will be send to an appropriate queue.

Another application for labels is to keep track of historical information about a product as it passes through the model. In an airport setting one might use a label *CheckInFinished* to store the time at which the passenger left the check in desk. Later on this information can then be used to compute lead times: how long does it take business passengers to clear the check in desk, how long does it take them to reach the gate, etc.

1.2 Triggers

To use labels you first need to be able to assign them to products. This assignment takes place at a specific moment or place in the model, for instance when the product enters a certain queue. This is arranged by setting triggers. Besides settings such as the cycle time of server or the capacity of a queue most atoms also offer trigger fields. The most import triggers are Trigger on entry and Trigger on exit. These fields are triggered if a product enters or exits the atom. You can set a 4DScript command to assign the action that should be performed when the field is triggered.

Exercise 1: Triggers
Build a Source – Queue 1- Server – Queue 2 – Server – Sink model as depicted in figure 1. You only have to connect the atoms correctly. no adjustments have to be made to the settings of the individual atoms.

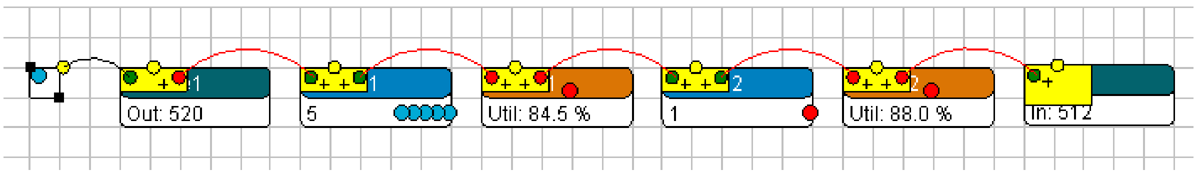

Figure 1: Model of a serial production line, with two machines separated by buffers.

Labels are one way to assign user-defined properties to product atoms, but all product atoms already have a property ‘Icon’. This property is used to visualized the product in the 2D window. You can view this property in the user interface of a product atom. Double click on the product atom and go to the Visualization tab. Double-click the field ‘2D Icon’. A window will appear, in which you can select an icon. Change the icon and run the model. You will see products with the new icon flow through the model.

You can also change the icon of an individual product atom during the simulation run. For example you can write a 4DScript command that changes the icon of a product when it leaves to first queue. To do so, write the following 4DScript command in the ‘Trigger on exit’ field of the first Queue:

```
Icon(i) := IconByName([circlered])
```

You can type this in yourself or you can open the list of the ‘Trigger on exit’ field and select the predefined logic:

```
Icon(i) := IconByName([?])
```

You now only have to change the question mark into one of the names of the available icons. The icons can be found in the Resource Manager which can be open from the main menu <Window | Resource Manager>. You can also add your own icons to the Resource Manager.

If you now run the model, you will see that the products are blue, but turned red after leaving the first queue.

1.3 Routing products

Besides using the trigger to change the icon of a product you can also use it to assign a label to the product. Having assigned a label to a product you can read it anywhere in the model and make different decisions based on the value of the label. For instance, you could send products to different queues or servers.

Exercise 2: Labels, predefined logic 7 Label direct

Build a model with two sources, which both send their products to the same queue. From the queue products can go to either one of two conveyor belts. They then leave the model through a sink, see figure 2.

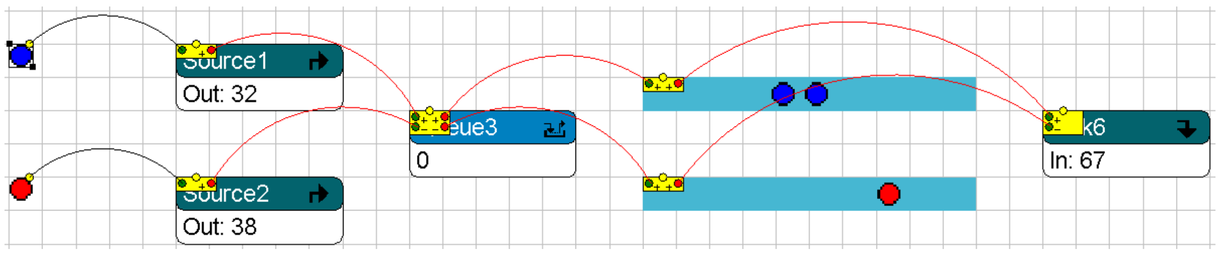

Figure 2: Model with two product types which are separated at the queue by label direct.

Make sure that the products connected to the sources are visualized by different icons by double-click the icon and go to the visualization tab. Double-click on the 2-D icon field and

select a different icon. In the example we used the circlered icon. If you now run the model, you will see both blue and red products in the queue.
We could separate these products again, such that the blue products go to the first conveyor and the red products to the second. Therefore we will introduce a label named *Type* which we will assign to all products. For the blue products, we will set the value of the label to 1 and for the red products to 2. We will assign these labels and set the value on the Source atoms. The moment a Source creates a product the Trigger on creation is executed. We can write a command on this trigger, which states what action should be performed when the product is created. In this case, we would like to assign a label to the newly created product and set its value. For this action, we can use one of the predefined logics. Open the listbox of the Trigger on creation of one of the source atoms and choose the following logic:

1. Assign label: products are assigned a label named LabelName with a value of 1

In a predefined logic the blue text can be changed. For the assign label logic we can change the name of the label and change the value. On both Source atoms we will change LabelName to *Type*. On the first Source we will leave the value to 1, but on the second we will change the value to 2.

All products that are created, will now be assigned a label named *Type*. We can read this label anywhere in the model. Since we would like to separate products in the queue and send them to different conveyors will do this on the queue. The ‘Send to’ property of an atom determines to which atom a product atom should go next. If this field contains the value 1 then all products will be sent to output channel 1. If the field contains the value 2 then all products will be sent to output channel 2. In our model all products have a label named type. The value of this label is 1 for the blue products and 2 for the red products. If we read the value of the label in the ‘Send to’ field, then the blue products will go to the first conveyor and the red to the second. Note that there is a direct correspondence between the value of the label and a number of the output channel of the queue we would like to send it through. We can again use a predefined logic to read out the label.

In the ‘ Sent to’ field of the Queue select the following logic:

7. By label value (direct): Use the value stored in the Label named LabelName and send to the corresponding channel.

We only have to change the LabelName to *Type* such that the correct label is read. There could be more than one label assigned to the products, therefore you should always state the name of the label.

After you've made all the changes to the model run the model and check that all blue products go to the first conveyor belt and a red go to the second.
Selecting

Exercise 3: Labels, predefined logic 8 Label conditional

We could also do it the other way around: if the value of the label *Type* is 1 we will send it to channel 2 and if the value is 2 will send it to channel 1. Reading out the label now isn't enough any more, but we can state a condition to create this behavior. If the value of the label *Type* is smaller than 2, which is only true for *Type* 1 products, we send the product to output channel 2, otherwise it is a *Type* 2 product and we sent it to output channel 1.

You only need to make a minor change to the previous model to create this behavior. Change the ‘Send to’ field of the Queue. Choose the following predefined logic:

8. By label value (conditional): if the value stored in the label named LabelName is < than 1 then send to channel 1 else 2.

Change the blue text, such that the blue *Type* 1 products go to channel 2 end the red *Type* 2 products go to channel 1. After you've made the changes check that the red products go to the first conveyor and the blue to the second, see figure 3.

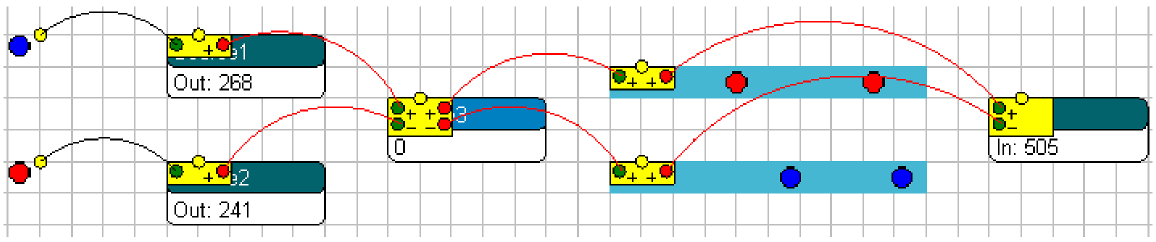

Figure 3: Blue Type 1 products are sent to the second output channel of the queue by label conditional.

1.4 Bernoulli distribution

In all previous exercises we used two Source atoms to create the product types. We can also use one Source to create both. All products will be assigned a label named *Type*, but for a certain percentage the value of the label will be set to 1, and otherwise to 2. For this we need a Bernoulli distribution. This is a discrete probability distribution, which takes two values. The 4DScript function for this distribution takes three parameters: **`Bernoulli(e1,e2,e3)`**. For e1 percent of the cases the function takes the value e2 otherwise it will take the value e3. If we for example execute the expression **`Bernoulli(40,6,10)`** then in 40% of the cases the return value will be a 6 and in 60% of the cases a 10.

Exercise 4: Bernoulli

Build a model such as depicted in figure 4. Select the Assign Label predefined logic on the ‘Trigger on creation’ of the Source. Change the LabelName and change the value to **`Bernoulli(80,1,2)`**. You should now have the following logic:

1. Assign label: products are assigned a label named Type with a value of Bernoulli(80,1,2)

Every time a product is created a label named *Type* is assigned to it. For most of the products (80%) the value of the label will be set to 1, and for the others (20%) it will be set to 2.

To be able to distinguish the two product types in the 2-D window, write the following command on the ‘Trigger on exit’ of the Source:

```
If( Label([Type], i) = 2, Icon(i) := 26)
```

This command checks if the label named *Type* of the product that is leaving the Source is equal to 2 and if so changes the icon of this product to icon number 26. After this change the ‘Send to’ field of the Queue such that *Type* 1 products go to output channel 1 and *Type* 2 products to channel 2. The name of the label should be the same name as the name of the label that we assigned to the products on the Source. Run the model and check the results.

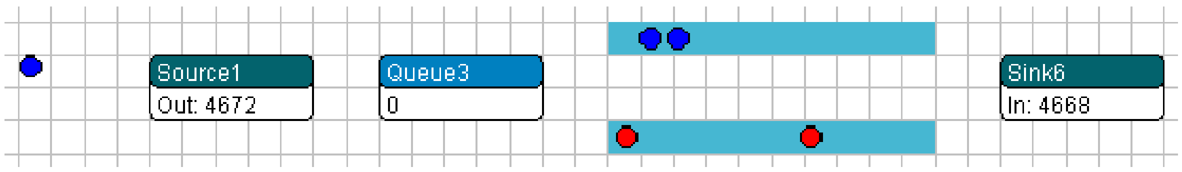

Figure 4: A source creating two product types which are distributed to 2 different conveyors.

In this exercise, we only needed to distinguish two products types and we could use the Bernoulli distribution to set the value of the label. If there is a need to distinguish more products than one could use an Empirical Distribution atom. Use this atom to create your own discrete probability distribution.

1.5 4DScript and Labels

Behind each of the predefined logics is a 4DScript command. If you start to build models for real applications you will find that you cannot build these models using predefined logic alone. You will need some basic knowledge of the 4DScript language to be able to build these models. Therefore will now take a better look at the code behind the logics of section 1.3.

Before we can start looking at the 4DScript code we first need to know a little bit about ‘atom references’. Properties and labels belong to specific atoms. If for example, we would like to assign a label or read a label of an atom we need to make clear which atom we mean. For this we use ‘atom references’. The most commonly used atom references are ‘i’ and ‘c’. If you have a 4DScript command on the ‘Trigger on entry’ of the Queue, then you can use the reference ‘c’ to refer to the Queue itself. You would use this if the moment of product enters a Queue you would like to change the property of this Queue. The ‘c’ stands for current and refers to the atom on which the command is written.

Another useful atom reference is ‘i’. This reference is frequently used in trigger fields. The ‘i’ stands for involved and refers to the atom that caused the trigger to execute. If you write a command with both ‘i’ and ‘c’ on the ‘Trigger on creation’ of the Source, then ‘c’ refers to the Source and ‘i’ refers to the product that was just created and which caused the trigger to be executed. But if you use ‘i’ and ‘c’ in a command on the ‘Trigger on exit’ of a Server, then the ‘c’ refers to the Server and the ‘i’ refers to the product that just left the Server and which caused the trigger to be executed.

Let’s examine the code behind to predefined logic ‘Assign label’, which we used on the ‘Trigger on creation’ of the Source. Open the user interface of the Source, open the listbox of the ‘Trigger on creation’ and select the ‘Assign label’ logic. To view and edit the 4DScript code of this predefined logic press the square button below the button to open the listbox. If you press this button the following code will appear:

Trigger on creation of the Source: `Label([Type], i) := 2`
Explanation: Each time a product is created this code is executed. In this example a label named *Type* is assigned to the product that has just been created, the ‘i’ will refer to that product atom. The sign ‘:=’ is used to assign the value 2 to the label.

You can also write a 4DScript expression that results in the value to a label. An example of that is the following:

Trigger on creation of the Source: `Label([Type], i) := Bernoulli(80,1,2)`

Explanation: Each time a product is created this code is executed. When the Bernoulli function is executed it returns a value, which in 80% of the cases will be 1 and in 20% of the cases will be at 2. This value will be assigned to the label *Type* on the product that has just been created.

Anywhere in the model this product goes we can always read out its label. We could for example write a command in a 'Cycletime' field, such that different product types will have different cycle times or we could write a command on the 'Send to' field of an atom to control the routing of different products types. To be able to do this we need to know how to read the value of a label and how to use the result.

Earlier, we already saw an example on the 'Trigger on exit' of the Source:

```
If( Label([Type], i) = 2, Icon(i) := 26)
```

The 4DScript command: **`Label([Type], i)`** returns the value of the label *Type* on atom 'i'. Since this code is executed at the moment a product atom leaves the Source, the 'i' will refer to the leaving product. In our example, the return value will either be a 1 or a 2. This value will be compared with the value 2. If the value of the label is 1 then the comparison is false and nothing will happen. If the value of the label is 2 then the comparison is true, and the command **`Icon(i) := 26`** will be executed. This command will change the icon of the product leaving the Source to 26.

We also used the label *Type* in the 'Send to' field of the Queue, such that *Type* 1 products go through output channel 1 and type 2 products go through output channel 2. In a 'Send to' field the result of a 4DScript expression should be the number of the output channel through which you want to send the product. If you write a 2 in this field, then all products will be sent through output channel 2. In the example, all products of which the value of the label *Type* was equal to 1, i.e. *Type* 1 products, were send through output channel 1, and if the value was equal to 2 it should be send through output channel 2.

Send to of the Queue: **`Label([Type], First(c))`**
Explanation: This command returns the value of the label named *Type* of the product atom 'First(c)'. 'First(c)' is just like 'i' and 'c', an atom reference. There can be more than one product atom in the queue, thus we have to make sure that we read the label of the product that is first in queue. Here, we cannot use the reference 'i' because the first product in the queue is not the product that triggered this field. If you need to refer to a product atom from a field which is not a trigger field you can make use of the relative atom references 'First(c)' and 'Last(c)'. The 'c' again refers to the current atom on which the code is written. 'First(c)' and 'Last(c)' refer respectively to the first and last product that is contained in the atom 'c'. If only one product is contained in the current atom then both 'First(c)' and 'Last(c)' refer to the same product.

1.6 Example model: Bank

Customer waiting times are regarded as one of the most critical aspects of service quality. At a consumer bank they therefore want to investigate the waiting time performance of two alternative concepts for serving their customers. At the bank they distinguish several customer types.

For simplicity they assume there are two customer types, each with different service times. On average 50 customers per hour arrive at the bank with a service time of 1 minute. We will indicate these customers as type A customers. In addition, on average another 5 type B customers arrive, with a service time of 10 minutes. There is one single queue for both customer types, serviced by two counters according to the ‘first come, first served’ principle. Furthermore, we assume that all arrival processes are exponentially distributed and that all service times are constant.

First build the model with a single Queue.

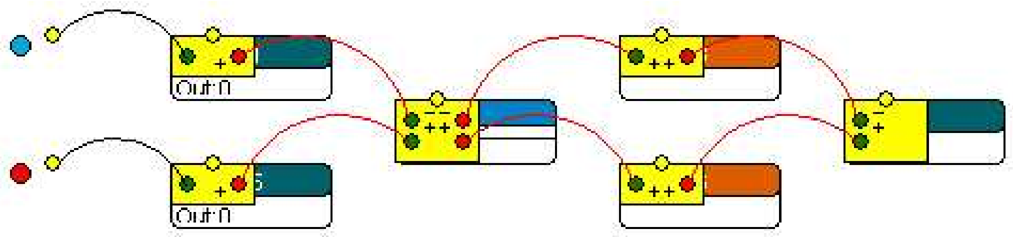

Figure 5: Model of the bank with two customer types.

To distinguish the customers in the simulation model, type A customers are represented with a blue icon and type B customers with a red one.

The time needed for a bank teller to serve a customer has to be known on the servers. This service time, however, depends on the customer type. We can solve this problem by using labels. The idea is to determine the service time at an early stage and store this time, in a label on the customer. In the server we only have to read the label, we do not have to check for the customer type. This approach can easily be extended for three or more customers.

Write code on the ‘Trigger on creation’ of the sources to attach a label named *servicetime* on each of the customers. The value of the label is set to the correct service time. In both Sources select the following predefined logic:

1. **Assign label: products are assigned a label named LabelName with a value of 1**

Change the name LabelName on both Sources to *servicetime* and set the value to the service time. For the Source that creates the type A customers the value should be 60 (1 minute) and 600 (10 minutes) for type B customers. Recall that in ED times should always be given in seconds.

We should now make sure that the cycletime fields of the Servers are set to read the label *servicetime* and thus use the value stored in the label as service time. Select the following predefined logic in the cycletime field on the Servers

```
Label([Labelname], First(c))
```

and change the LabelName into *servicetime*. The 4DScript command **`First(c)`** is just like ‘i’ and ‘c’ an atom reference. The ‘c’ refers in this setting to the server, **`First(c)`** is a relative atom reference which refers to the first product in the server. The label on the product tells us how long the cycle time should take.

Run the model for several hours and use the summary report to check if the service time is as expected.

1.7 Determining lead times

Exercise 5: Using labels to determine the lead times

In this exercise we will determine the lead times through part of a production line. Build a simple model with two serial servers, with queues before both of the servers, see figure 6.

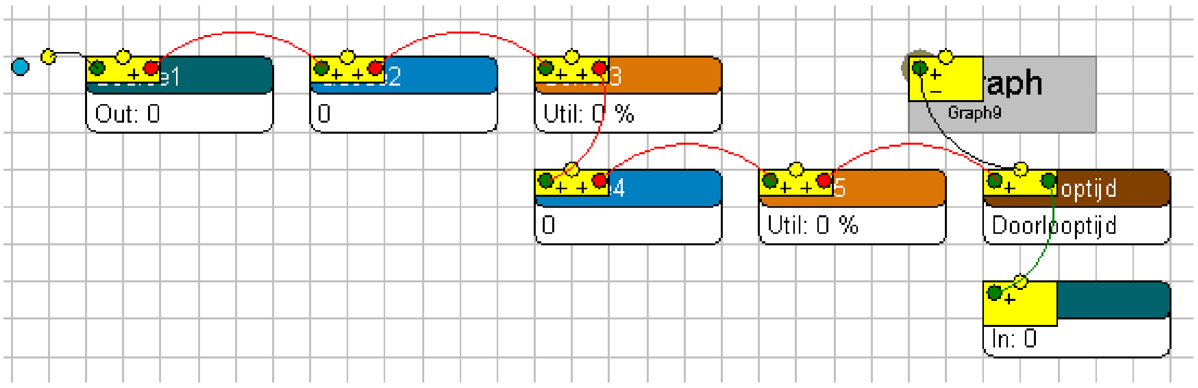

Figure 6: Model of a serial production line with two machines separated by buffers and a Data Recorder to measure the lead times.

We will measure the lead time, starting at the time that products enter the second queue until they are processed by the second machine. On the 'Trigger on entry' of the second queue, attach a label *EntryTime* to each of the passing products and set the value to **`Time`**. The 4DScript command **`Time`** gives the current time in the simulation model in seconds. We can assign a label to each of the passing products in the 'Trigger on entry' with the 4DScript code

```
      Label([LabelName], i) := value
```

The 'Trigger on entry' will be executed the moment a product enters the queue. **`Time`** returns the current time in the simulation model. In this case, it will thus return the time that the product entered to queue. This is the value we want to assign to the label.

We also add a Data Recorder, which can be found in the Results section of the library, to the model, which we place behind the second server before products leave the model through the sink. A Data Recorder atom can be used to record data of passing products in a table. We would like to record the lead of the passing products. Therefore we need to measure the time starting when products enter the second queue until they arrive in the Data Recorder.

Double click on the Data Recorder and go to the Variables tab. Add a variable Throughput with the following 4DScript expression: **`Time – Label([EntryTime], i)`**

Again, **`Time`** will return the current time in the simulation model. In this situation, it will thus return the time that the product enters the Data Recorder.

The next step is to draw a graph of the lead times. Drag a graph atom from the library in the model and connect its input channel to the central channel of the Data Recorder. Double-click on the graph atom and go to the Specific tab. Change the Value to **`Cell(gp, 1,`**

`in(1,c))`. This code reads the values from column 1 of the table connected to the first input channel of the graph in this case the Data Recorder. Reset the model and run it for several hours. Right click to graph atom to view the graph of the lead times. It is also possible to use the Data Recorder to export the data to excel and use Excel to make a graph of the lead times.