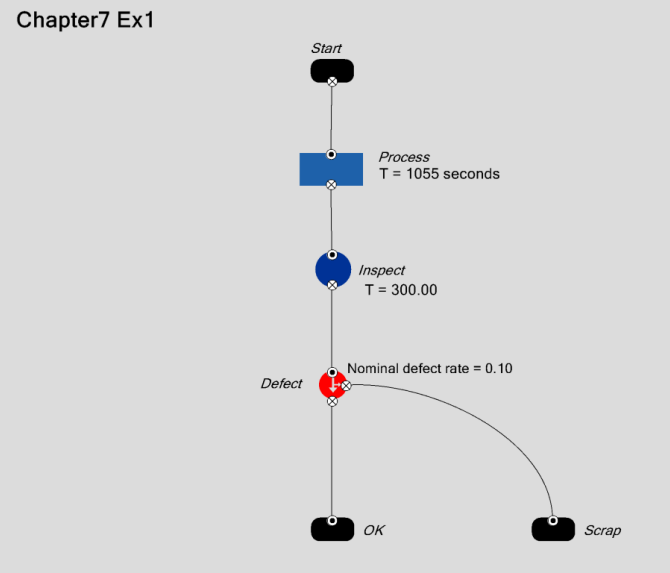
Chapter 7 – Capacity

A process does not make a part directly. It is the blue print of what happens to make a part. With the timings we have looked at how long tasks or actions take. But you may have noticed that in our process models, we are simultaneously running the process for up to 10000 items (determined by our sample size) . A real lathe can usually only do one at a time. In this chapter we are going to look at production capacity constraints and how they can be modelled. We will start with a simplistic approach. We will then look at a resource approach and finally we will look at a gate based approach. All of these modelling techniques have advantages and disadvantages. You get to choose the ideal one for your application.

The simplistic method – putting a capacity on a delay.

In Aliquis terms, all the process blocks you have used are actually delays. Lets go right back to a highly simplified version of our hub process :

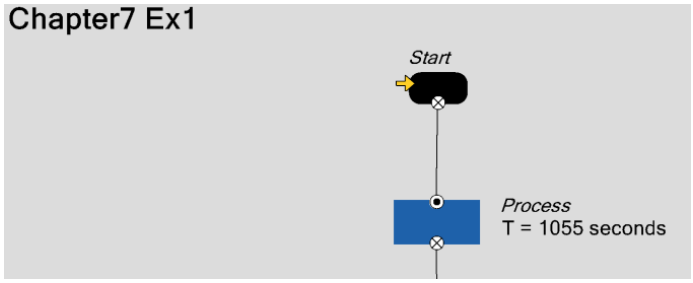
Open *Chapter7Ex1.inp*

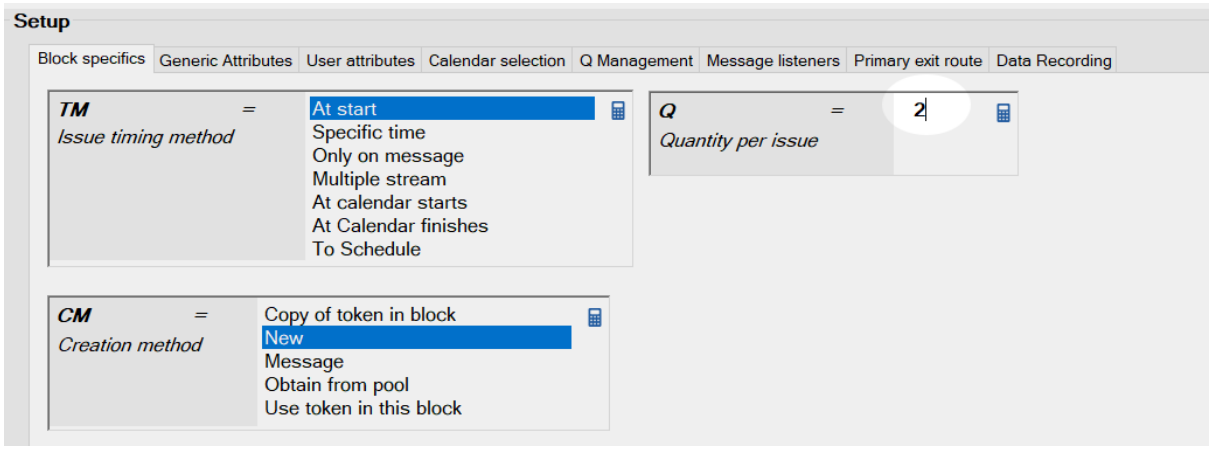


What happens if you step solve this :

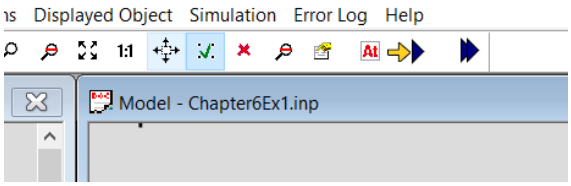
We will now adjust the start block so it emits 2 tokens at the start of the solve :

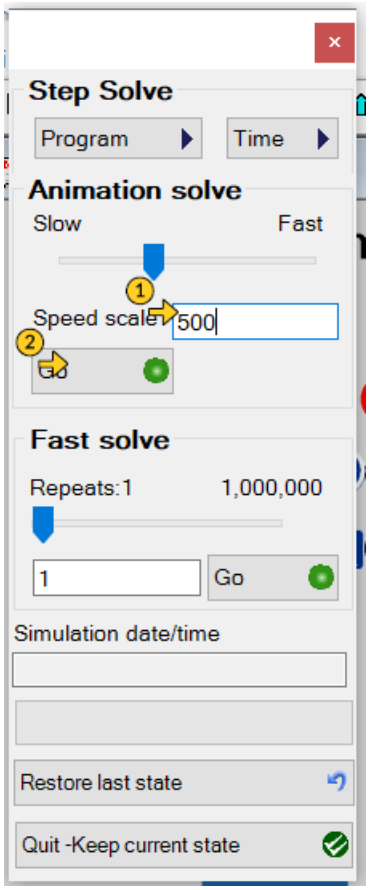
Click the block :



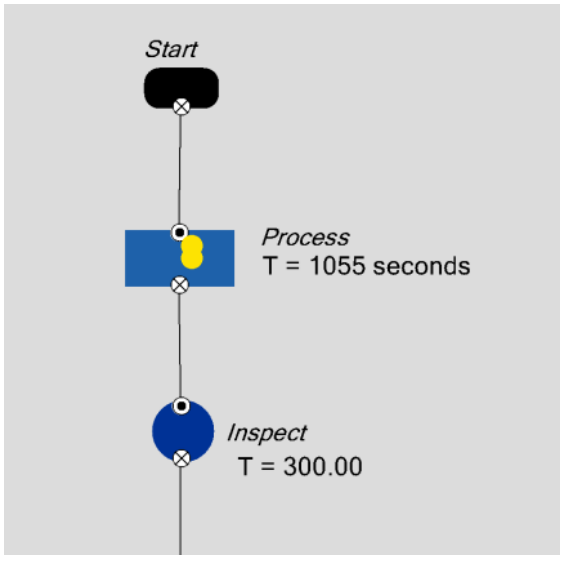


Bring up the solver and animate . It is suggested that you use a time scale of around 500.



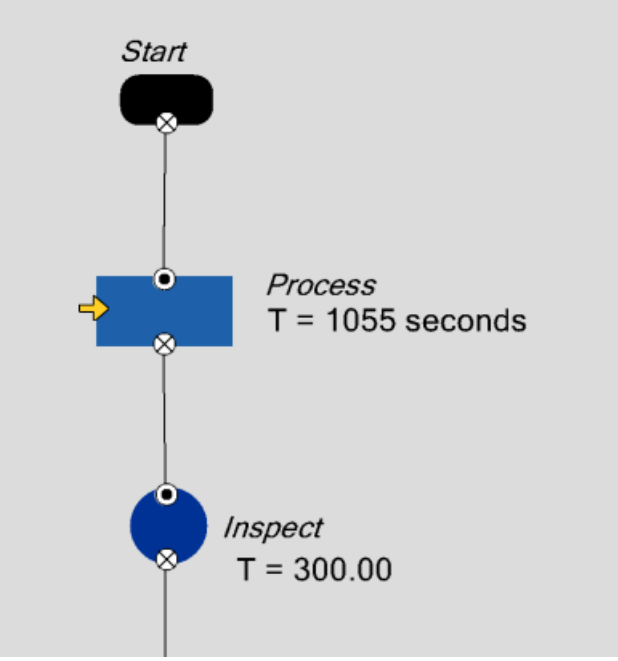


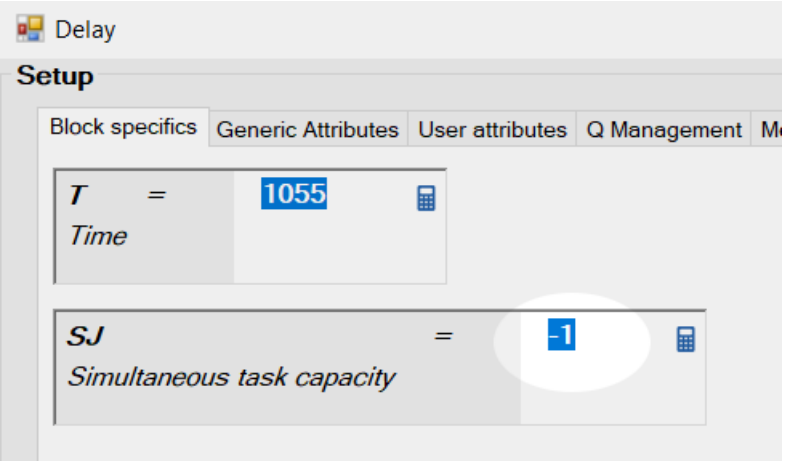
You will see a tokens go through in pairs :



Why – because the capacity of the delay blocks is defaulted to infinity.

Quit the animation and edit the first process block :





Notice the simultaneous task capacity is set to -1 . This is interpreted as infinity.

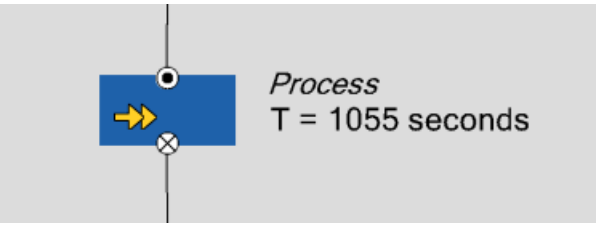
Change it to 1

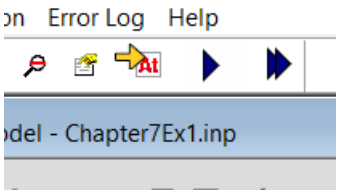
Do the same on the inspection block.

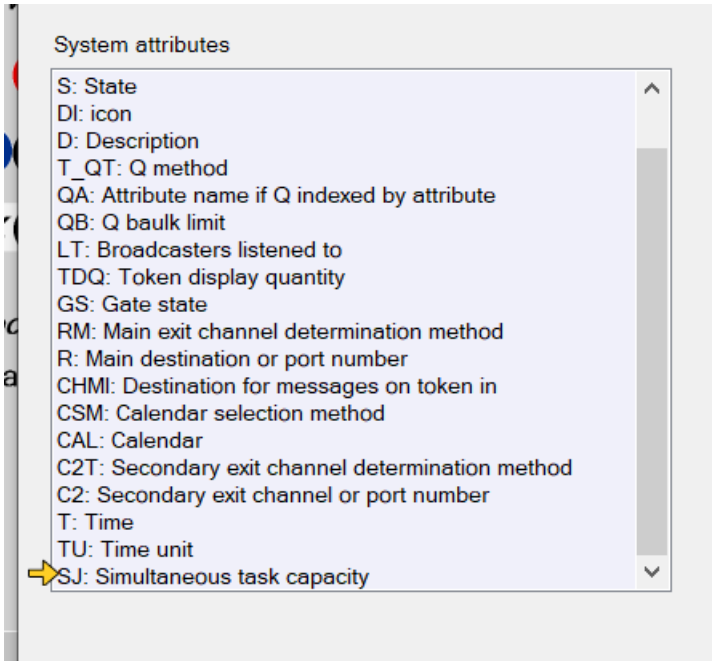
Try the animation again.

Now the process block only releases its tokens one at a time. The first one is released at 1055 seconds and the second at 2110 seconds.

Now in Aliquis you can always edit the blocks in this manner but don't forget you can also expose the system attributes for the user to see :

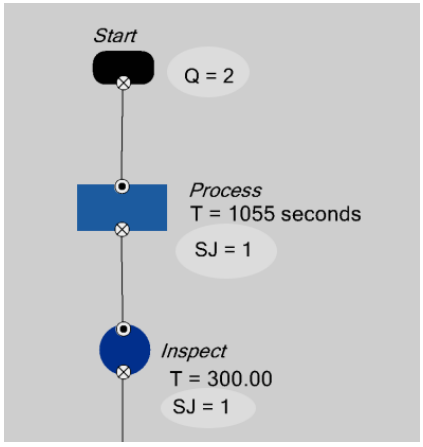






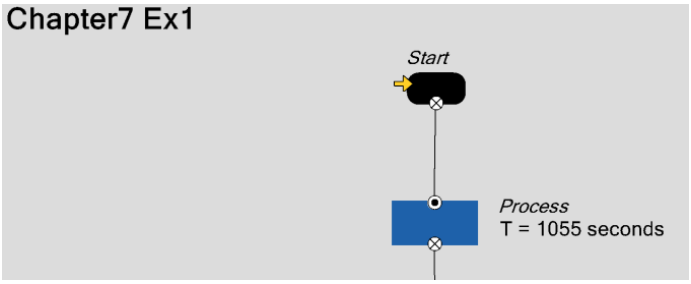
Result :

On your own – expose the other key block attributes like this:

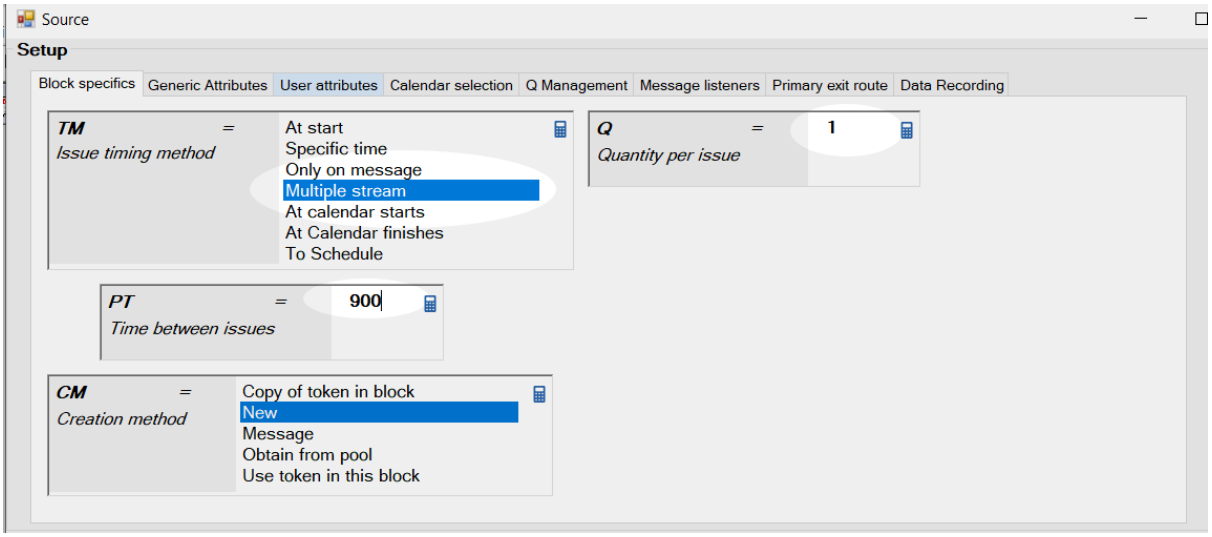


Take some time, using the animator to investigate how this runs. Also play with the process times – can you shift the bottleneck to the inspection process? What does it look like?

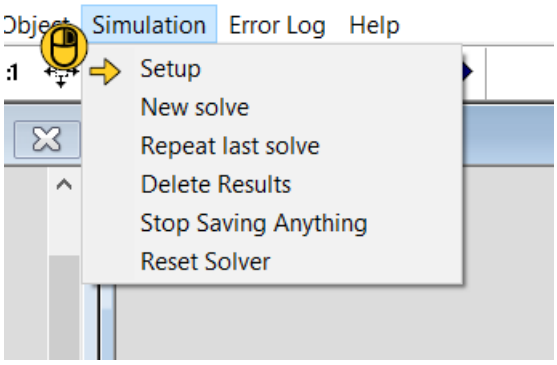
One last change to the source which might be useful :

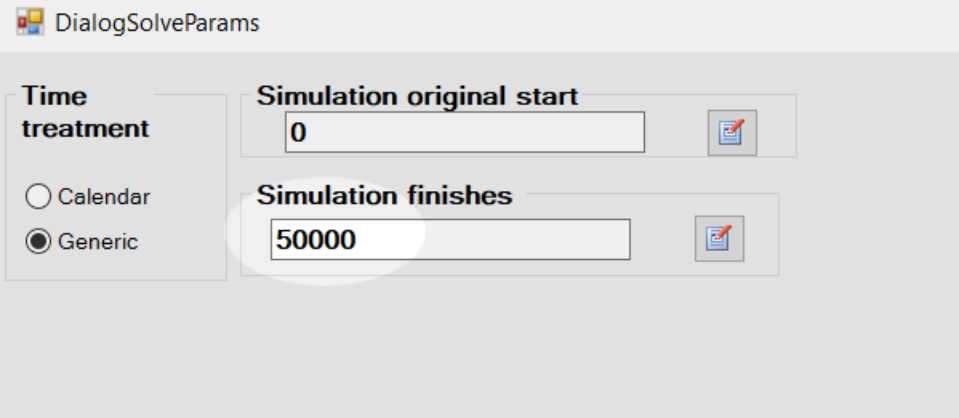


We will now set it up as a periodic emitter. Like this :

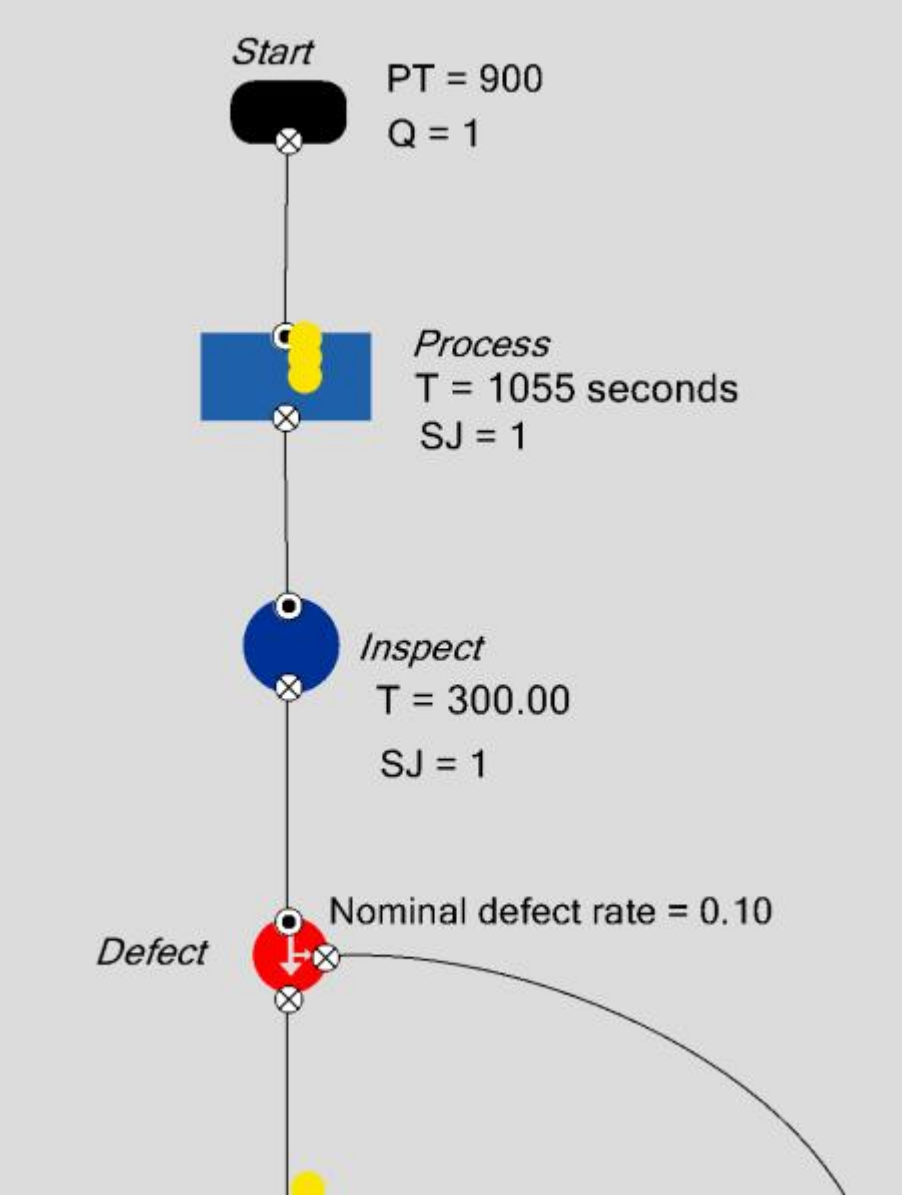


You might like to extend the length of the solve (it defaults to 5000 seconds which is not long enough)





Animate the solve again :



You should see a bottleneck building up on the process block. How can you get rid of it ?

By all means user chapter7Ex1a as a reference model at this point.

The other thing you will observe is that the inspection block is not fully utilised – it often contains no token – indicating no activity.

The idea of “line balancing” is to get all your production facilities operating at high utilisation while avoiding the formation of bottlenecks. We will expore this further in chapter 8.

For now, let’s pose another question. Suppose the inspection were done in the machine. Let’s also assume that the machine has a production capacity of 1.

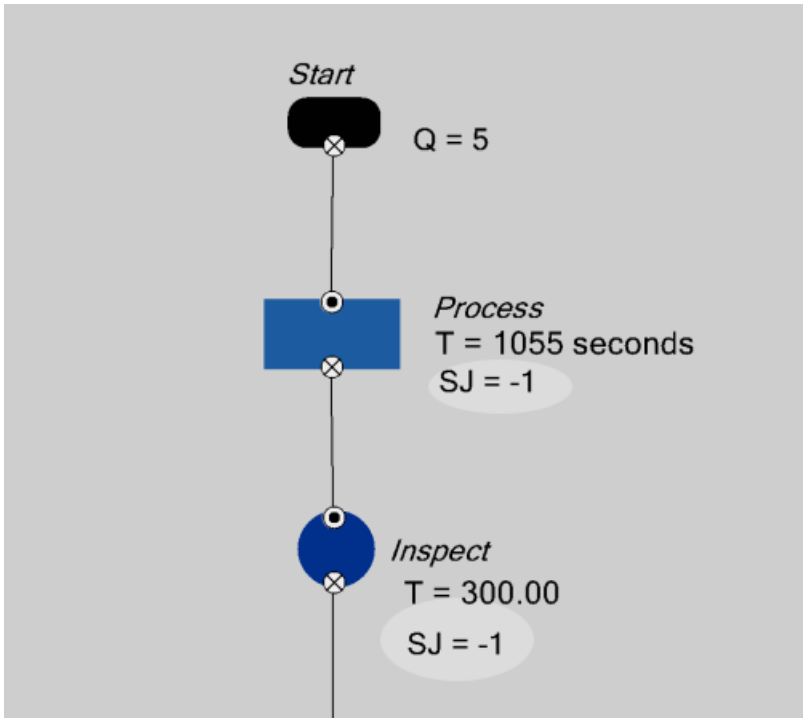
In this case we would not be able to inspect and machine at the same time! Unfortunately this model does allow this to happen. There is activity in both processes.

We will now offer two methods to deal with this. The first method is to treat the machine as a resource which is used by the process while work is taking place.

Resource based capacity constraints

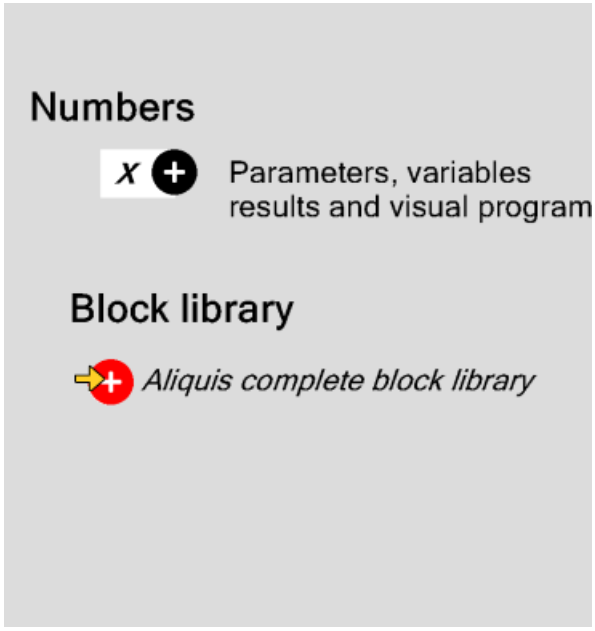
This approach is perhaps more intellectually satisfying than others. The basic idea is that any process element will need a resource to run. A resource could be a person, a tool, or a machine. It could also be a consumable such as a fastener or a quantity of paint. Aliquis has a comprehensive way of attaching and detaching resources to tokens as they go through a process. Lets go this simple example.

Start with chapter7Ex2

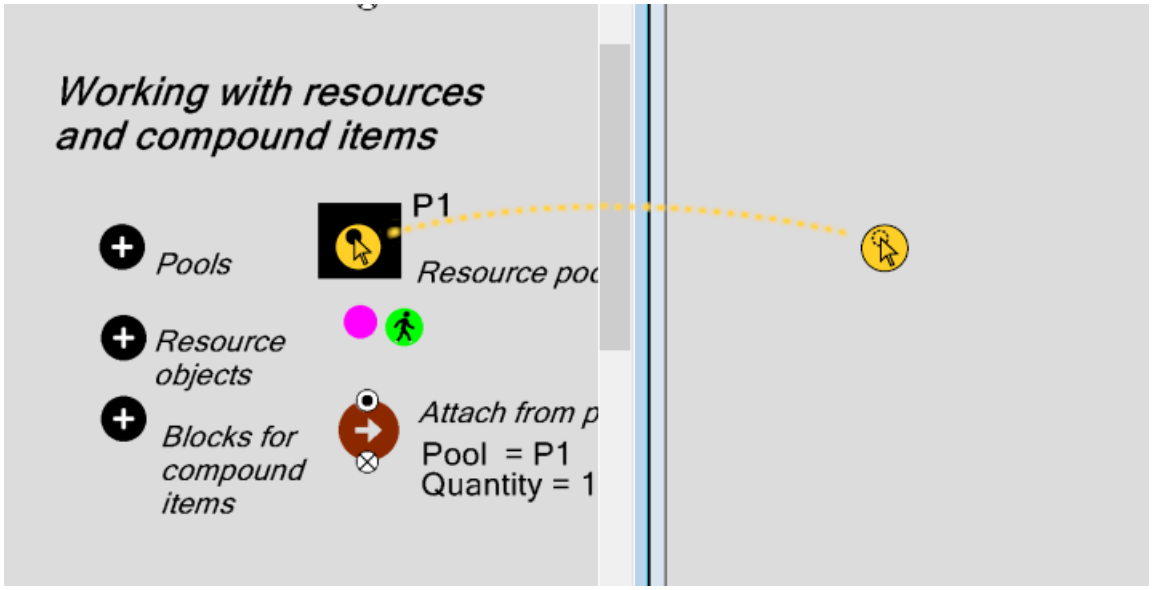


Note the capacities have been set back to infinite – we are not controlling capacity with the blocks.

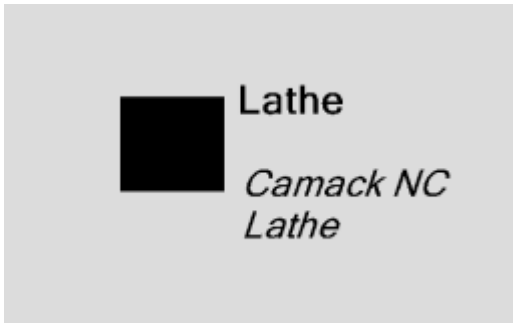
Go into the Aliquis complete block library :



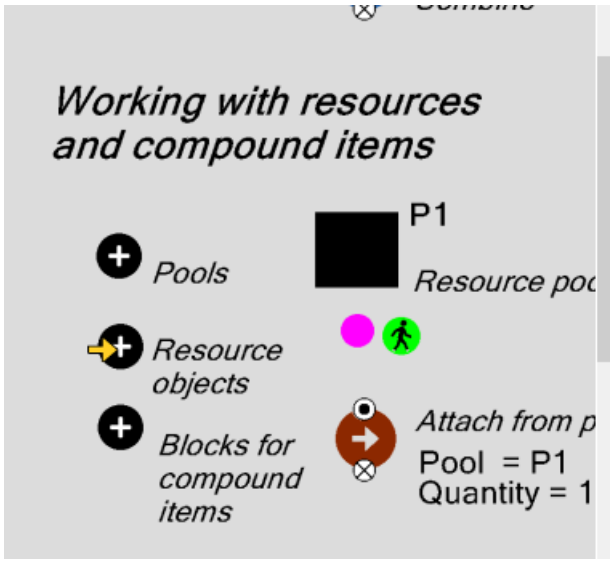
From here find Drag a resource pool into the model:



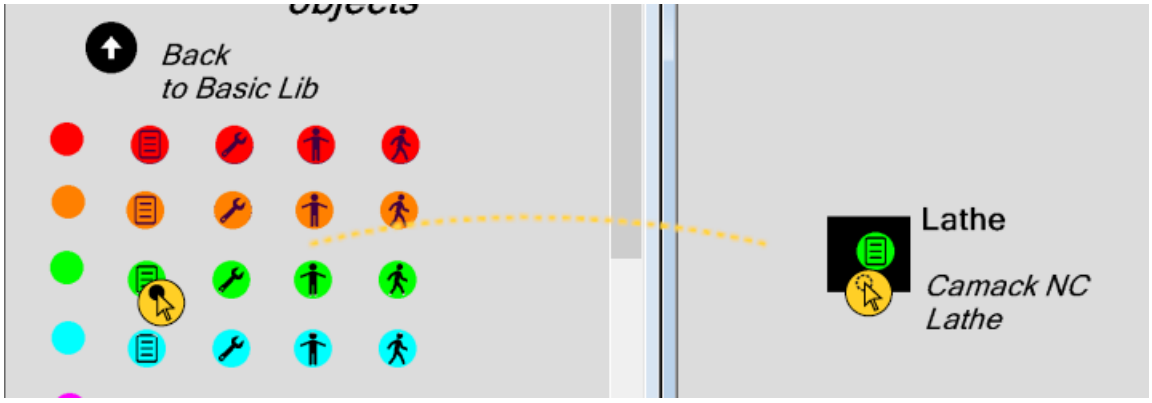
Name it “Lathe”



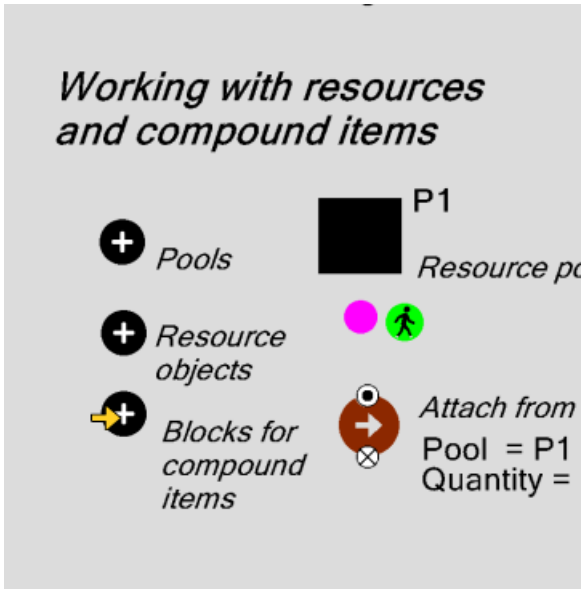
Go into resource objects :

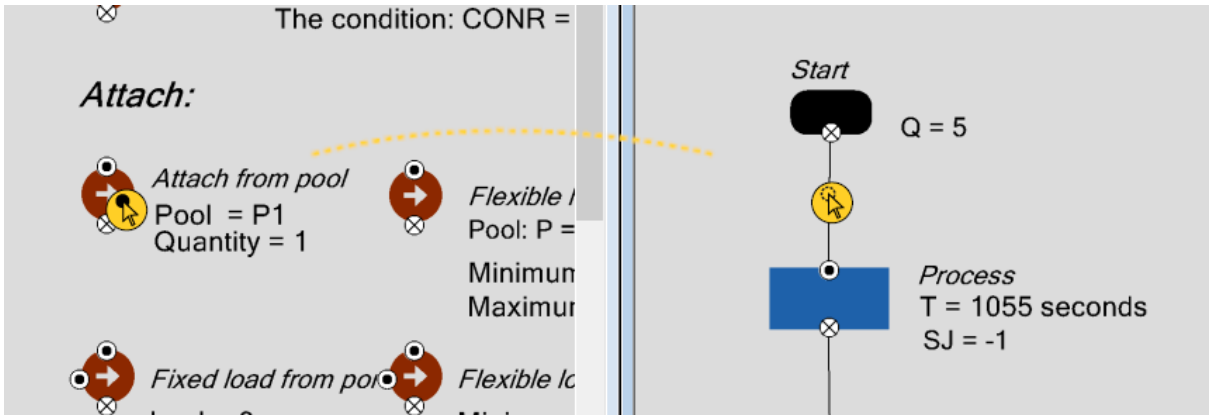


Drag something that vaguely looks like a machine into the pool :



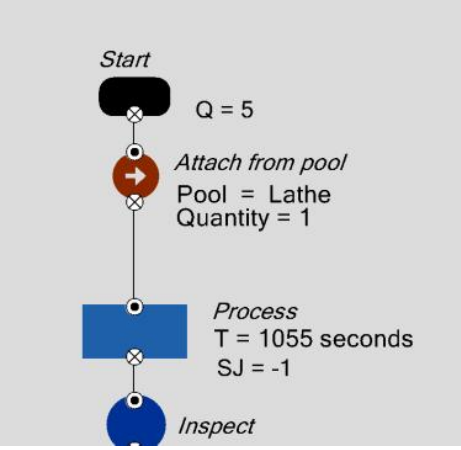
Now we will put a resource into the process at the point we need the resource :



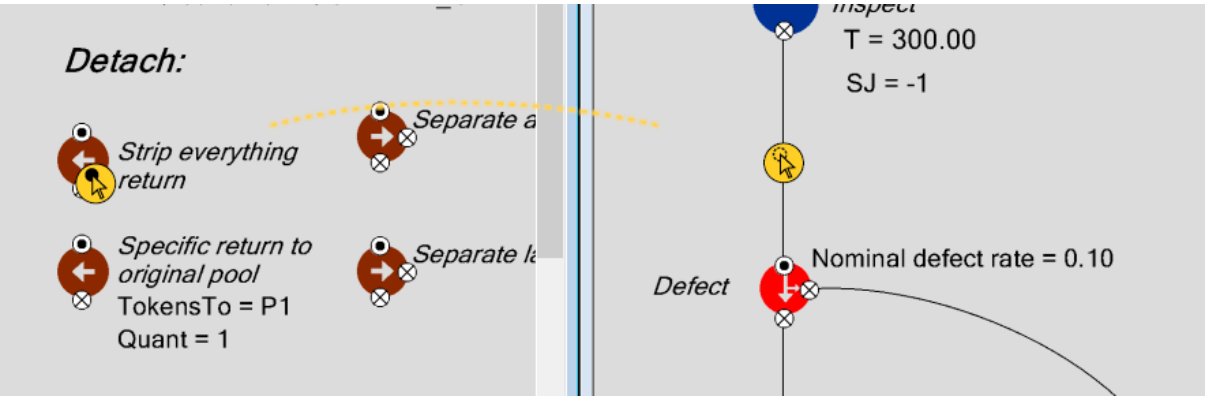


Change the name of the pool to lathe :

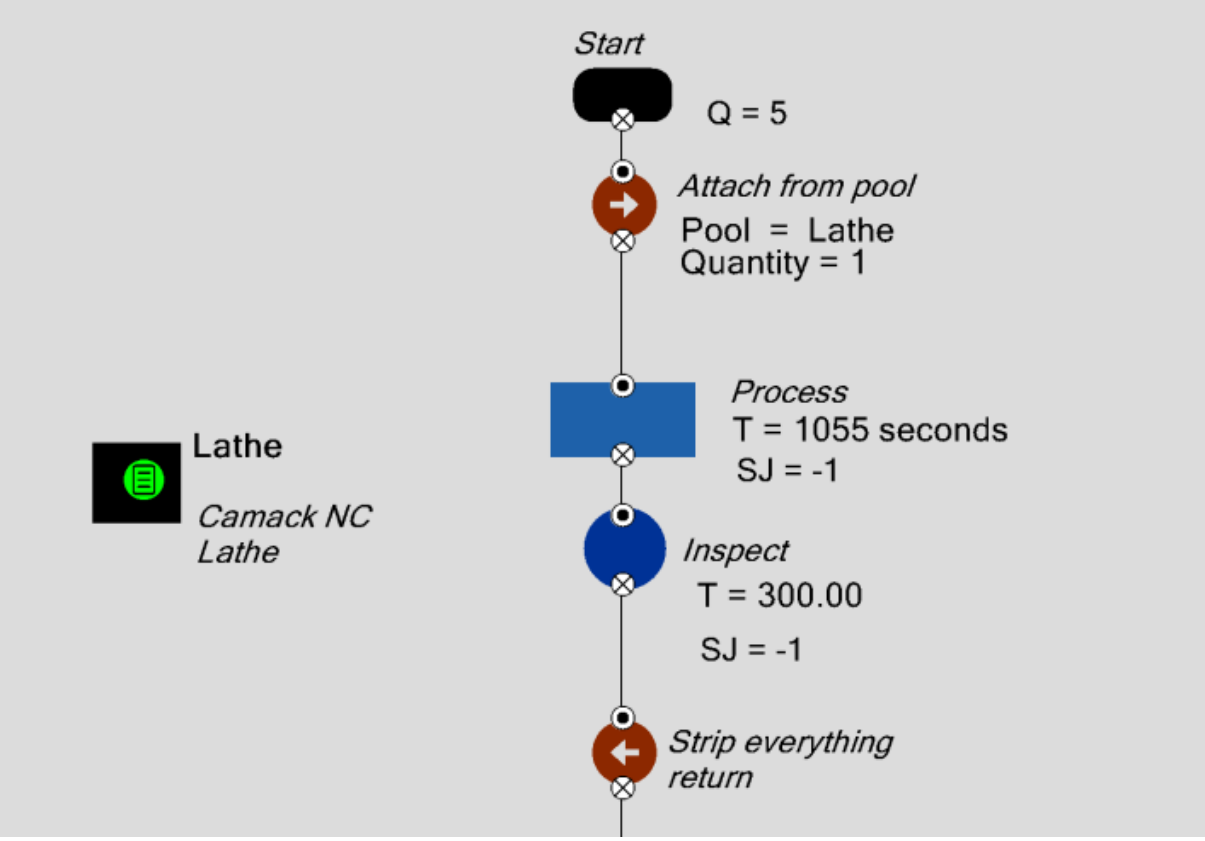
Result :



Now place a resource return where the process is done :

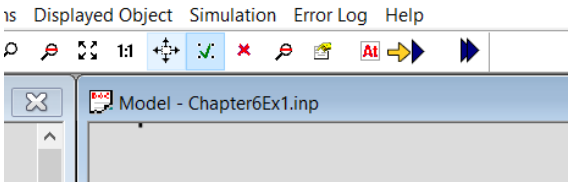


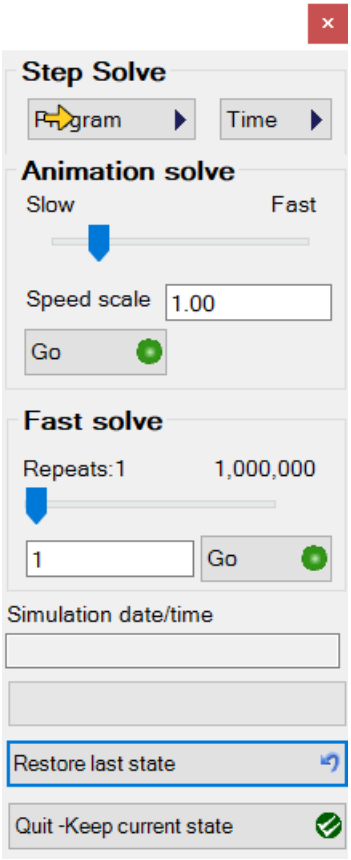
Result :



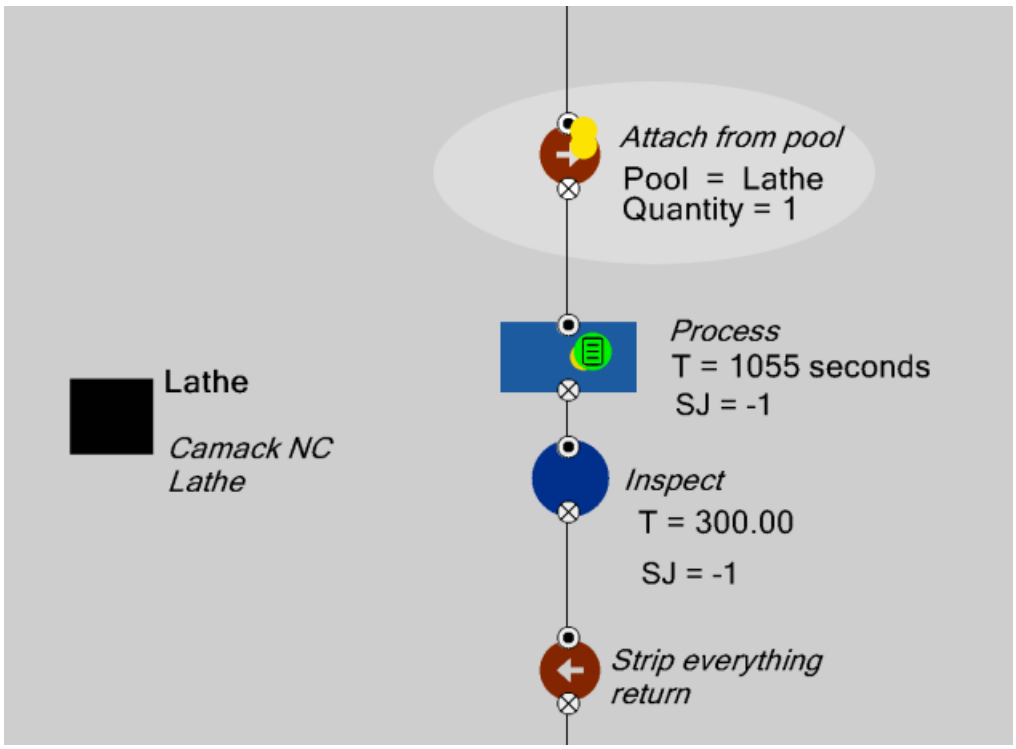
If you want a reference model for this it is *Chapter7Ex2a*

Rather than animate it is probably better to step solve to visualize how this is working :



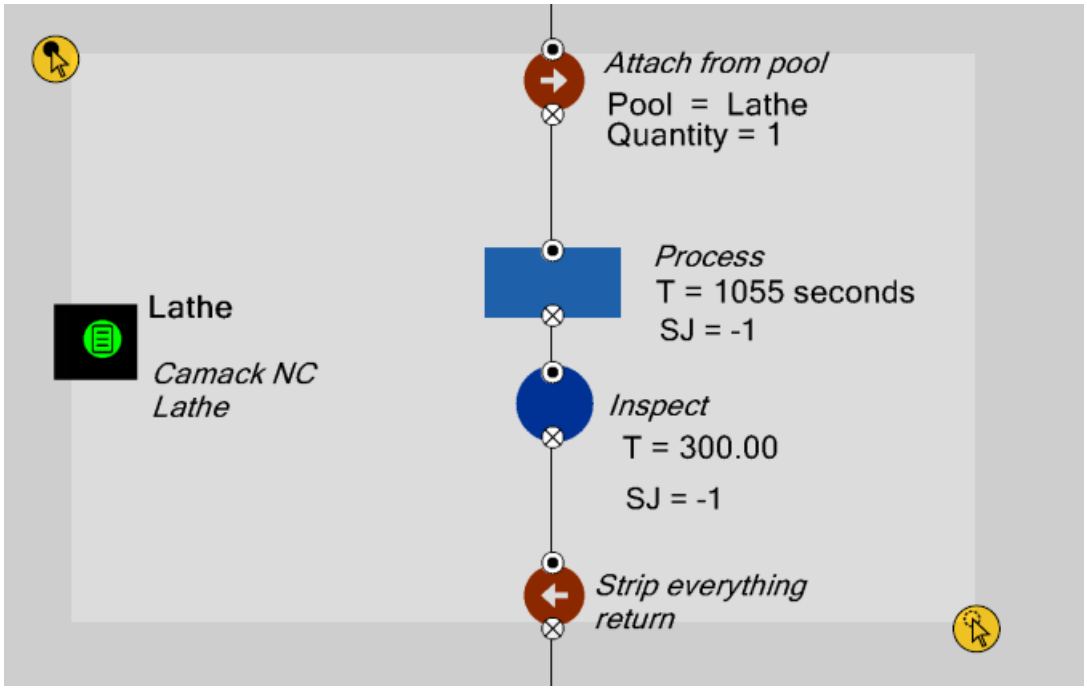


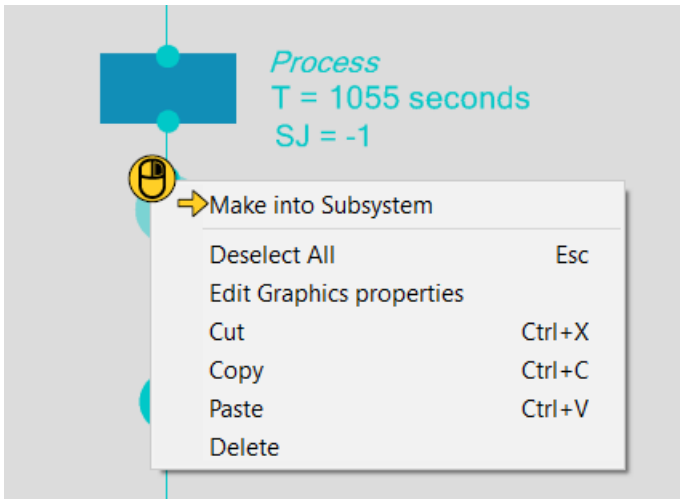
You will see that the resource block does not allow things through until the resource is available.



You will only ever see one item being worked on in the process.

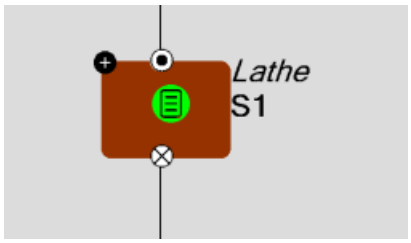
To tidy this up, it is suggested that you do this :





Name it lathe :

Result



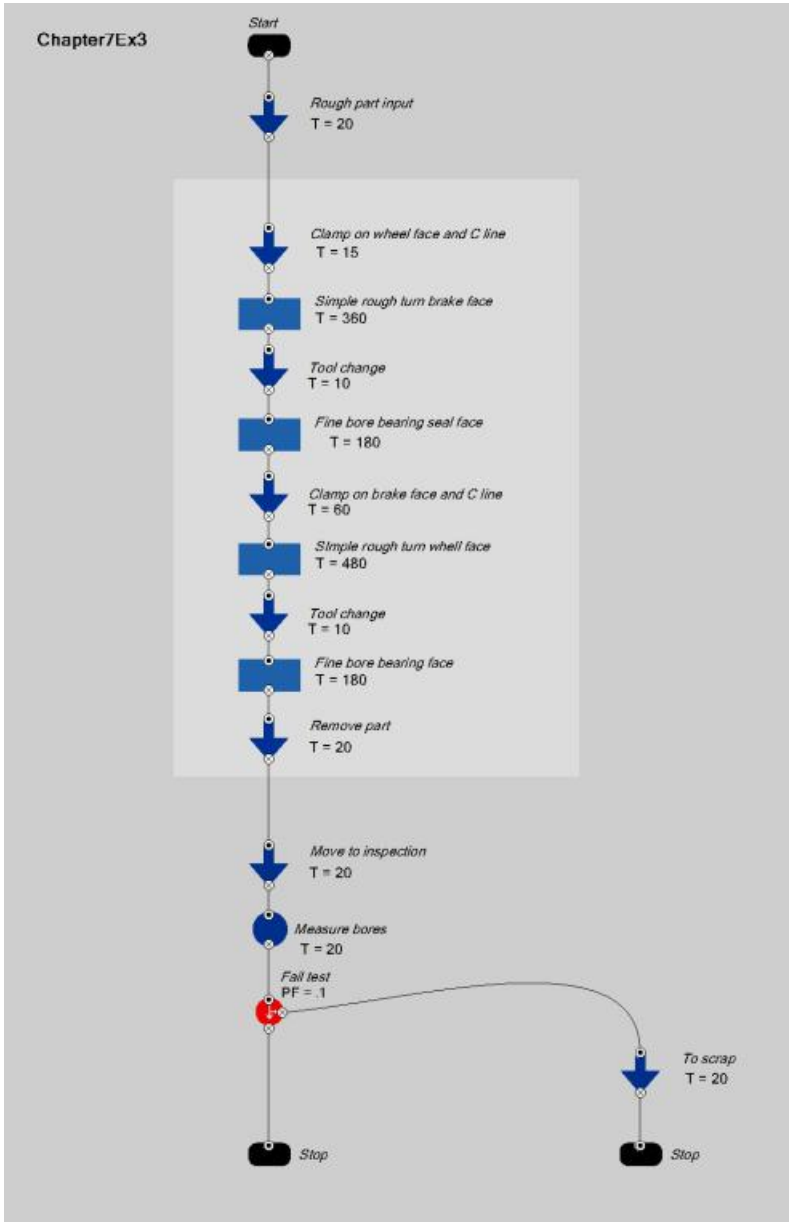
In Chapter XXXX we will discuss resourcing and compound tokens (a token piggy backing on another token) in much more detail. For now we have done enough to show how a machine can be treated as a resource to a process.

Gate based capacity constraints.

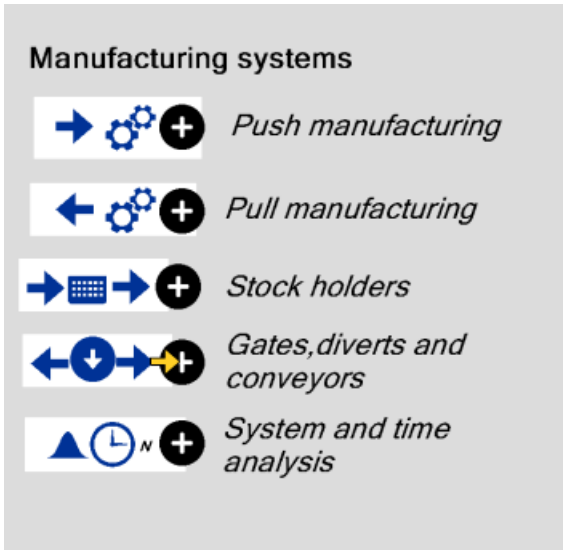
Start with the model *Chapter7Ex3*

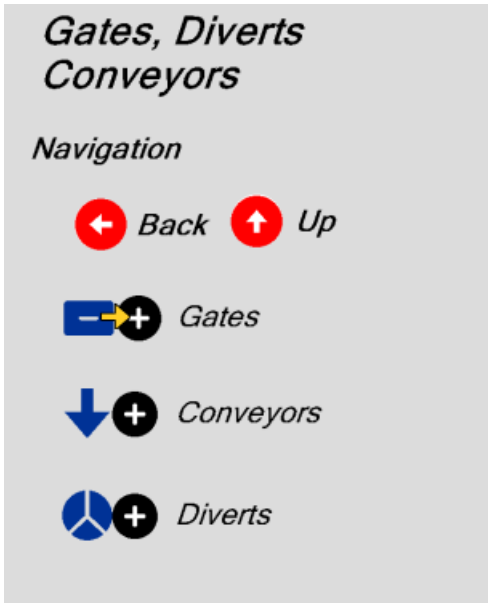
This is the lathe process we were looking at before with a simplified inspection procedure.

The jobs that are done in the lathe are highlighted below:

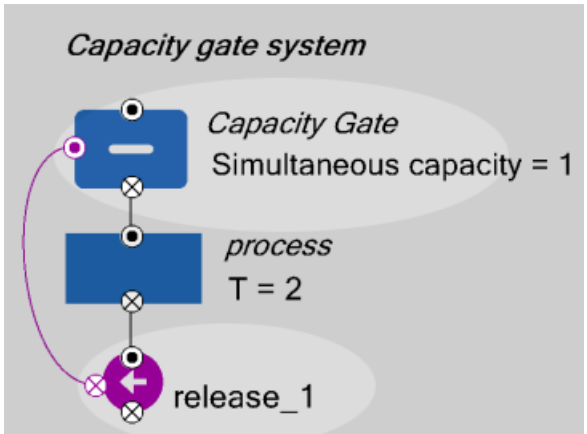


We will now put a capacity gate system around this process:





This is the capacity gate system:

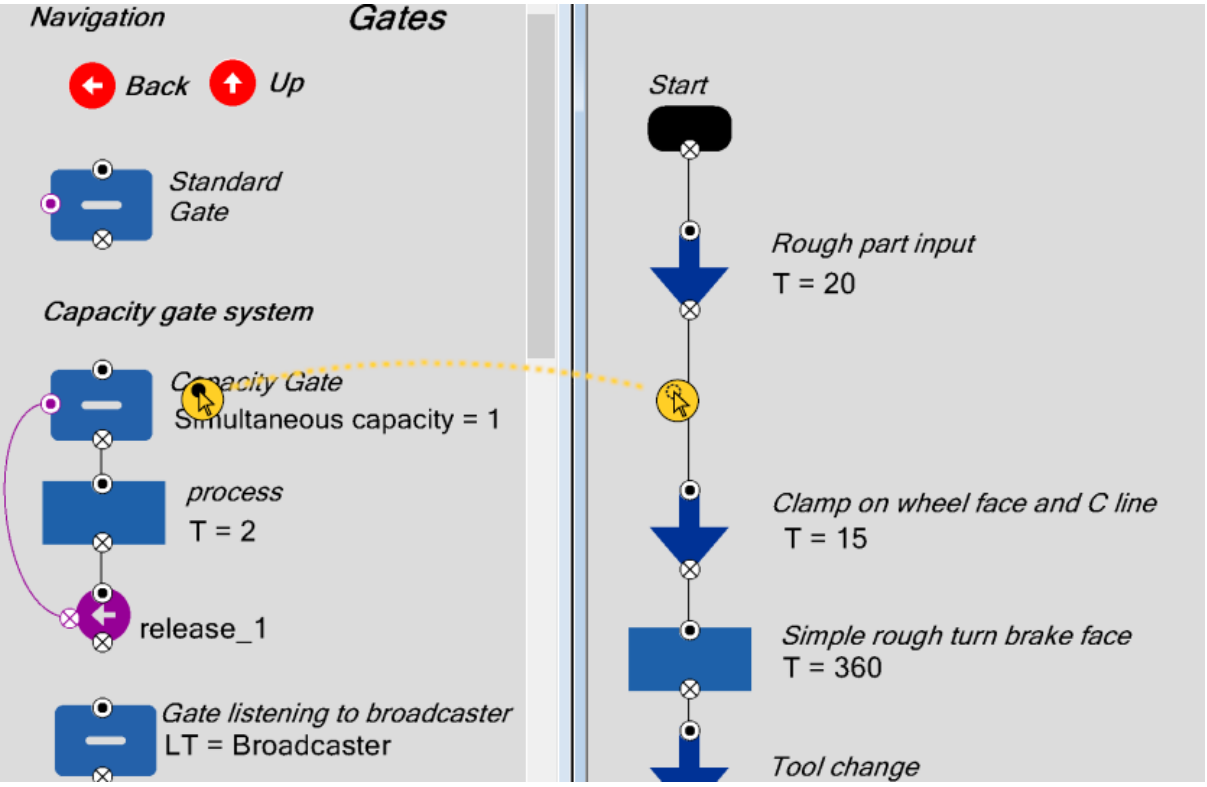


We surround the process with a capacity gate and a message sender. This capacity gate has a simultaneous capacity of 1 . This means it will initially let one token through and block any additional ones until it gets a message to release one.

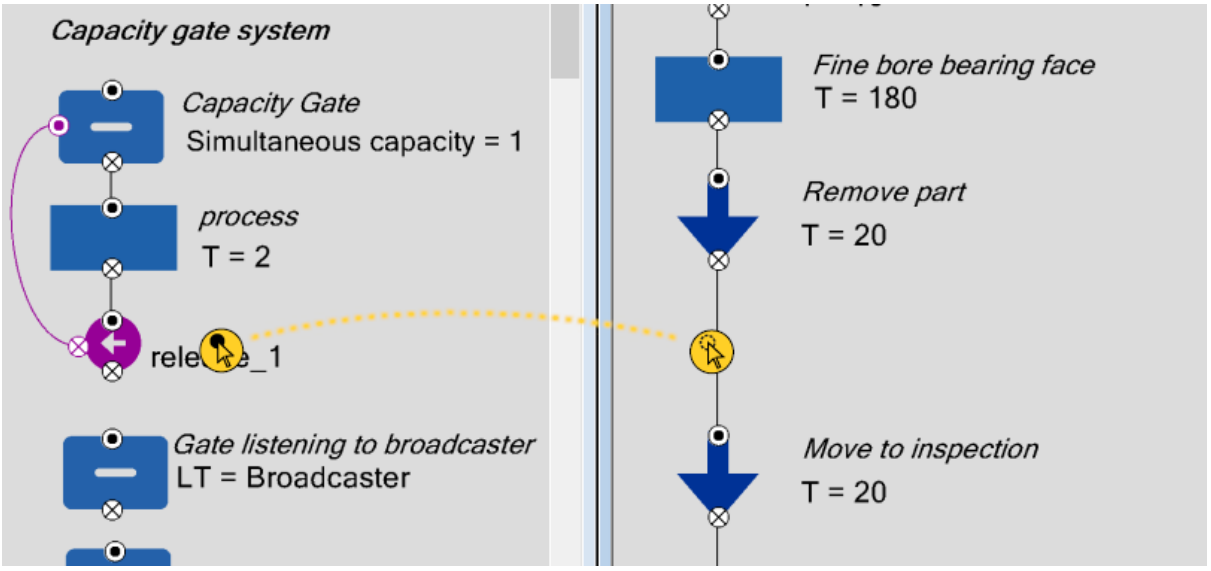
The message sender is set at the other end of the process. When a token gets here, it will open the capacity gate for one token.

As an aside, if the machine is underutilized, there may be no queue at the gate. If the block gets a message to release one in this case, it will count it as a request waiting to be fulfilled and immediately release the next arrival.

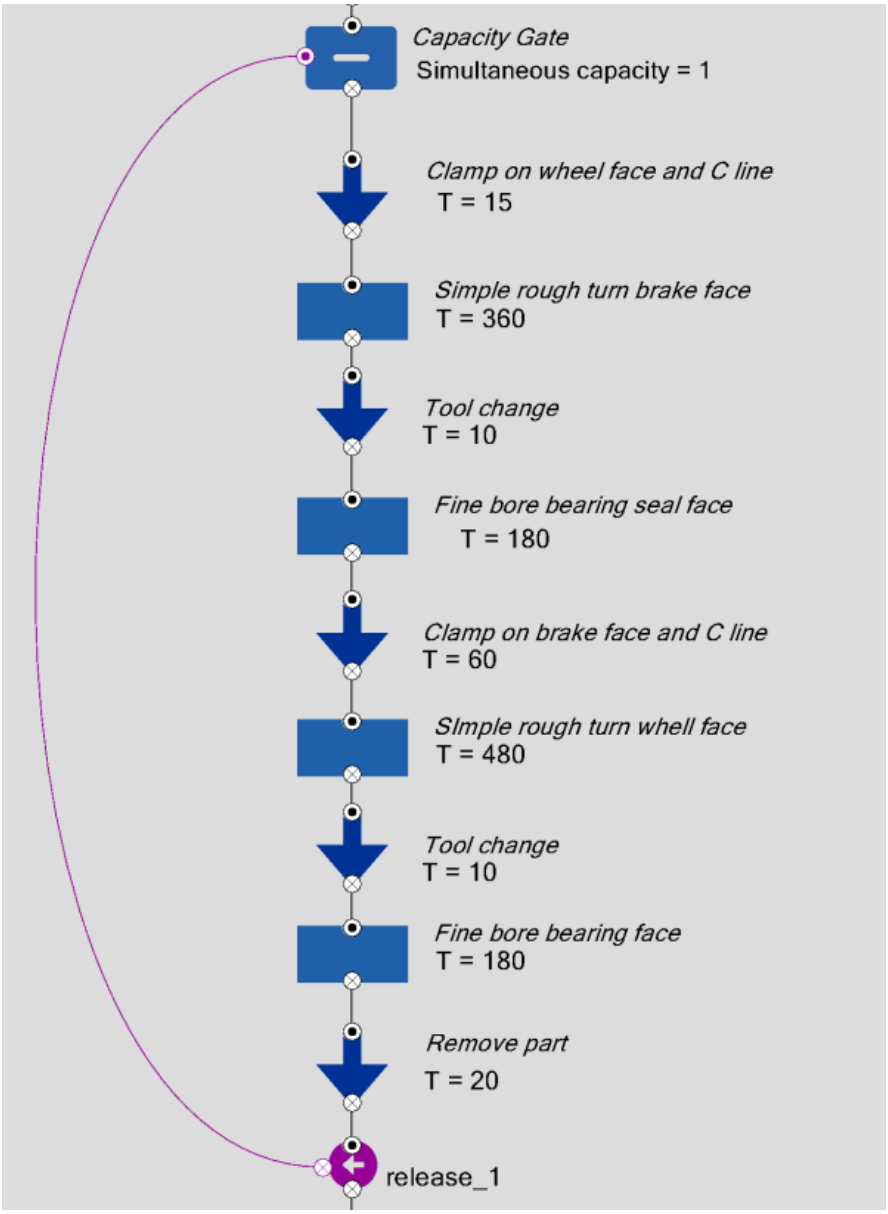
Drop your capacity gate in front of the first process element in the lathe



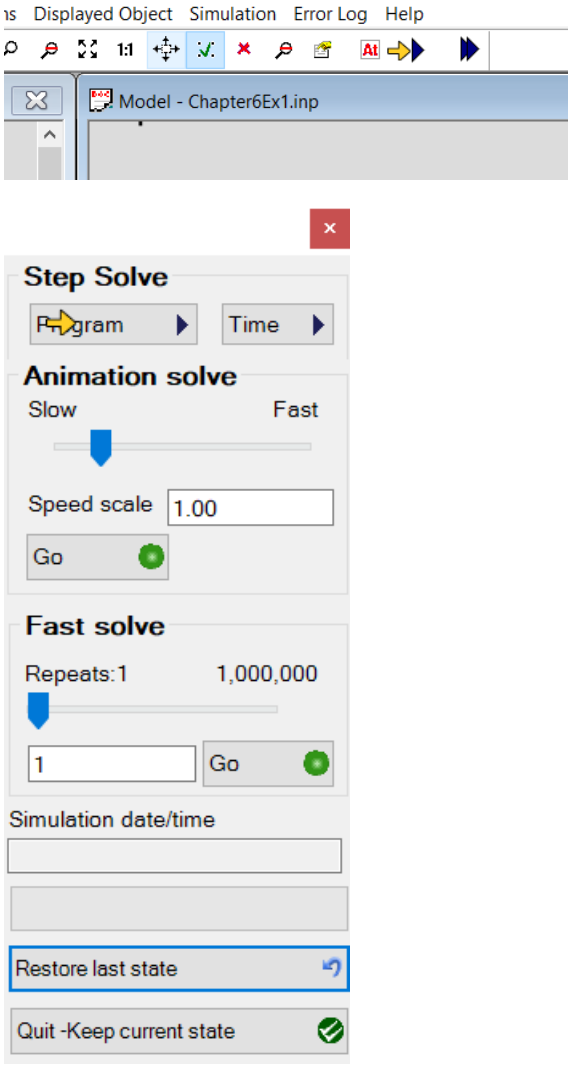
Drop the release message sender at the end of the lathe process:



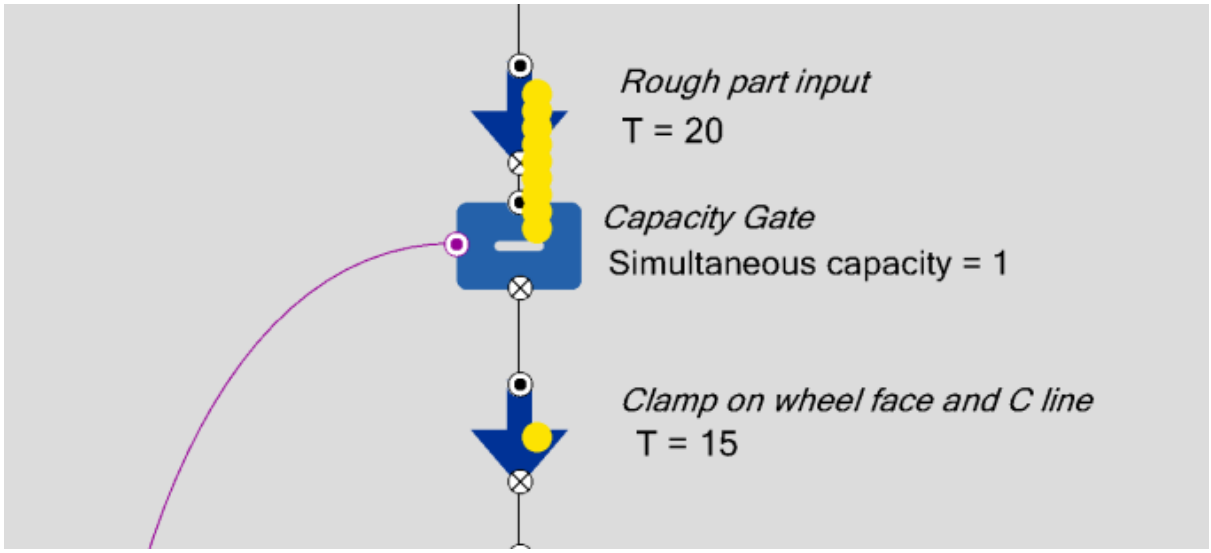
Connect the message sender to the capacity gate:



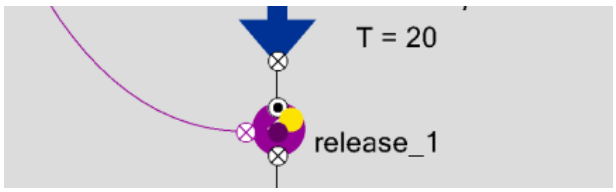
Step solve :



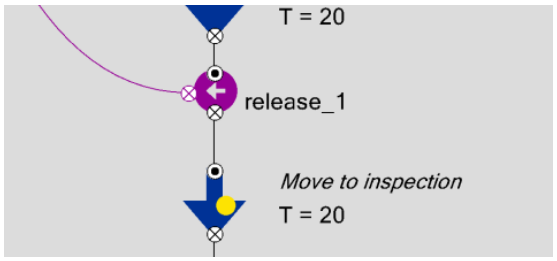
When the first token arrives, it is let through . The others have to wait :

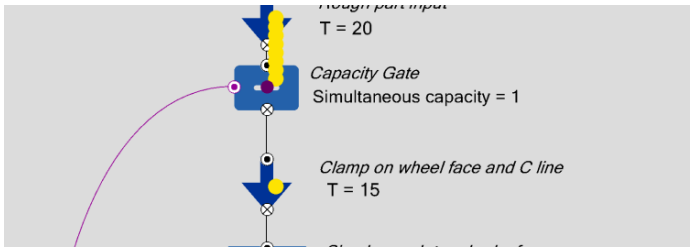


When the first token reaches the message , you may notice a small purple message token appear :



This is send to the gate, while the original yellow token carries on through the process:



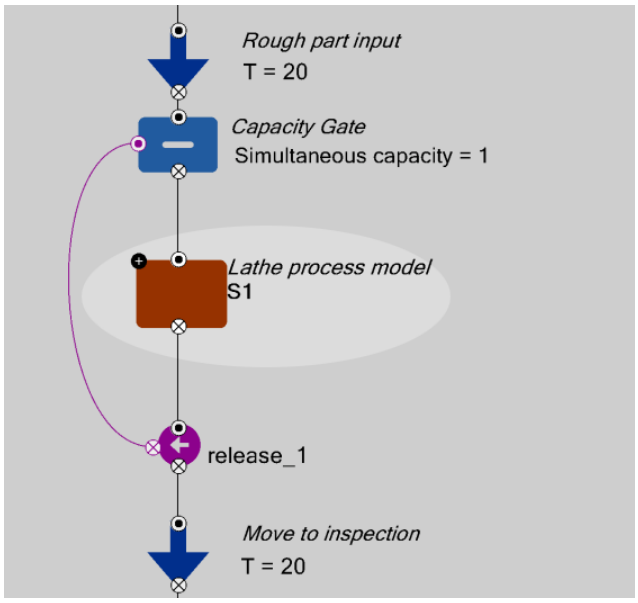


The gate receives and consumes the message. It lets the next token through.

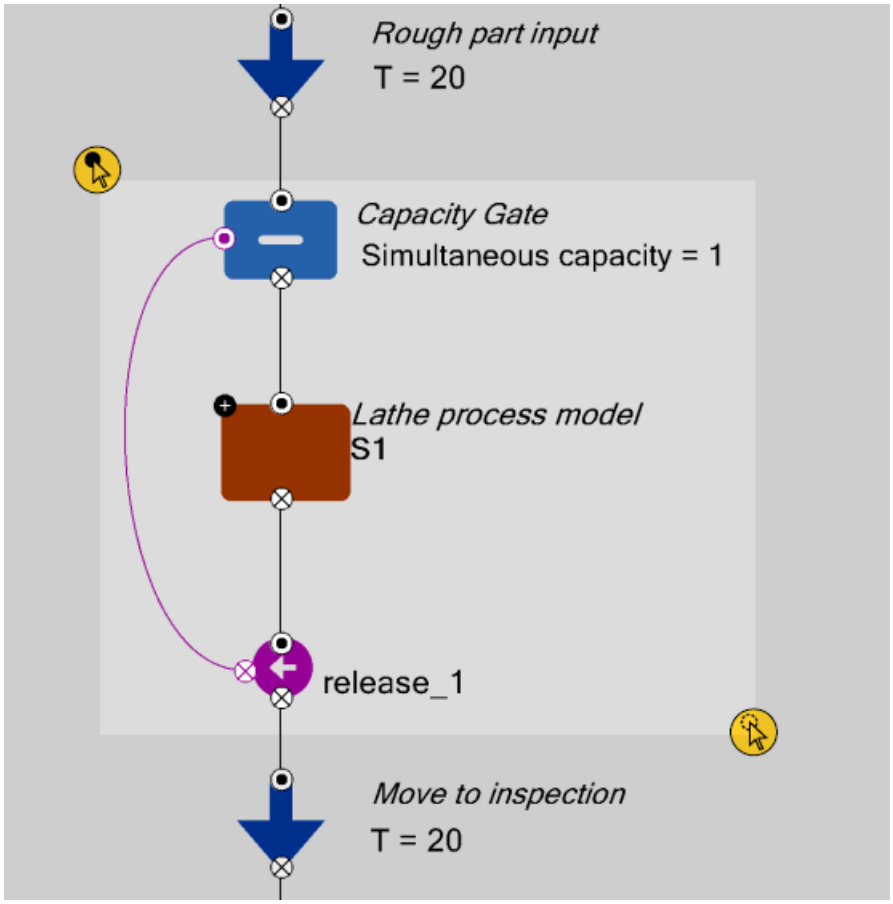
Eventually the solve will complete when all ten tokens have been processed in this way.

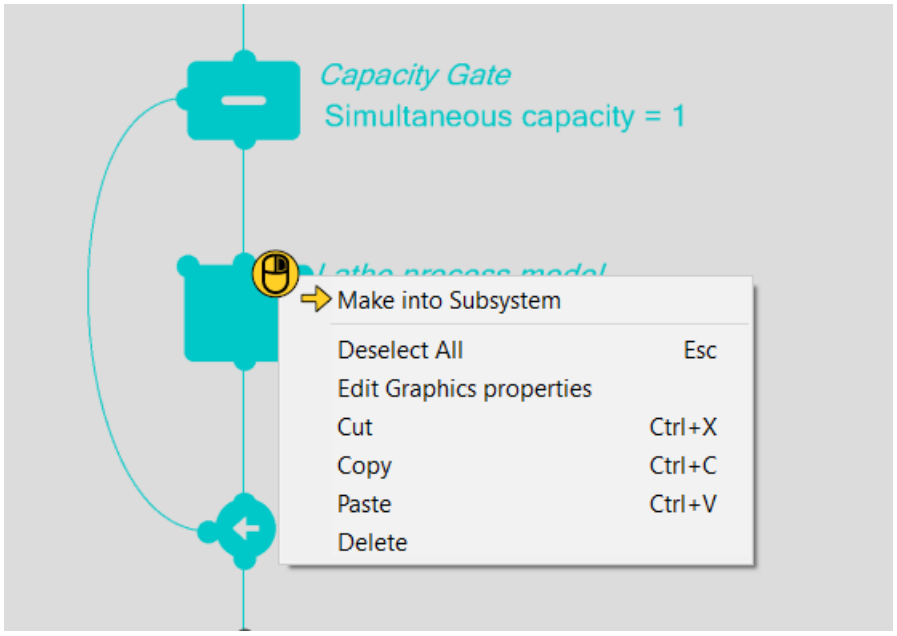
Lets continue with a bit of tidying up and putting capacity on the inspection process :

We can turn the main lathe process into a subsystem :

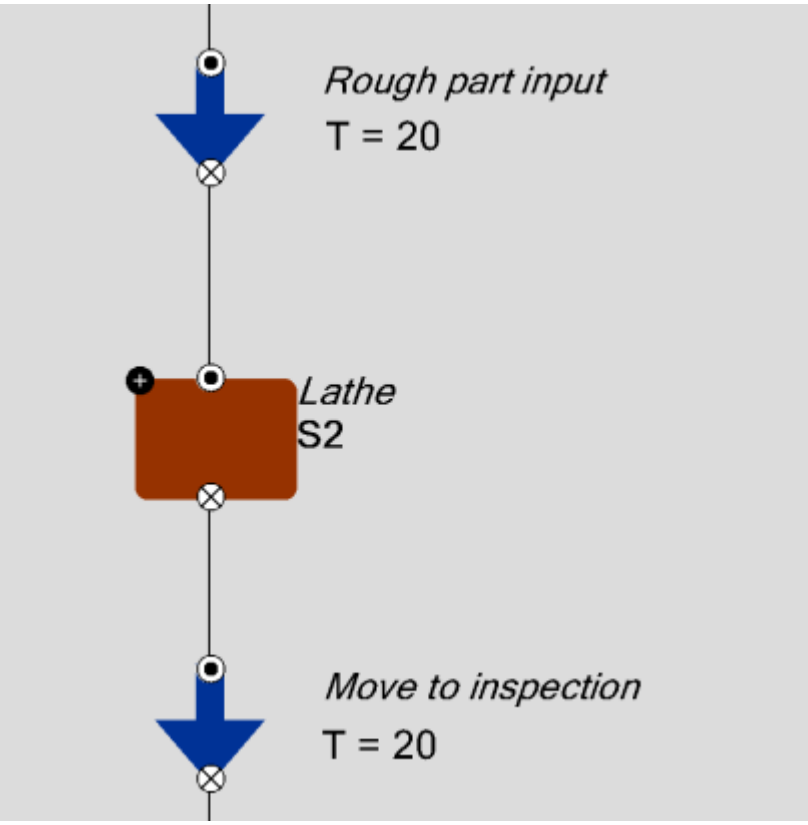


We can also out another subsystem layer around the capacity gate system :





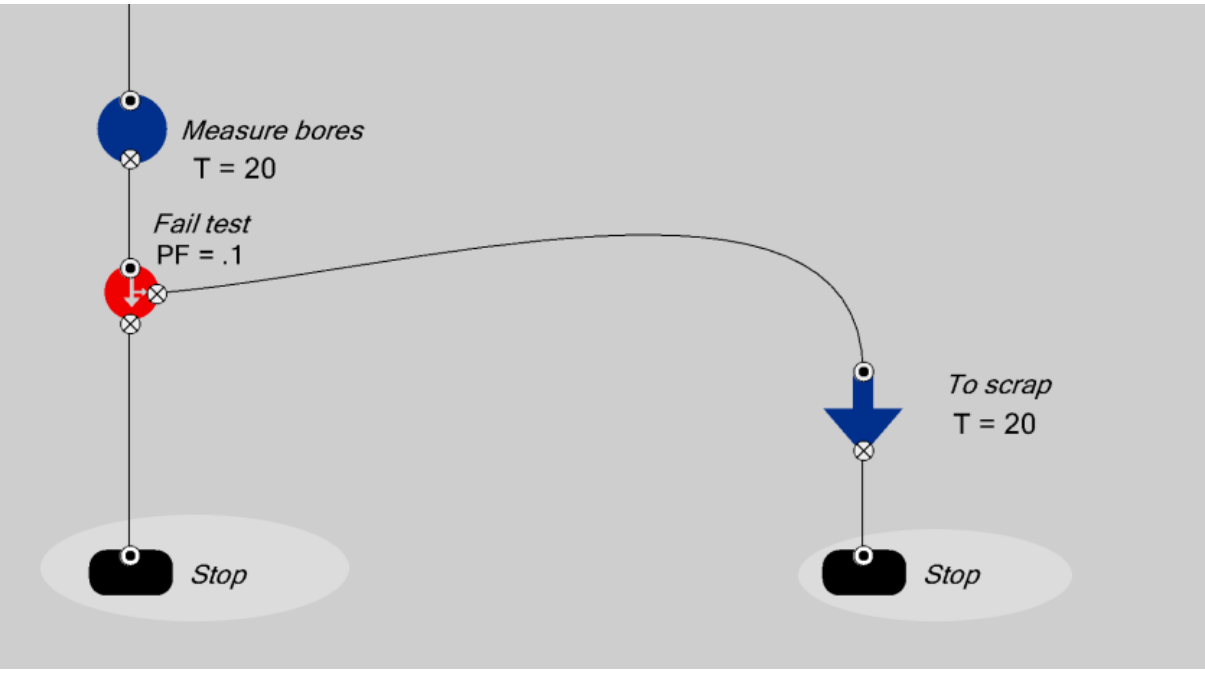
And change the description to get this :



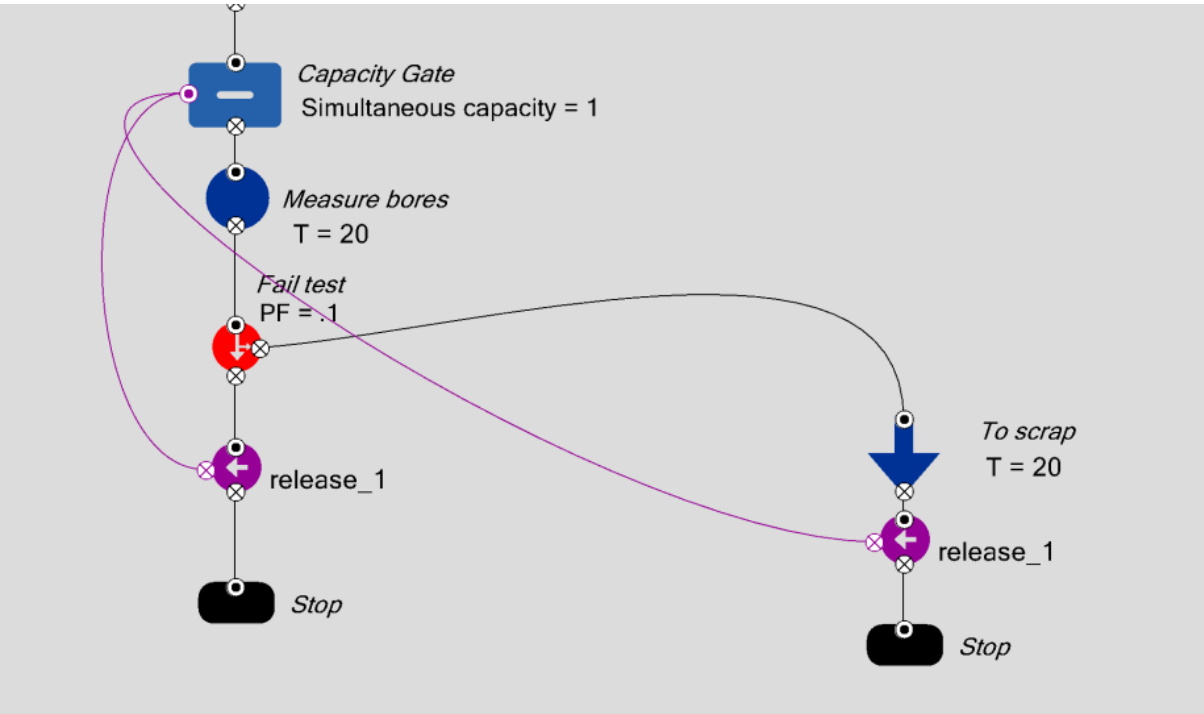
By all means use the animator or step solver to check that this works as you would expect

If you need a reference part for this stage, it is *Chapter7Ex3a.*

Lets now focus on the inspection process. The problem here is that we have two exit points, one for good parts and one for scrap:



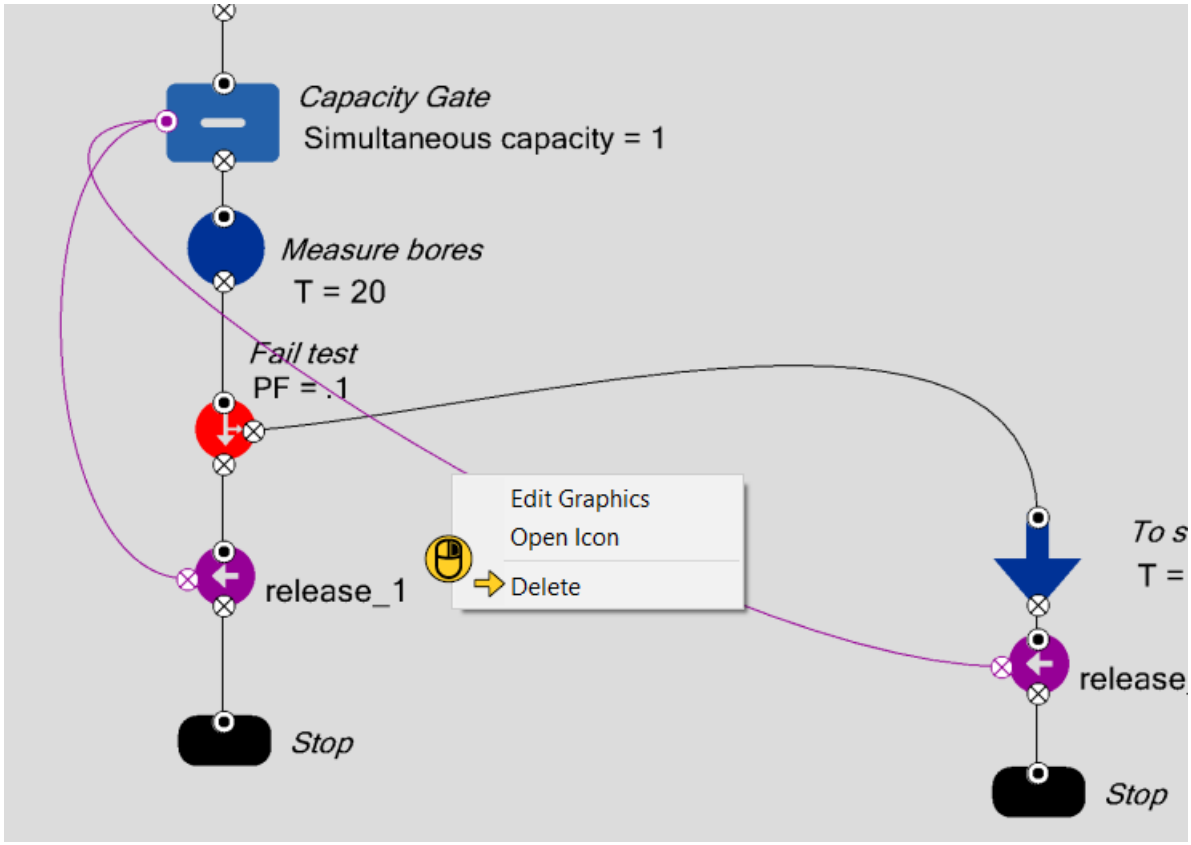
We need both pathways to deliver a release message to a gate. A working arrangement might look like this :



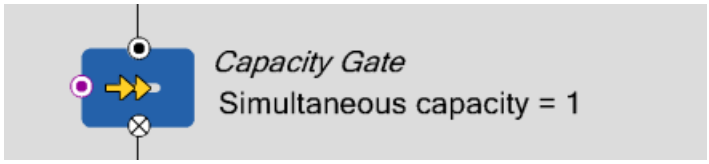
It works but it is a bit messy!

To fix the mess :

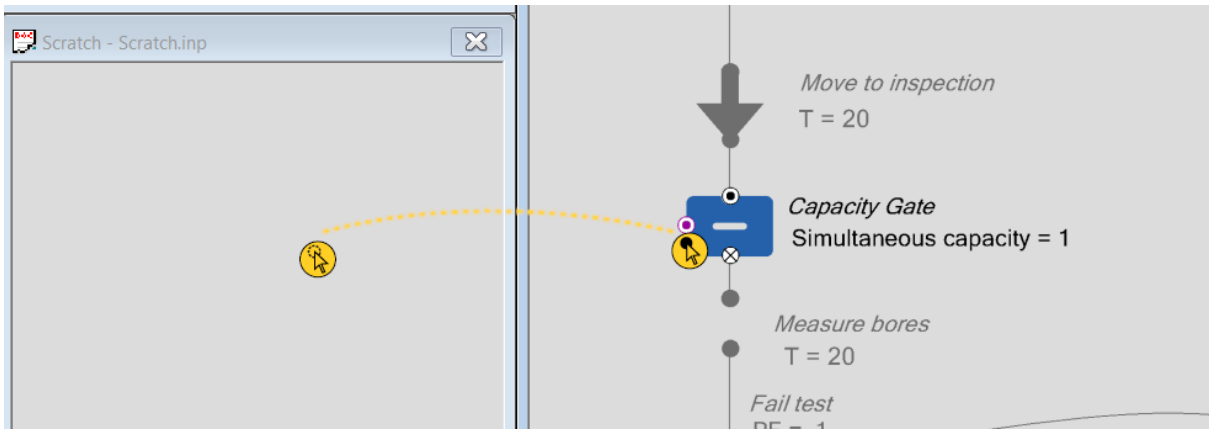
Delete both message lines :



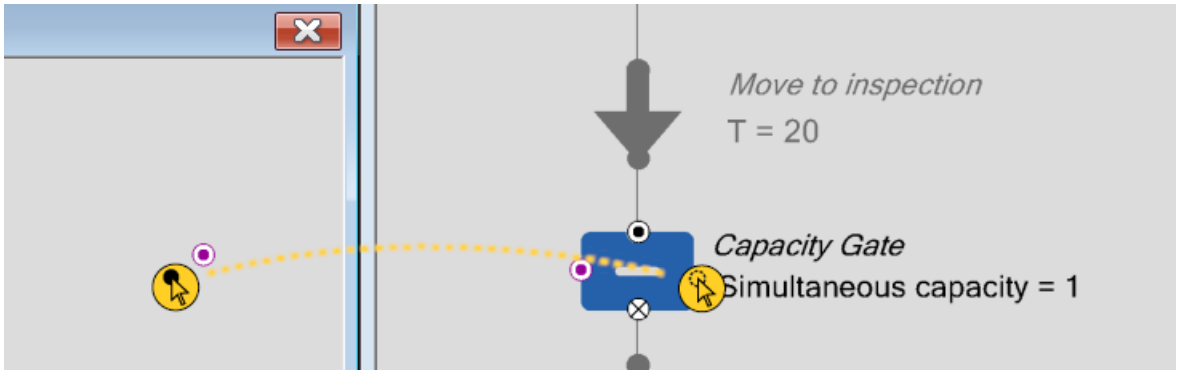
Double click the capacity gate:



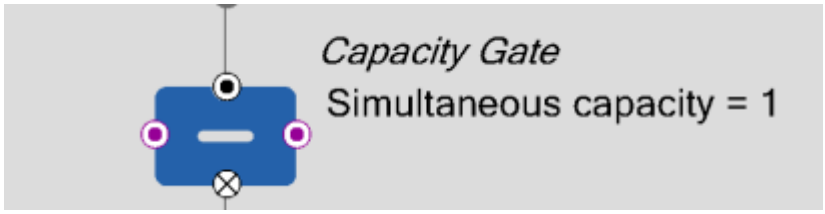
Drag the message port into your scratch area :



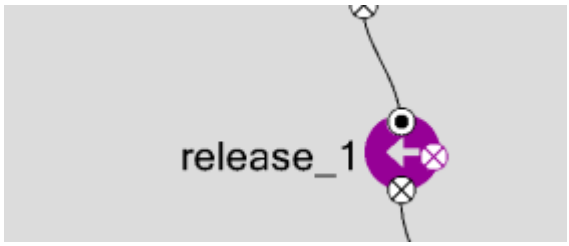
Drag a copy back into your block :



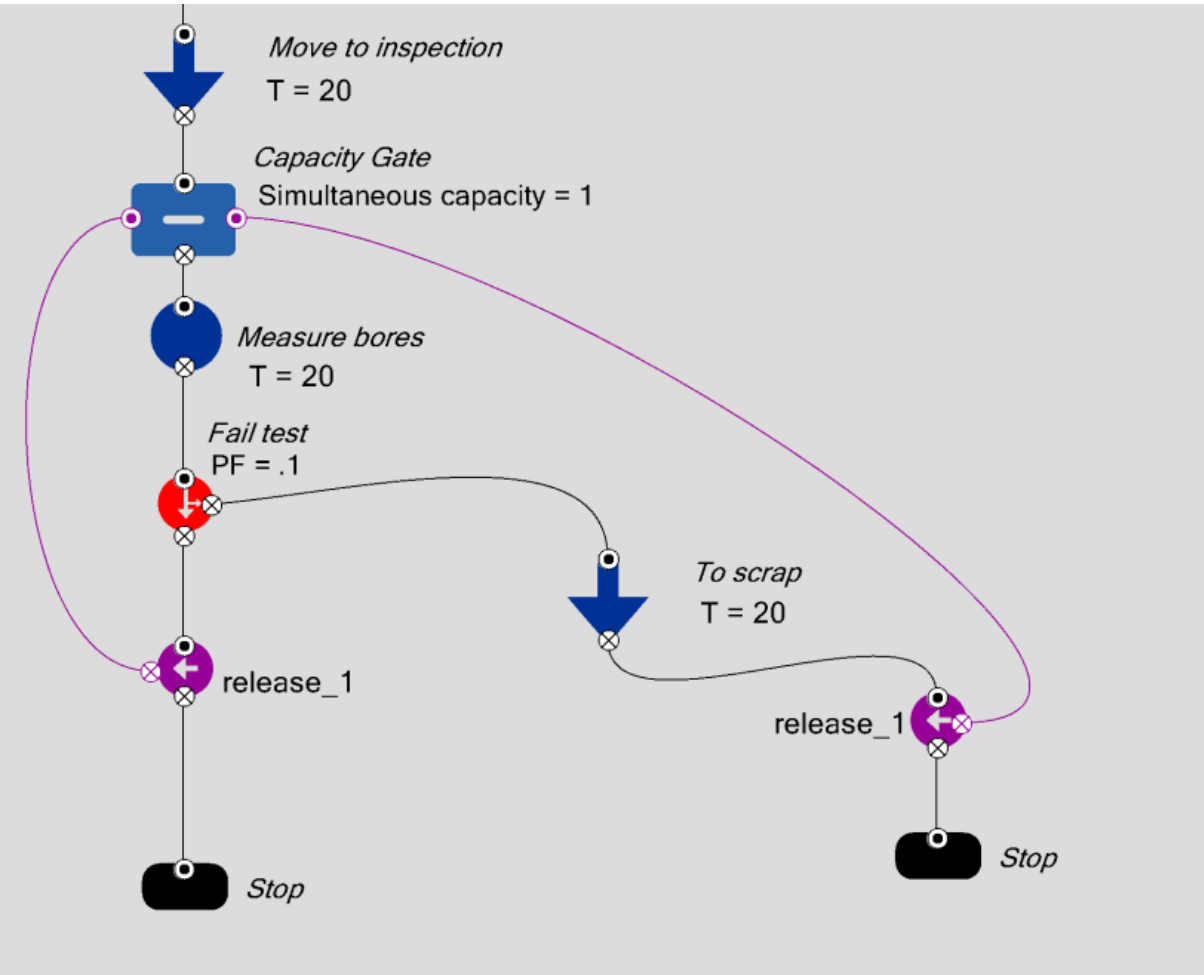
Tidy up the text position to end up with something like this :



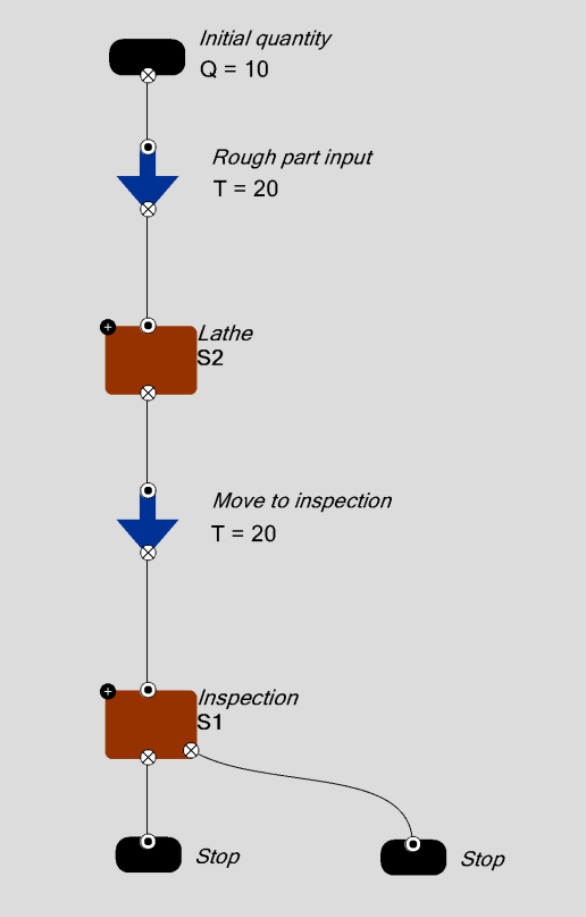
Likewise you can go into the message sender and simply move the port to the right position:



Now you can reconnect and move the blocks like this :



Of course you can finalise this to make a subsystem of the inspection facility.



Reference model : *Chapter7Ex3Done*.

The good thing about this approach is the fact that, like the resource based approach, the process is fully encapsulated within the model.

This method is probably easier for most to model than the resource based approach, and If you do a solve time test, you will find that it is a little faster.

We would recommend this approach for the pragmatist, and the resource based approach for the philosopher – or if you need to detail the resources used by process at much greater depth.

If you animate this, you will see that the lathe works flat out, while the inspection facility is under-utilised. This will bring us onto line balancing techniques which are outlined in chapter 8.

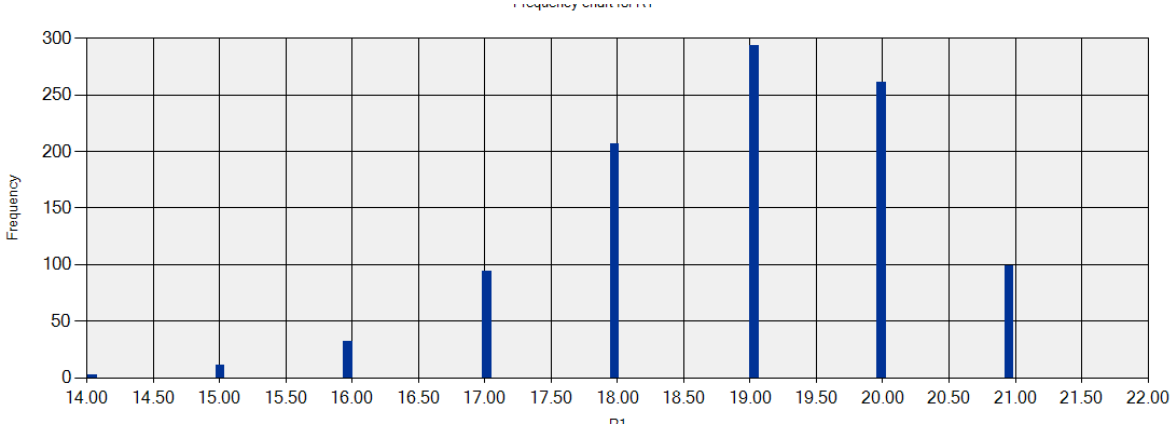
Exercise 4:

Using *Chapter7Ex3Done* as your start point. Estimate how many parts you could create with one lathe using this process.

Hints : Set the solve finish time to 8 hours in seconds and set the source to produce more than needed. Count the good output :

What is the range of possible daily outputs.

Answer : 14-21



Summary

This chapter brings in the concept of capacity constraints being forced onto an infinite capacity process model. Philosophically, all capacity constraints are caused by lack of resources and a resource based approach allows for this to be modelled explicitly. We have also looked at a gate-based approach which is a shortcut which still encapsulates possibly complex process models. Finally, we have looked at putting capacity on individual delay blocks when it is appropriate and when solve time is an issue.

Questions :

1. What are the advantages in managing the modelling process of a model that encapsulates the process verses a simplified, delay block based model.
2. If we want to know how many parts we can create in a shift, what is missing from our considerations so far.

Answers

1. One model contains both the process definition and the production capacities. If you decide to throw away the original process model, there is no need for

synchronisation. On the other hand, if a company operates the same process but with a different production capacities in multiple sites it would be advisable to keep the process model clear from these capacity models. In this case, a change management control needs to be instituted. If the process model changes, the down stream capacity models need to be revisited. Engineering data management is a tricky subject!

2. The main one is stoppages – breakdowns, breaks in demand and resource shortages (eg rough parts or people). We will look at modelling breakdowns in chapter XXXX. Also, this particular model does not really consider rework – although we took a look at this in chapters and 5.