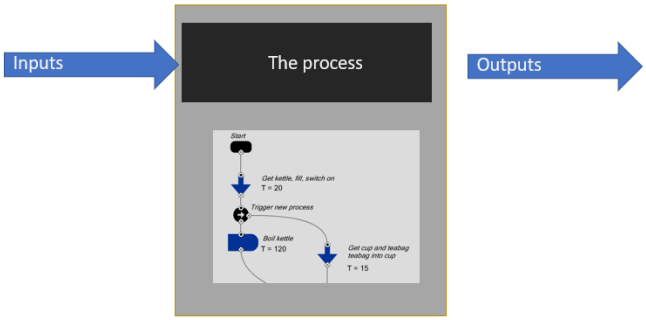
Chapter 6 – Encapsulation – making black boxes

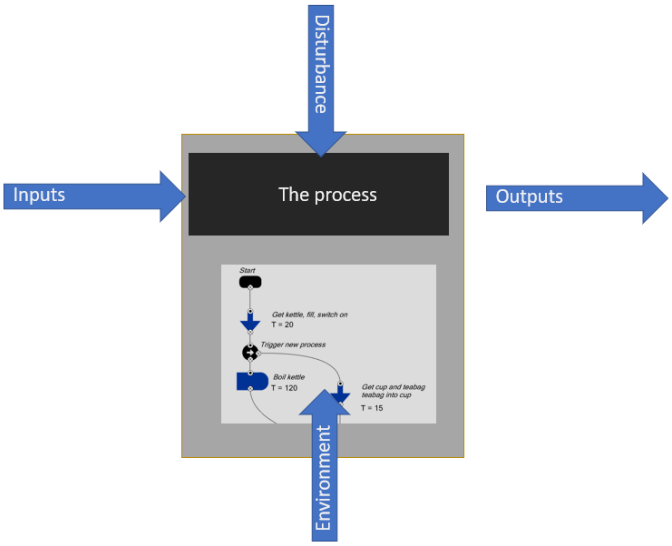
We already asked “How would you define a good manufacturing process? “ – and at the internal level we have partially answered this by thinking about waste (Chapter 4) or notional costs (Chapter 5). In this chapter we will look at the process as a black box – a black box is something where you can only observe it from the outside and you treat what is going on internally as irrelevant – So we will examine its performance from outside.

Well lets look at a common way of looking at any process:

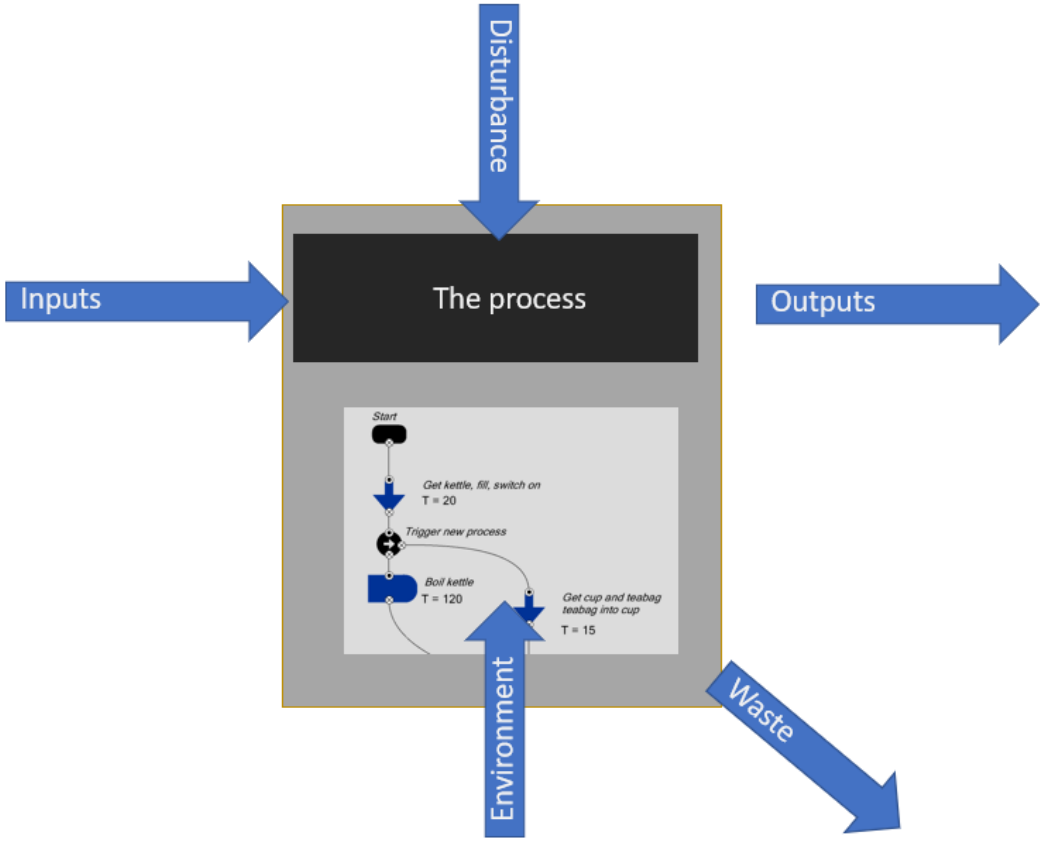


The process is seen as a black box with inputs and outputs. There may be more than one input and more than one output taking the form of physical goods, virtual artifacts or data.

In addition most processes are subject to disturbances or dependent on environmental conditions . They peturb the way a process is running. These are often shown separately from the inputs:



Finally – for manufacturing systems we may want to divide the output into useful outputs and “waste outputs” – this can take the form of scrap parts or waste products – such as the swarf that needs to be manged in a metal machining line – or the sawdust in a woodworking process.



If we look the system at this level – and focus on, for example, a mass production manufacturing system we can straight away see some of the answers to our question

1) We need manufacturing systems which produce minimal waste – especially scrap
2) We need both the inputs and outputs to tick like a clock – close to constant flowrates at both ends are ideal.
3) We do not want a manufacturing system where the physical inputs exceed the physical outputs over time. This will result in a build up of stock – with negative economic results. The systems need the capacity to mee their demand while being as economic as possible
4) We want systems to be as stable as possible in spite of disturbances and environmental changes

A key metric for manufacturing systems is : “ First time yield” - the ratio of the number of good units out divided by the number of units in. Scrap rate is the corollary - the ratio of scrapped parts to input part..

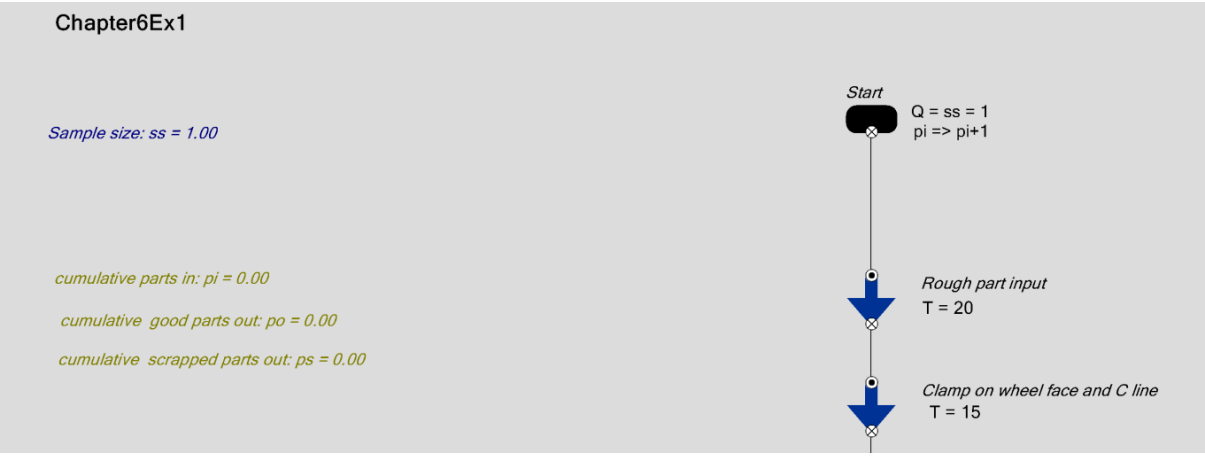
In this lesson we will start with a simple example showing how the process we created previously can be encapsulated into a black box which produces good and scrap parts. We then edit and examine the box in various ways including simplification and paramaterisation

and calculating first time yield. Finally then see how the box can be given an operational production capacity.

This chapter represents the cross over between modelling a process and modelling a production facility.

Creating a black box:

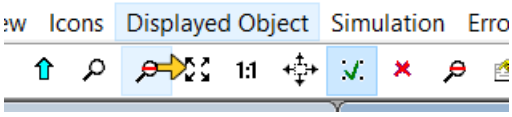
Open up the exercise Chapter6Ex1.inp



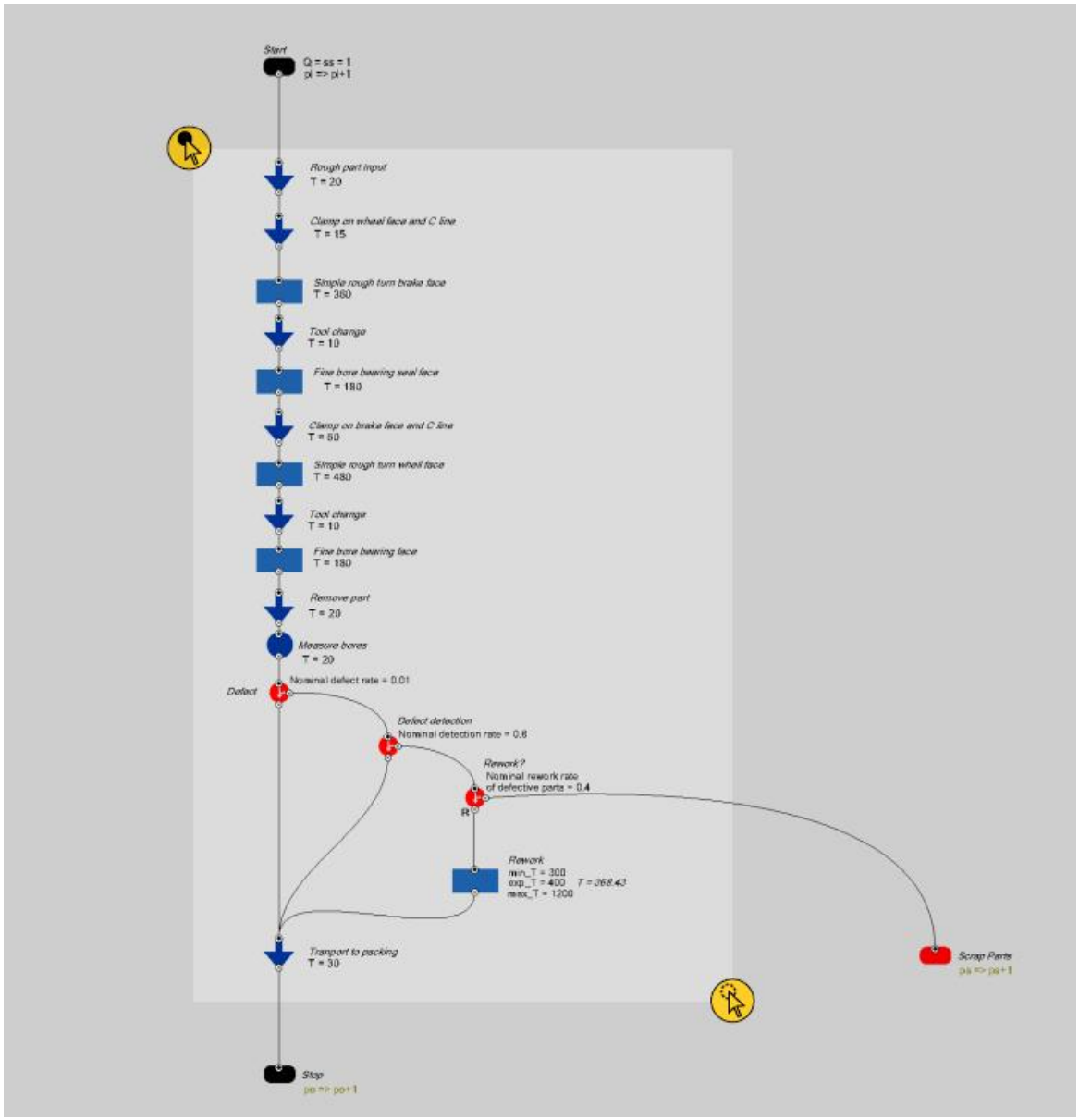
This is a stripped down version of the full waste analysis we did in chapter 4.

In aliquis, making something into a black box is done by turning a set of blocks into a *subsystem.*

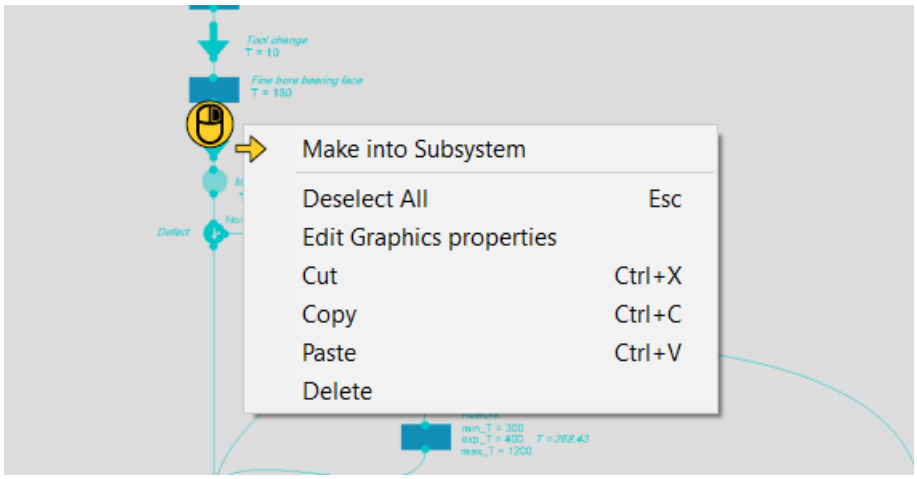
Zoom out to show the entire model :



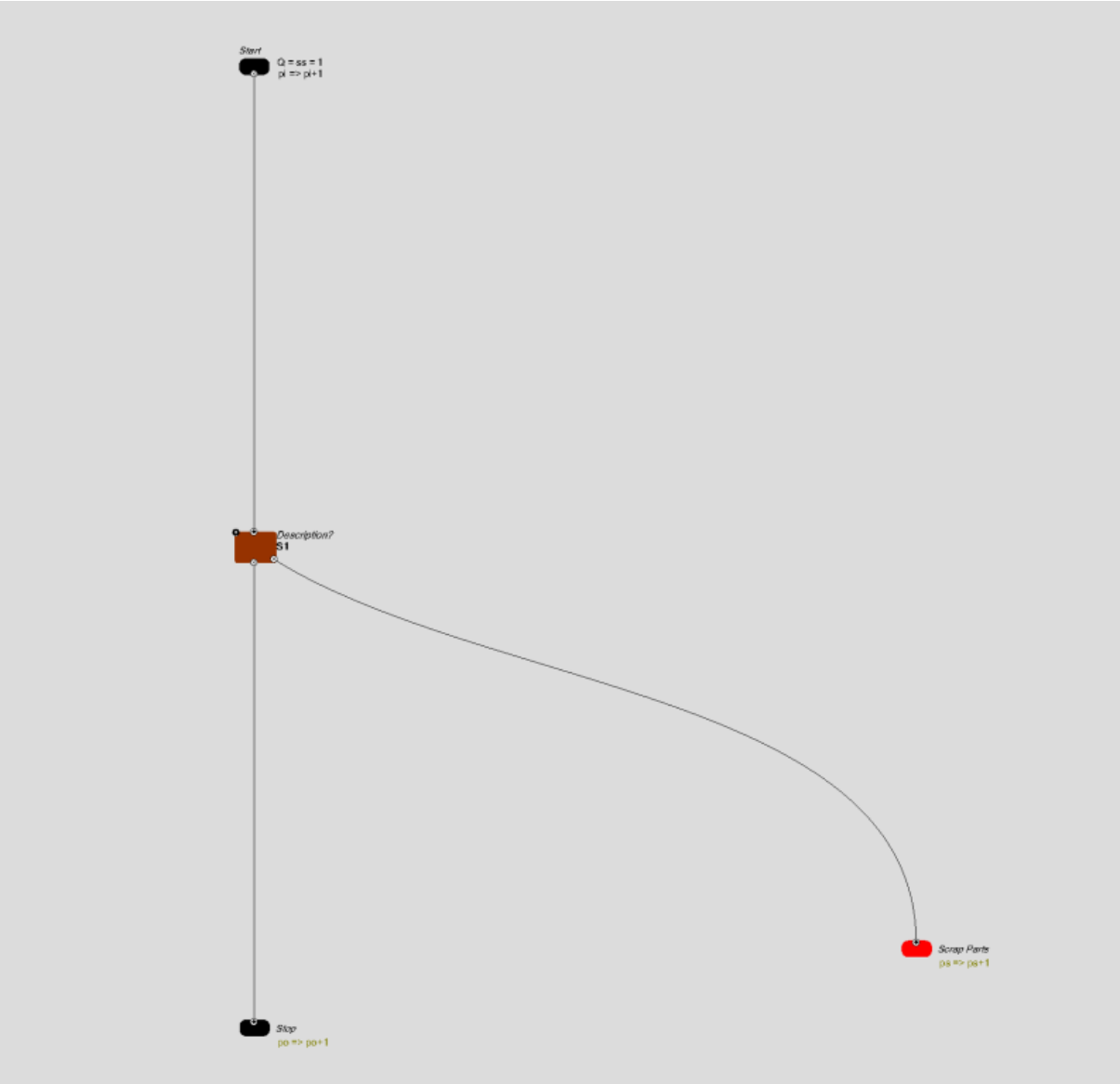
Select all the blocks except the source and sinks :



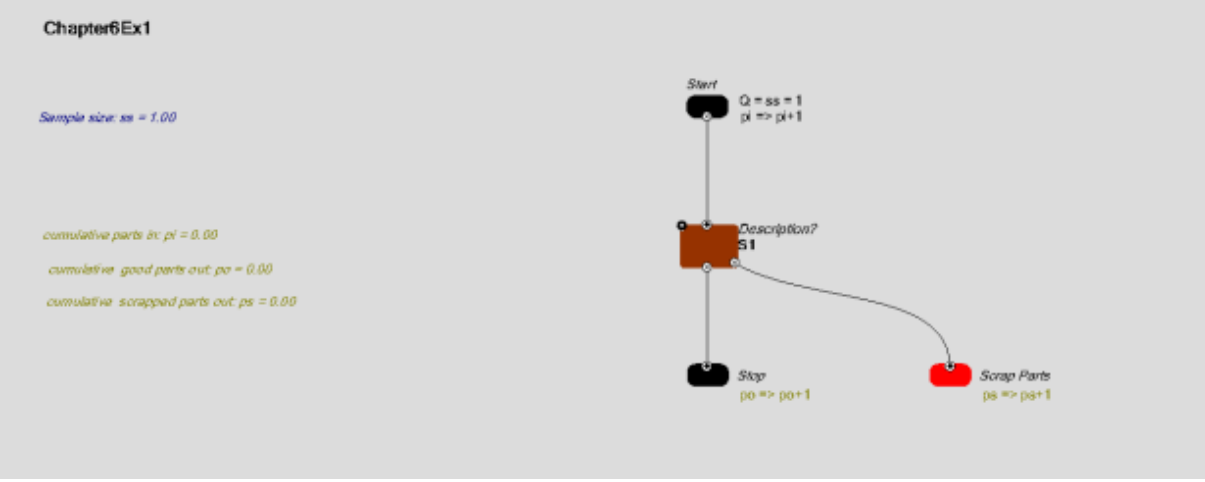
Now use the right button and Make subsystem :



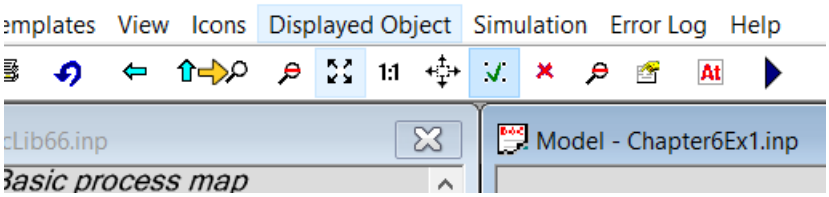
Result :

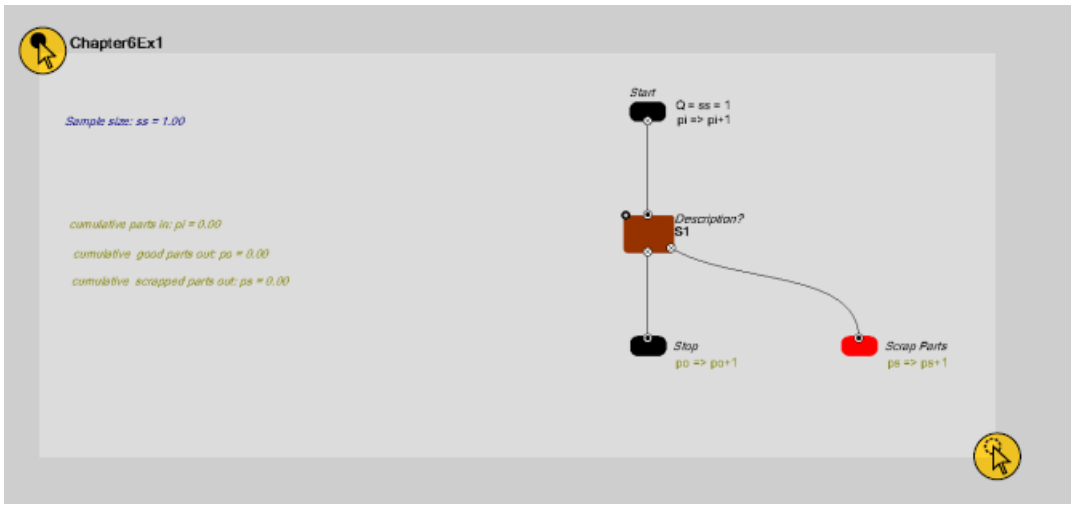


Move the blocks to make the model more compact

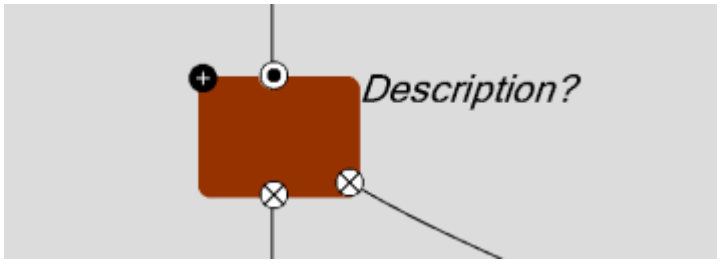


and zoom in :



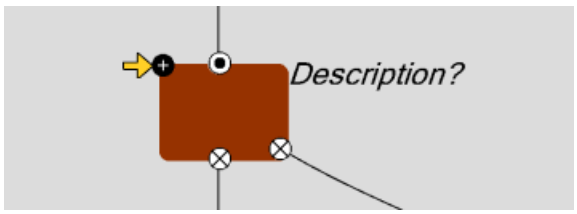


Although its brown ! – this is your black box

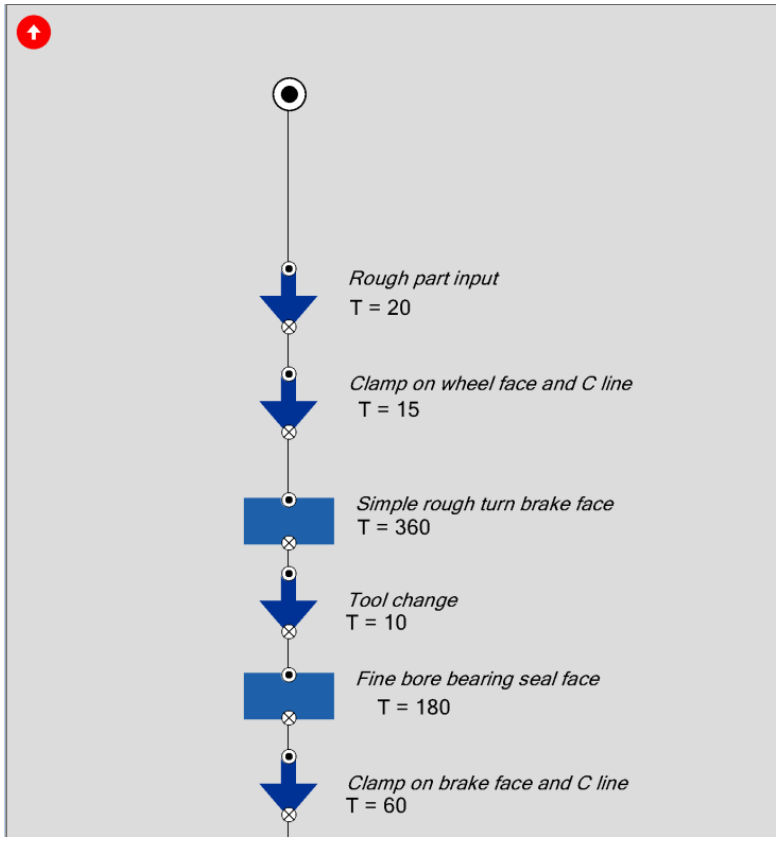


On the outside it is not dissimilar to the other blocks.

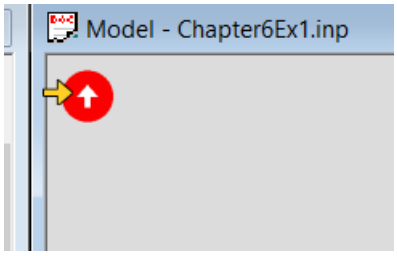
You can go into it by hitting the enter button :



Result :

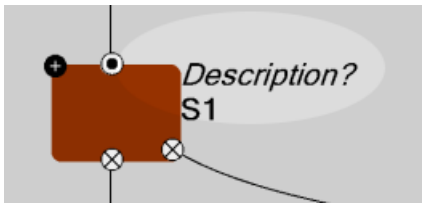


To Get out either hit the exit button or double click in space

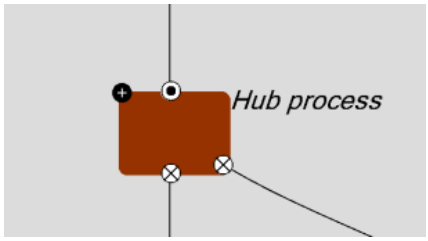


We now need to clean up the exterior of the subsystem:

This the unique name the new subsystem:

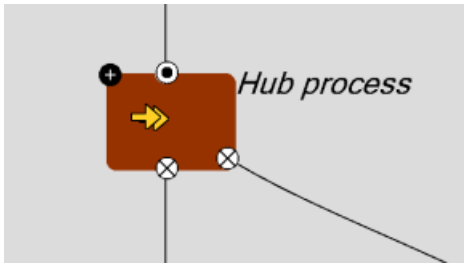


Change the description to something suitable :

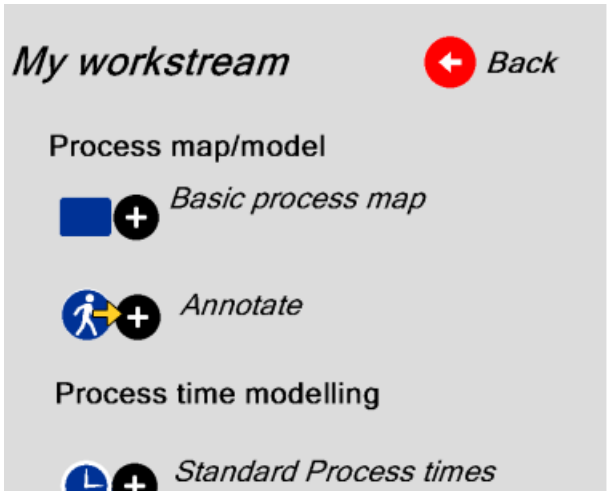


If you want to, you can go into the graphics and make alterations :

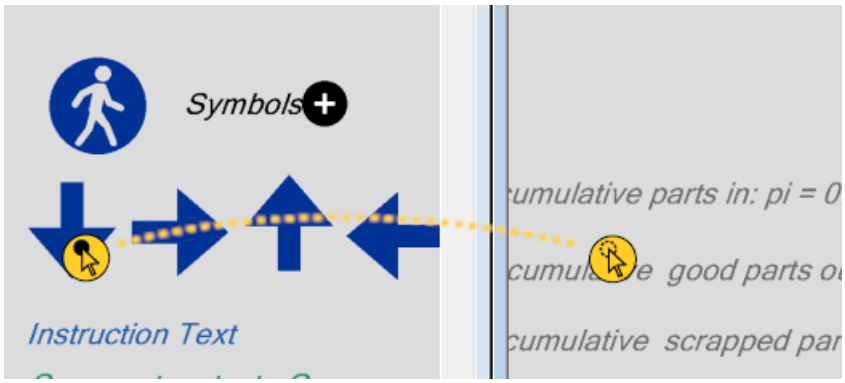
Double click the block :



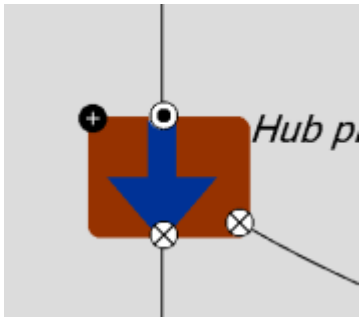
Eg to add a new symbol – go into the Annotate sub-library :



For example, we might want to show the main direction of flow like this :



Result :

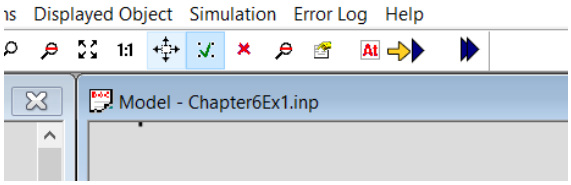


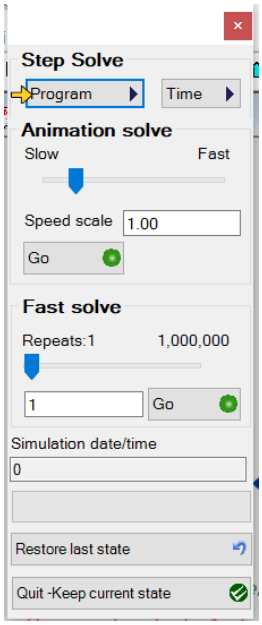
On your own you can change sizes and colours to get whatever best communicates the function of the black box.

Solving with a subsystem

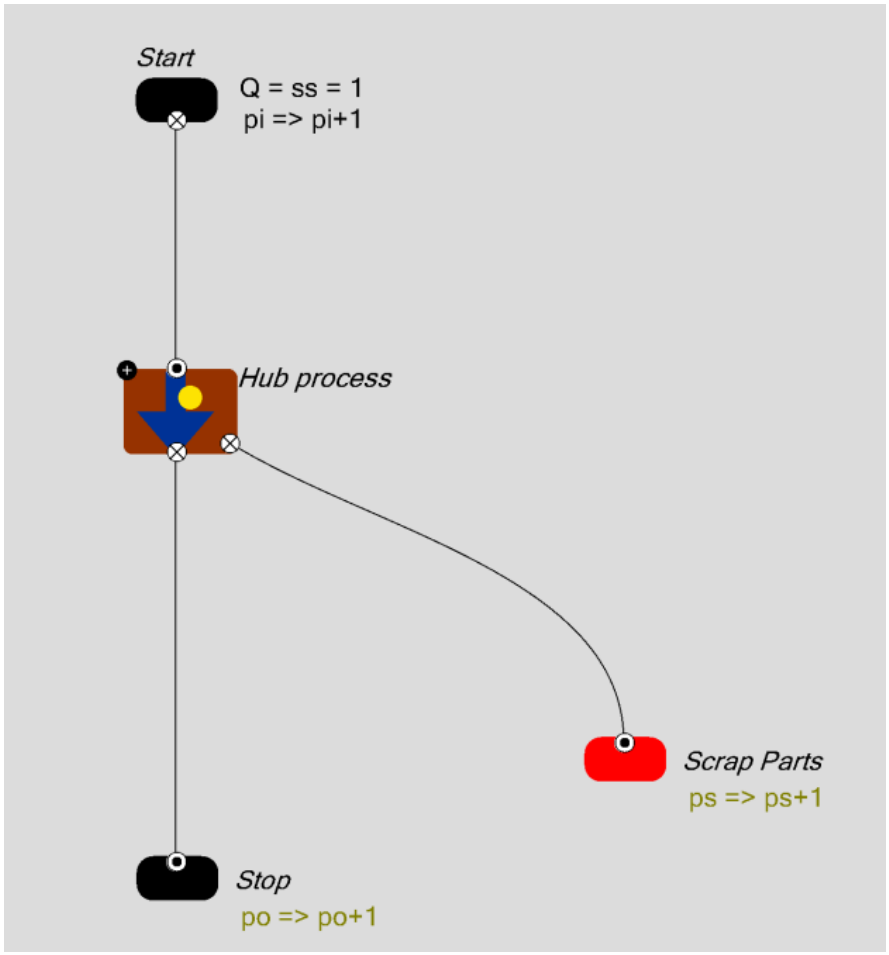
Once you have a subsystem you have the choice understand either as another block, or as a subsystem that you can go into during the solve.

Bring up the solver and step solve through :

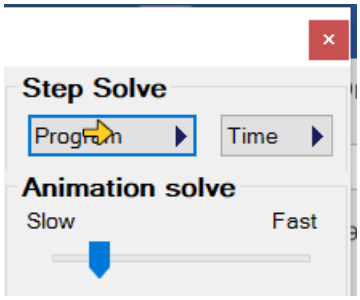




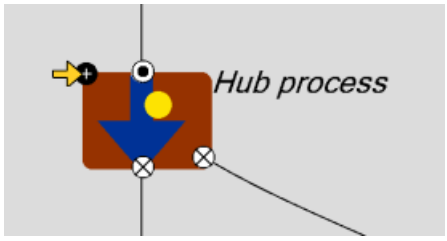
At this point you have the choice :



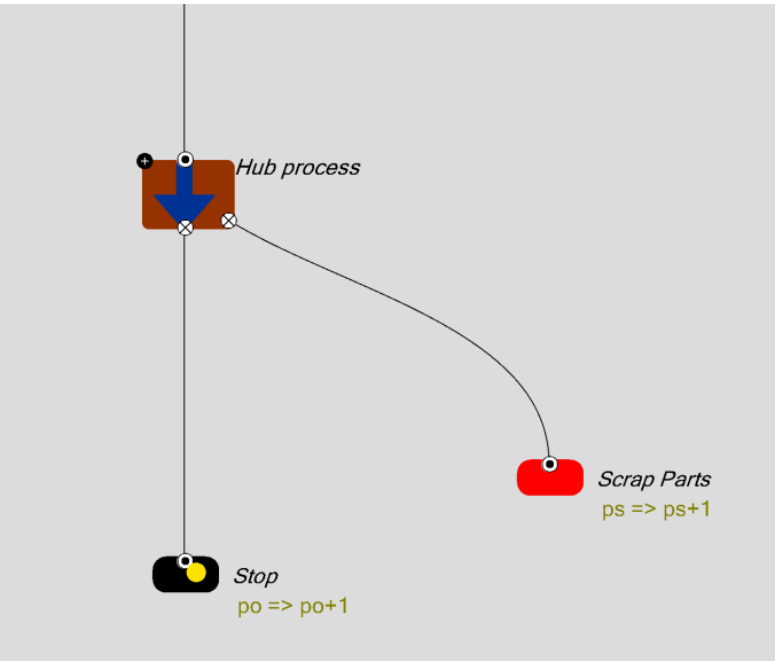
Either continue stepping past:



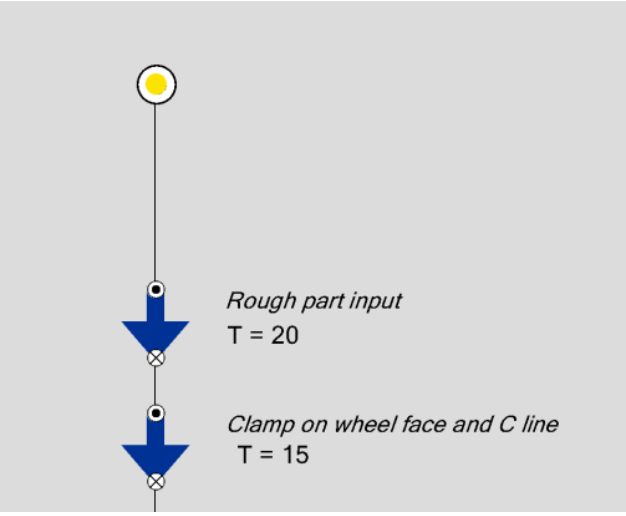
Or step in :



If you step over you will probably see this :



If you step in you will see this :



You may continue to program step . When you want to go back up – just double click in space. Or click the up arrow : 

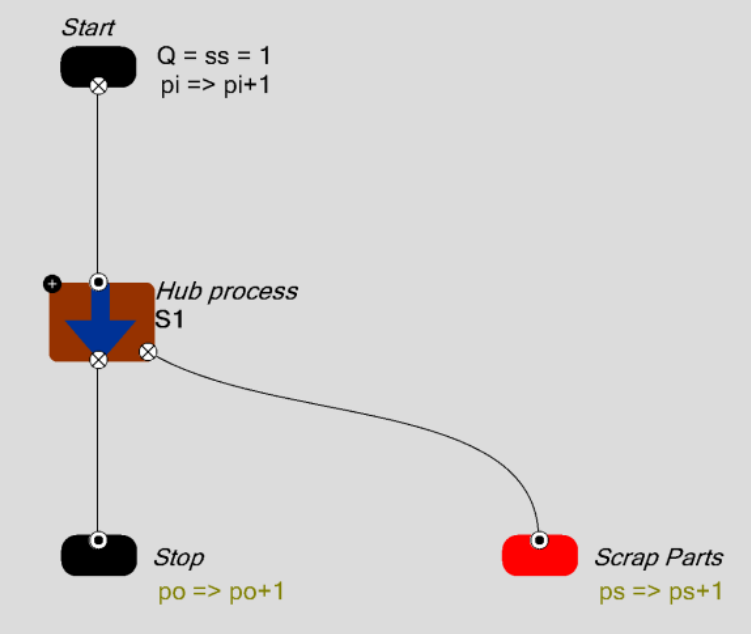
Once a subsystem has been established you will tend to just step over. It becomes the black box.

Simplification

In Aliquis, all blocks consume solving resources as they create events for the solver to manage. Reducing the number of blocks reduces the computing load. For smaller systems like this, it probably does not matter – but if we are trying to modeler more complex

processes, and if we are doing many optimisation and parameter sensitivity analyses, Solve time becomes important.

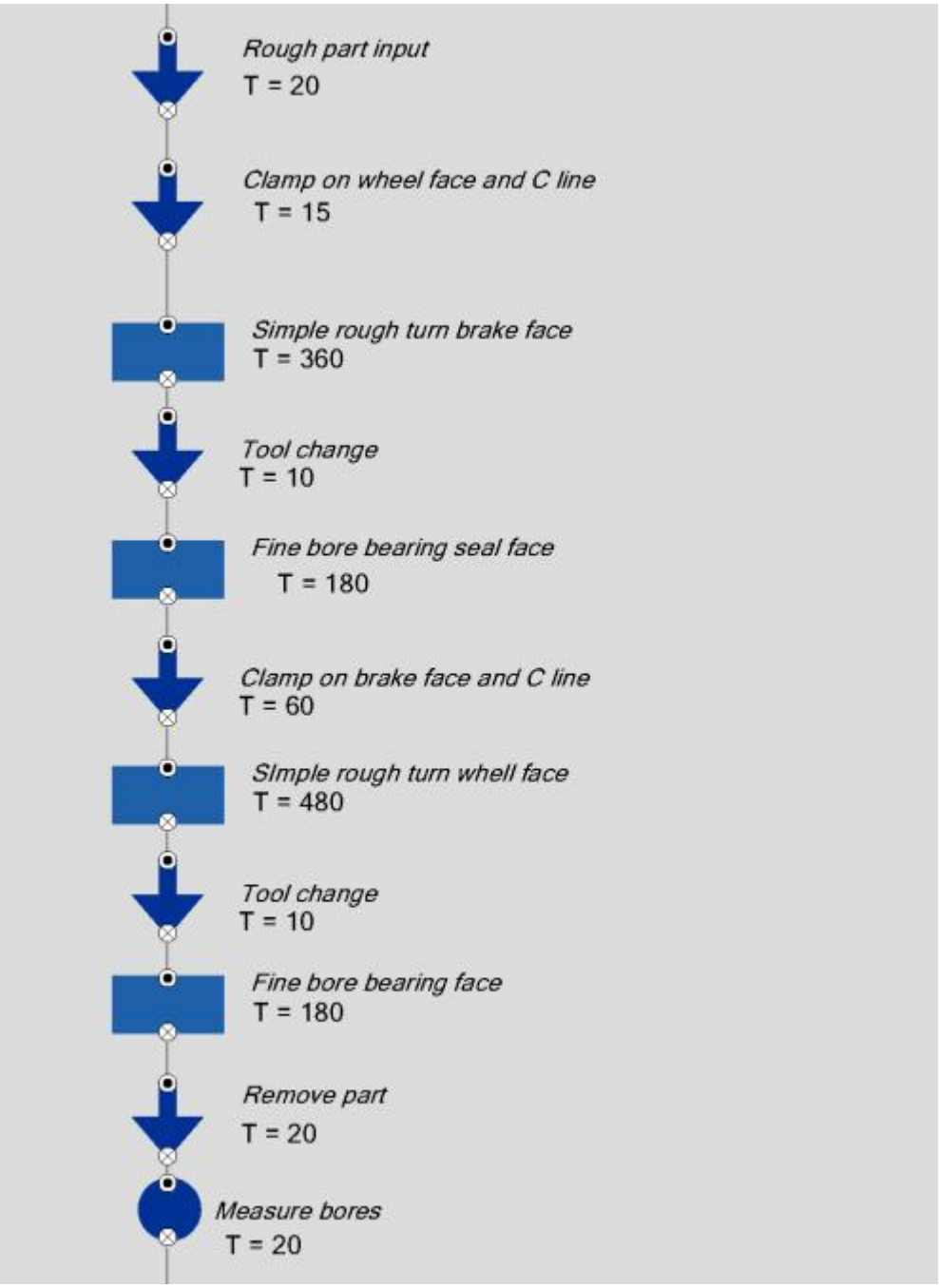
We should now be at this point with our model :



If you need a shortcut you can find it here : *Chapter6Ex1a.inp*

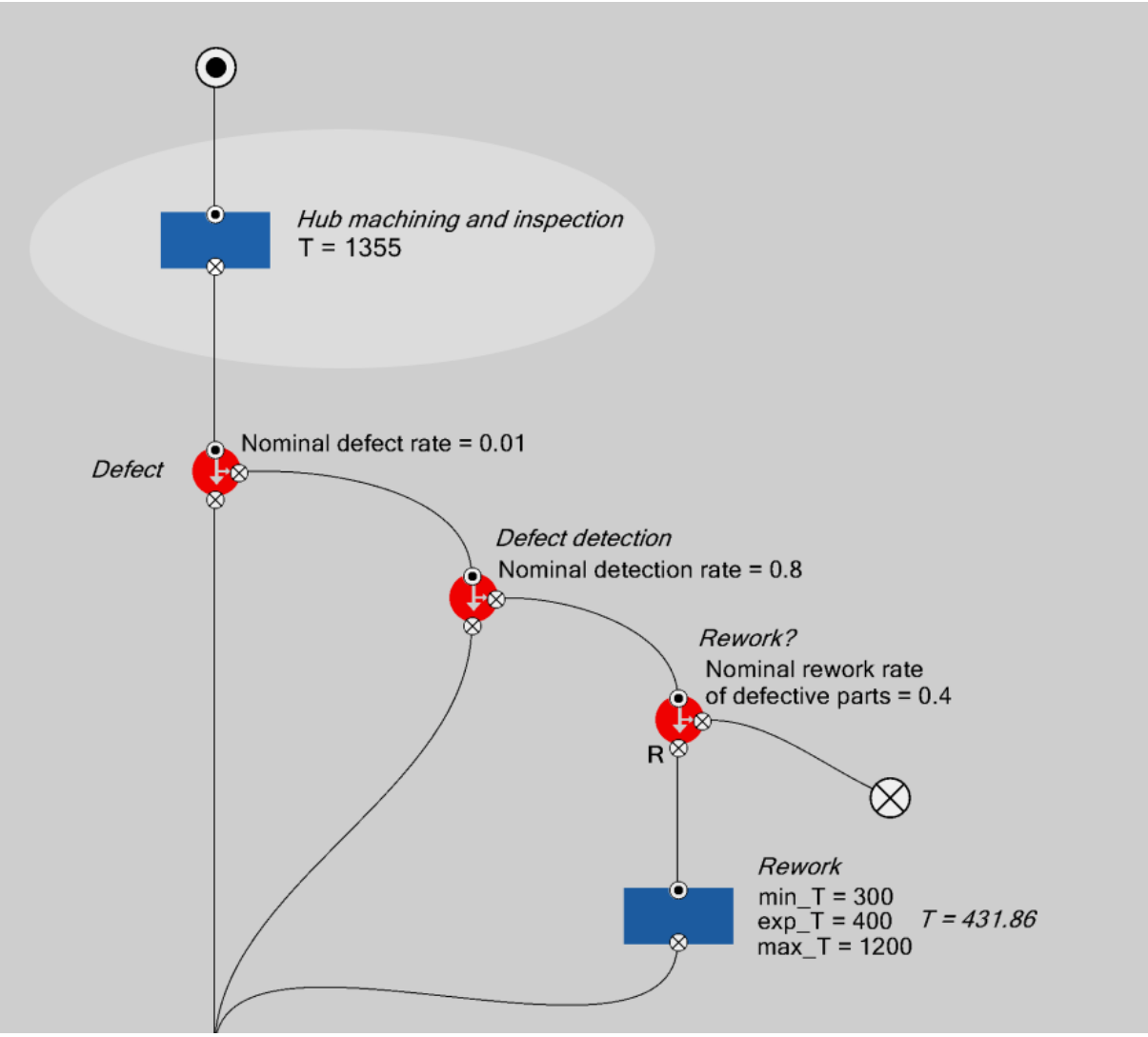
Lets go into the subsystem :

These blocks are deterministic in nature.



If we add up the time used (or use the Aliquis time measurers) we will see that it always takes 1355 seconds to get through them.

If you want people to engage with the process, you should leave them as they are. But if this part of the process is settled, there is no reason why you should not do this :

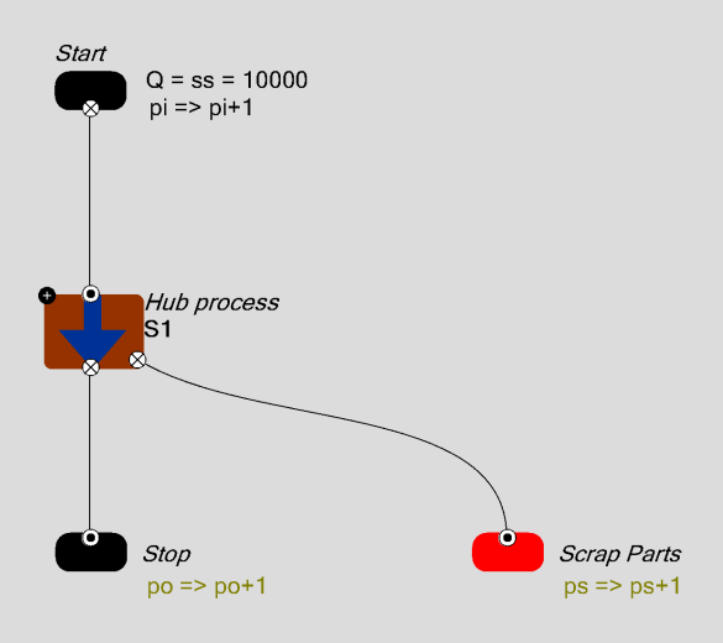


We will leave it to you to do some solve time trials. (For us it came to 45 % improvement in solve times).

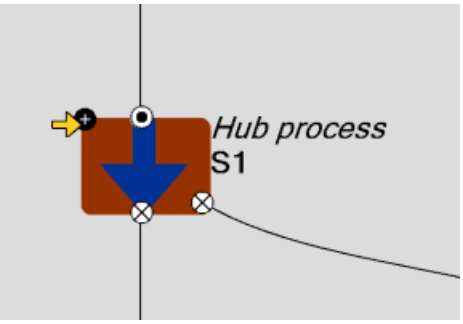
Paramaterisation of subsystem

We have already shown how individual blocks can be driven by system level parameters. They can also be driven by subsystem level parameters… We will drive the inspection process time and the inspection probabilities.

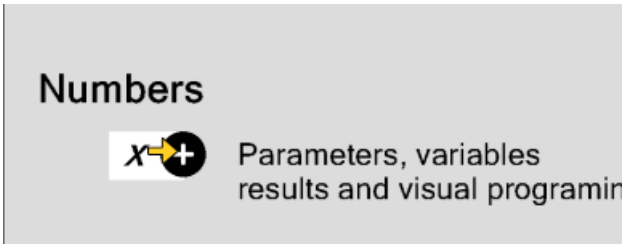
Go back to *Chapter6Ex1a.inp*



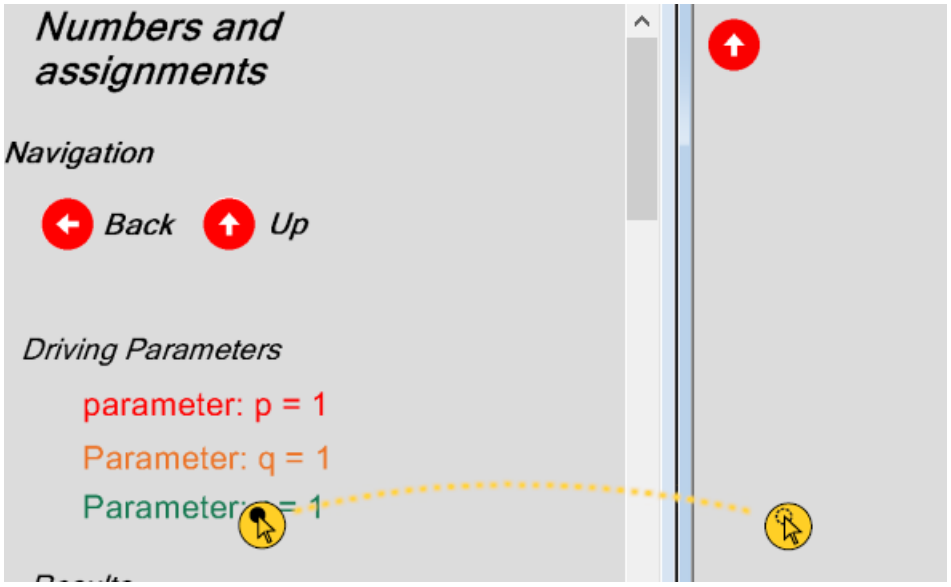
Enter the subsystem :



In the library go into numbers :



Drag 4 parameters into the subsystem:

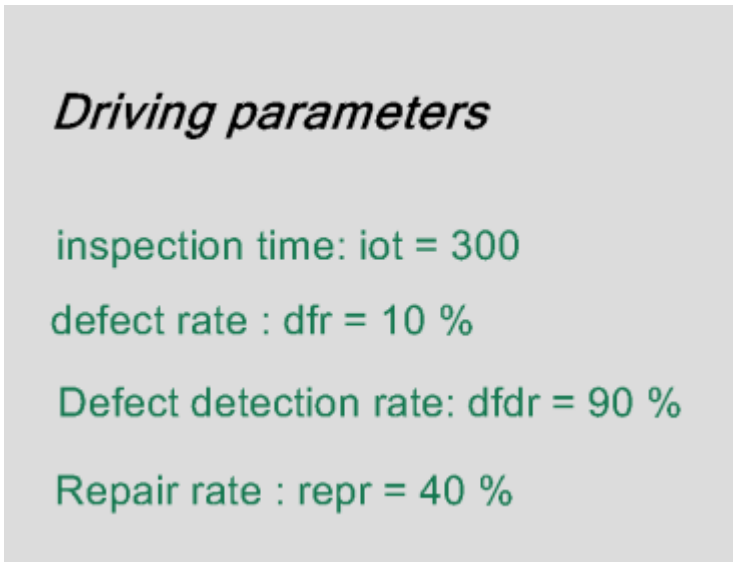


Hint : Make sure you drag them into clear space or they will attach to the blocks rather than the subsystem.

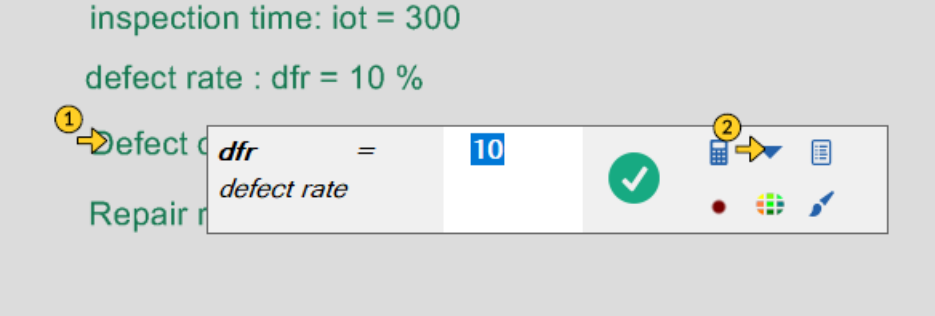
You can also drag some text to make it clear to users what they are.

Rename and re describe the parameters to get this :

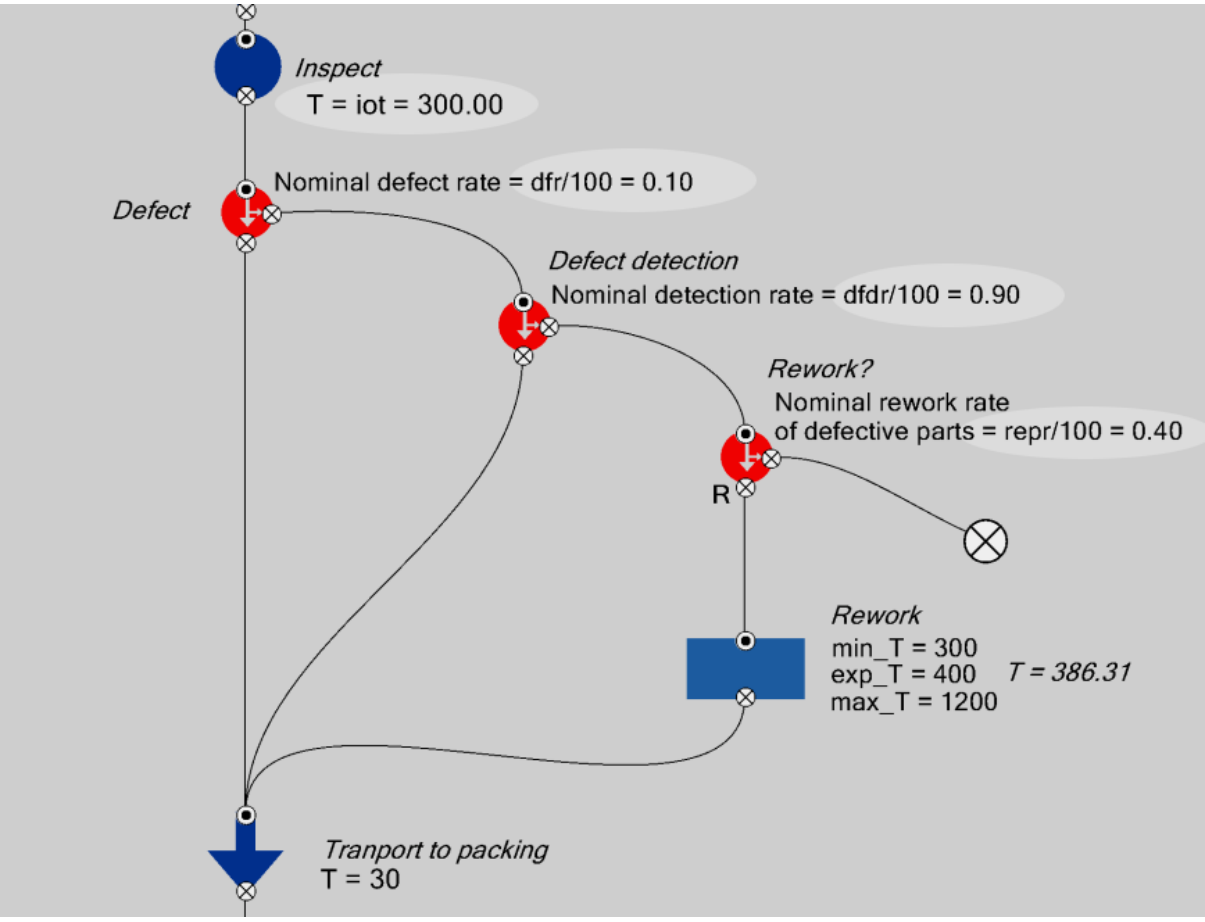
Result



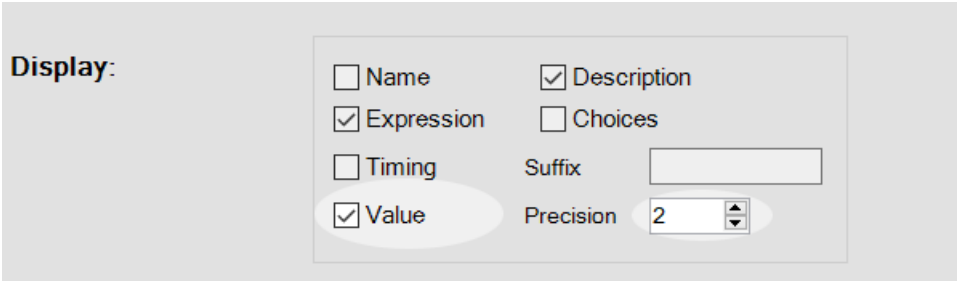
Hint : The percent sign can be added as a suffix to the attribute if you open up the attribute display sheet



Now within the subsystem set the block attributes to these parameters :

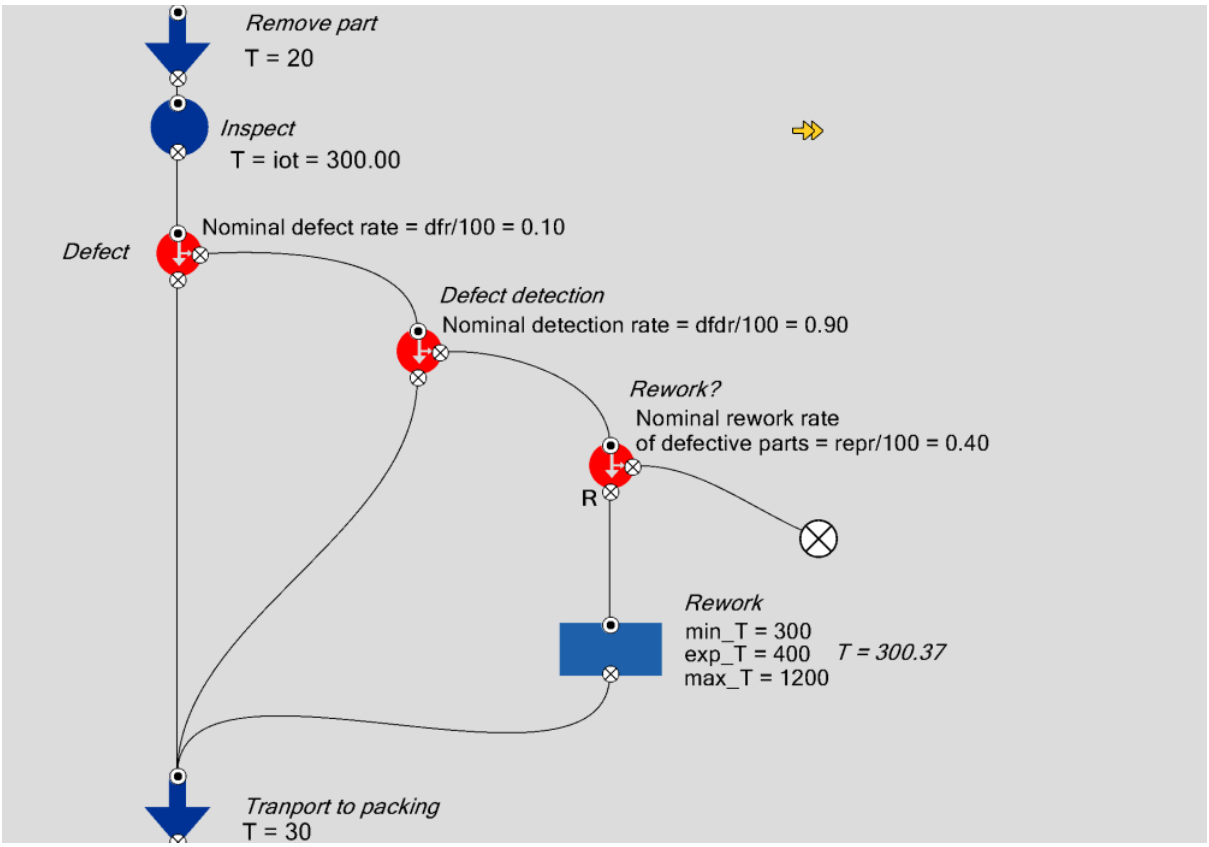


Hint : To see the value as well as the formula – open up the parameter and select display value:

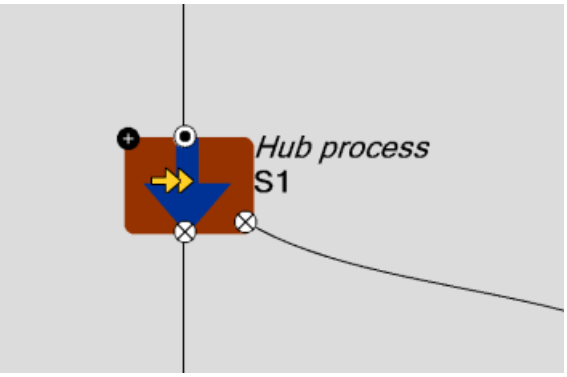


Now for the interesting bit… These parameters are driving the behaviour of our subsystem just like system attributes define the behaviour of blocks. We will now go outside the subsystem and make the parameters visible from there:

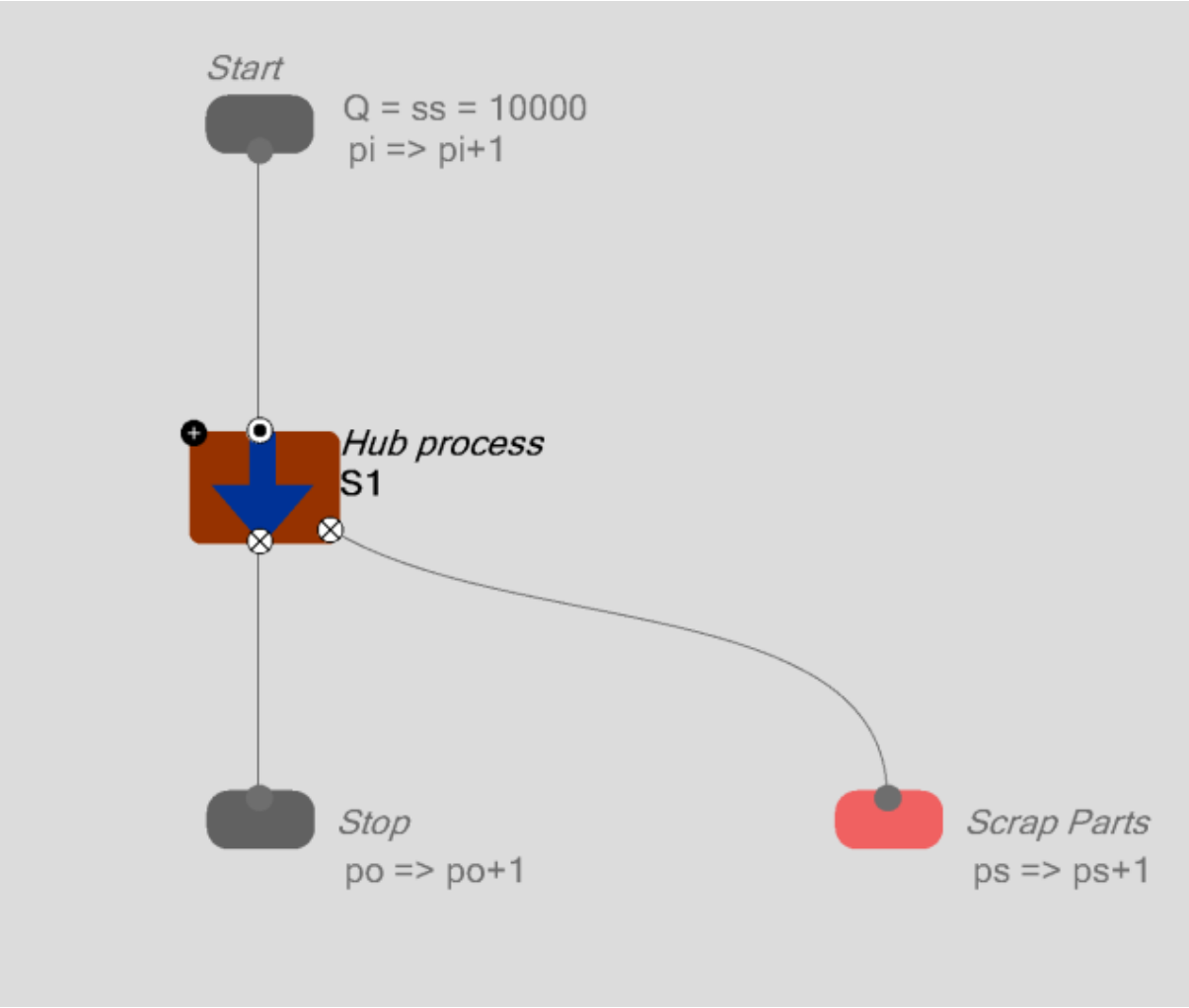
Exit the subsystem (double click in space) :



Now go into the outside of the subsystem by a double click. It behaves like a block:

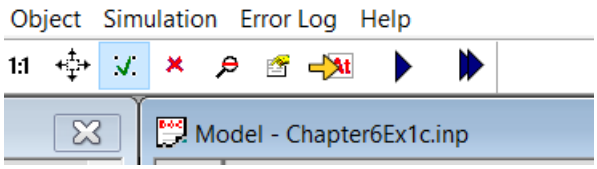


Result : (Notice the other blocks are greyed out. You are editing the graphics of the exterior of the subsystem)

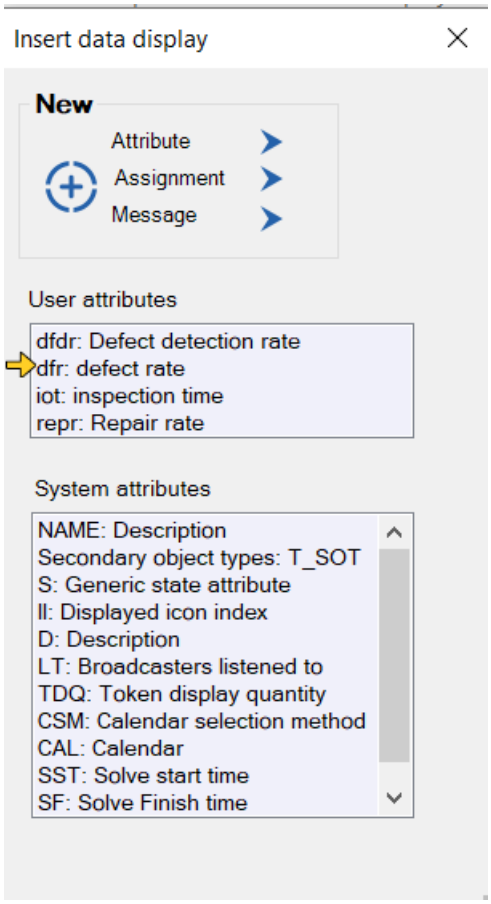


Do Not add another parameter as you did last time – this will create a new parameter in the system. Rather – we want to expose the existing parameters on this side of the subsystem.

Pick this button:

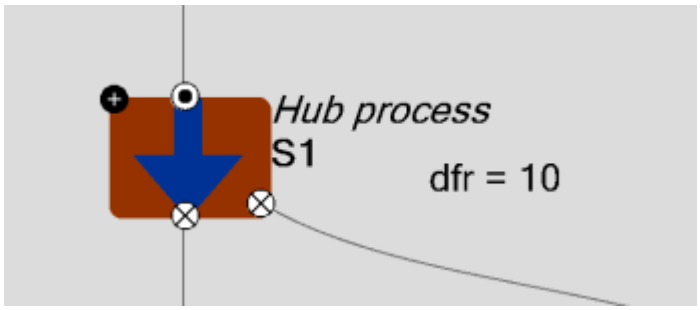


And pick the first of the parameters that you want to display :

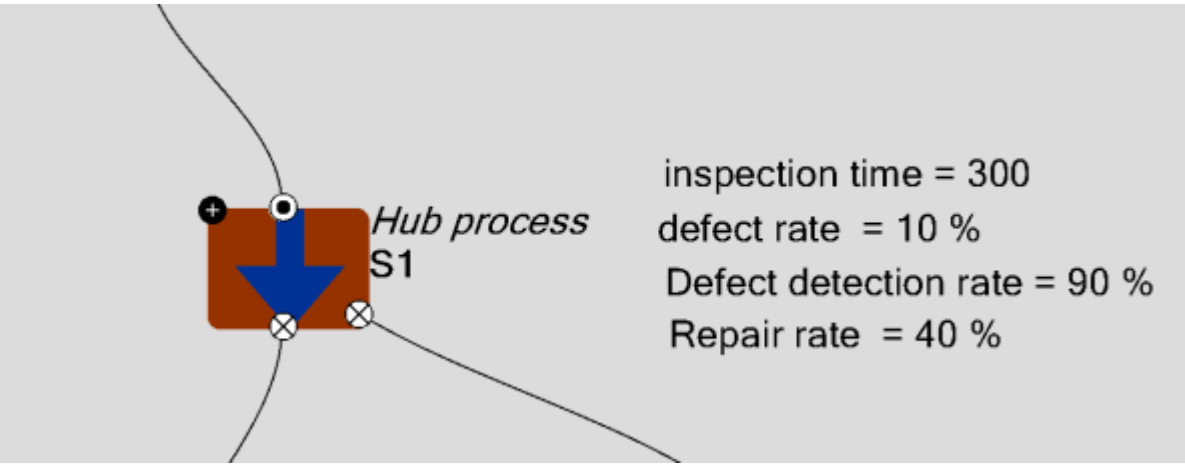


In the model, Click where you want it to appear :

Result :



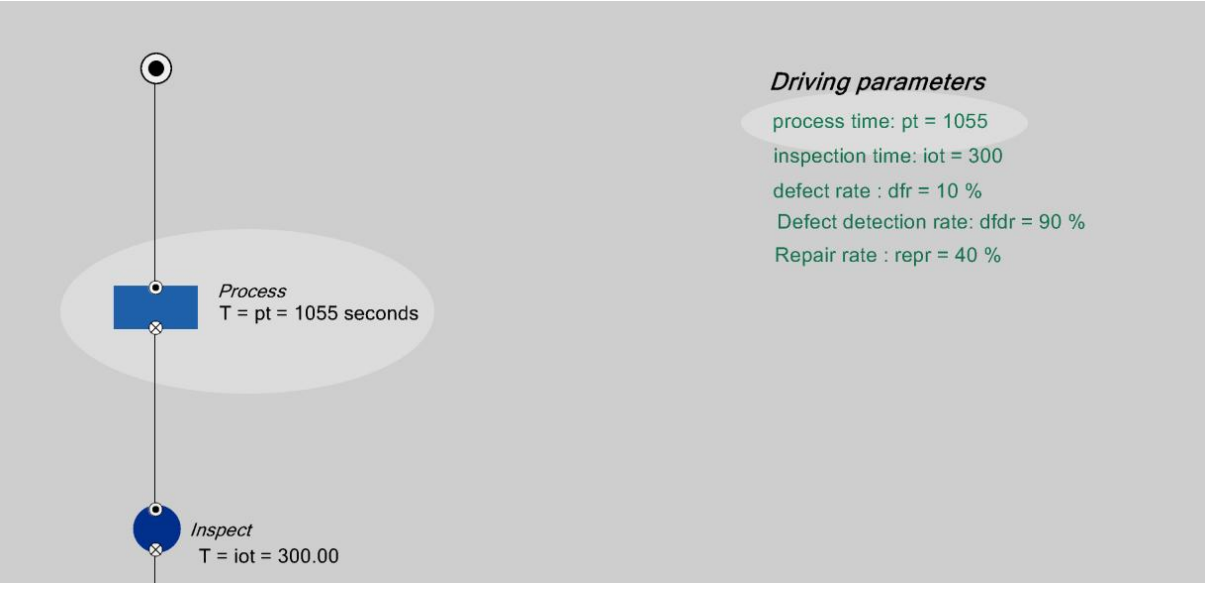
Repeat this for all the parameters you want to see and edit the visibility to get somethink like this :



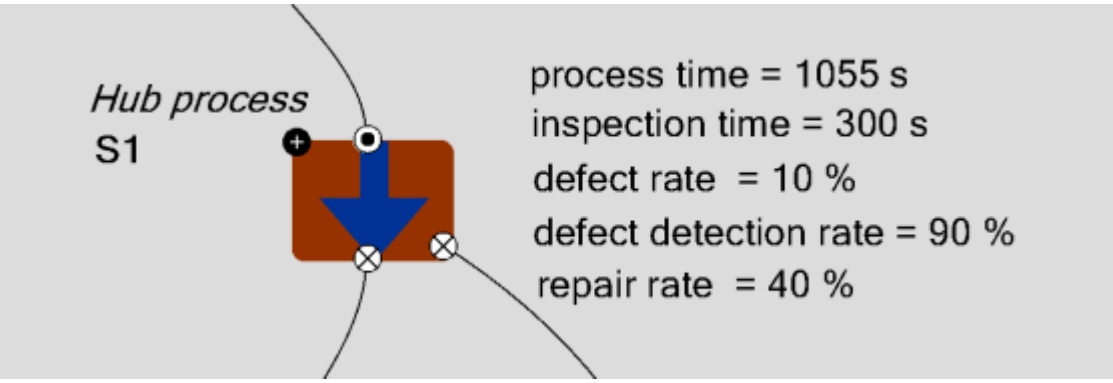
You can find this in *Chapter6Ex1d*

Exercise:

Go into your subsystem and simplify the process elements as we did earlier but excluding the inspection block. Paramaterise the process time



Make the subsystem look like this on the outside :



This is starting to look like a good generic block for examining the effect of 100 percent inspection across more complex systems. But how do we get results from our simulations?

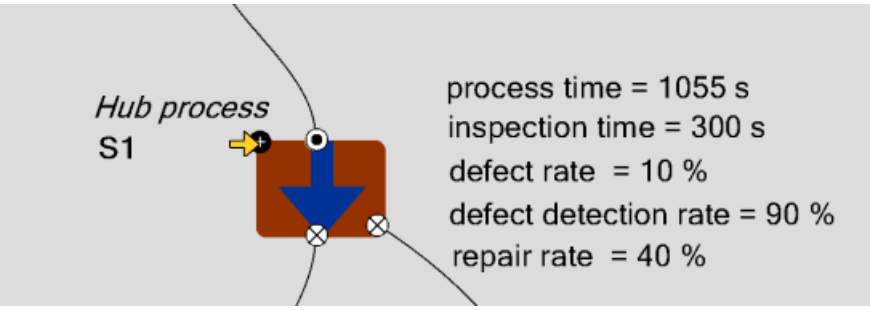
Well there are two ways – you can calculate results internally or externally. We will look at both methods with this subsystem.

If you have not succeeded in getting to this stage open *Chapter6Ex1dDone*.

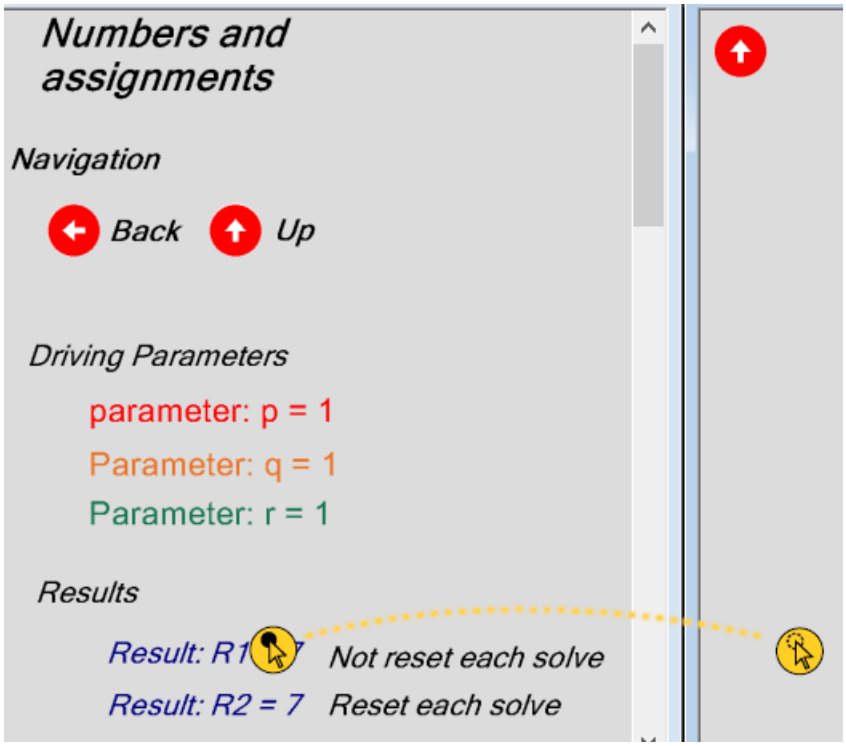
Results computed on the inside

We will examine a key result. Both internal and external methods will be employed to enable you to compare and contrast the methods. Lets start with calculating this metric internally…

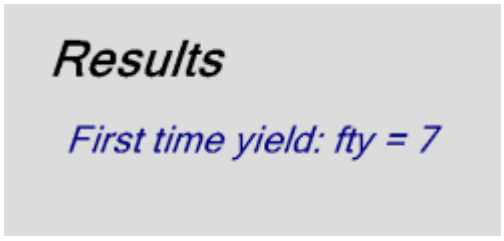
Go into the subsystem :



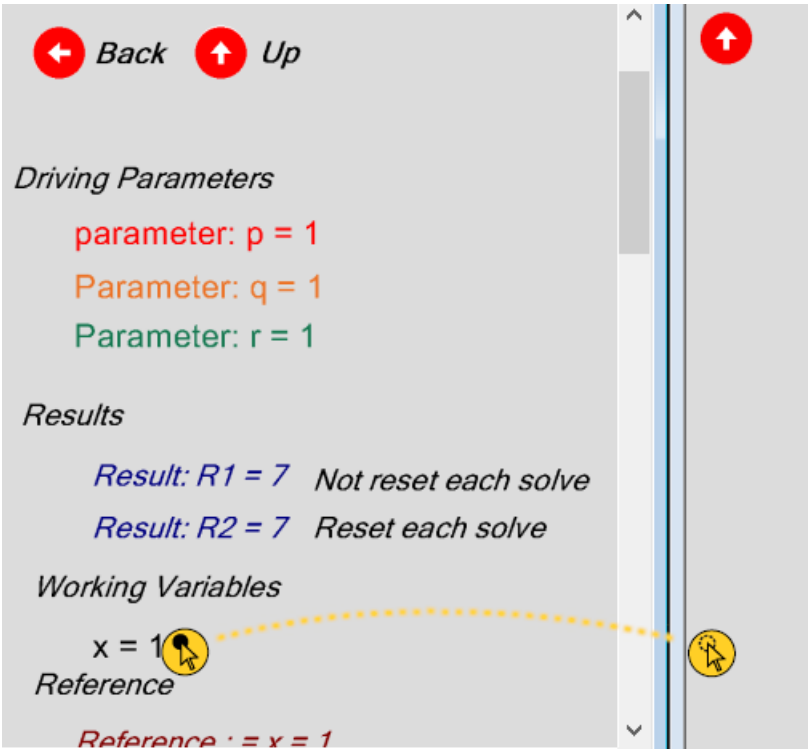
Add a result variable for first time yield :



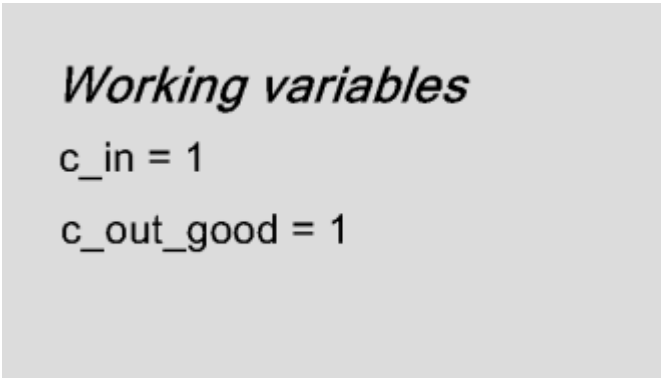
You are aiming for this :



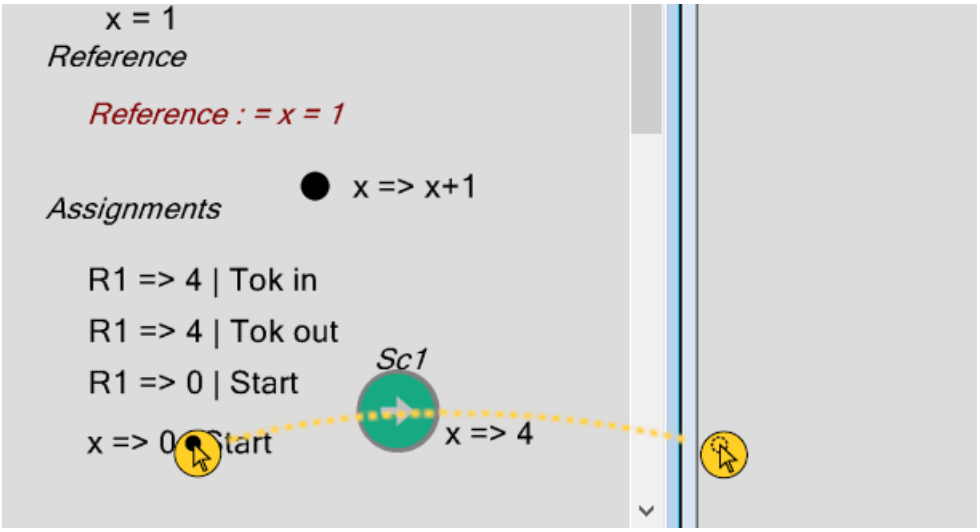
It is actually possible to get the first time yield with probability theory in this simple example, but we will take a modelling approach as it works regardless of the complexity of the model. Need to count parts in and parts out through the “good” channel. Lets put in a couple of variables for these counts:



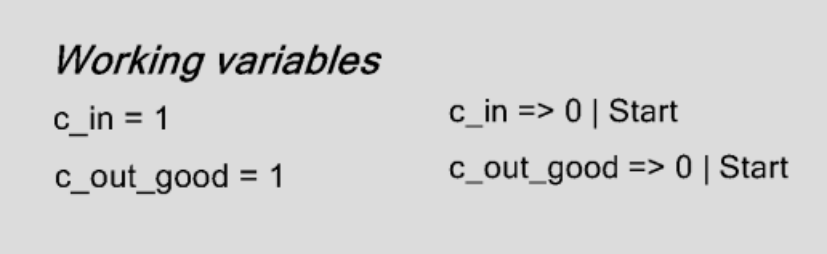
You are aiming for this :



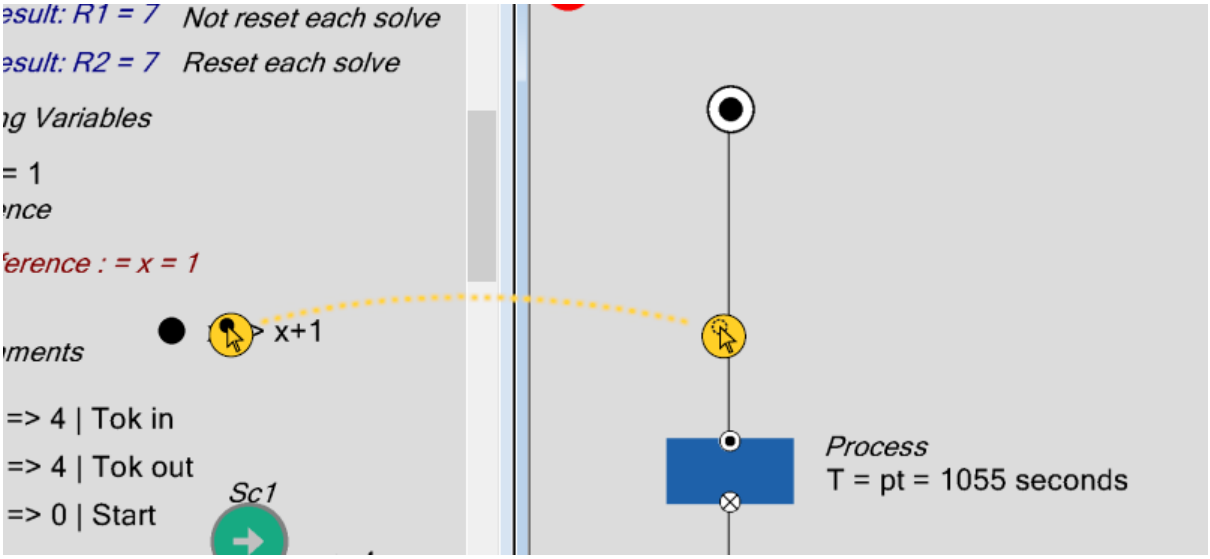
But we also need to initialise them to zero at the start of the solve:



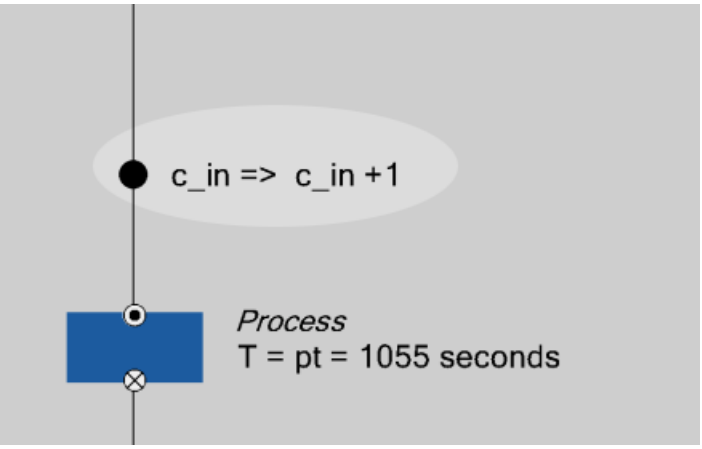
Aiming to get this :



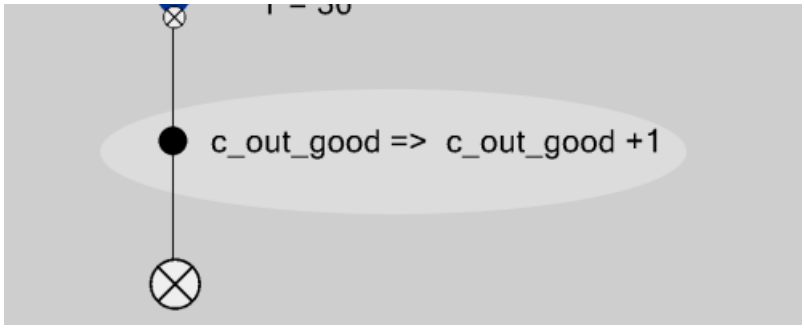
Finally we need to set up the counters:



Aiming for :

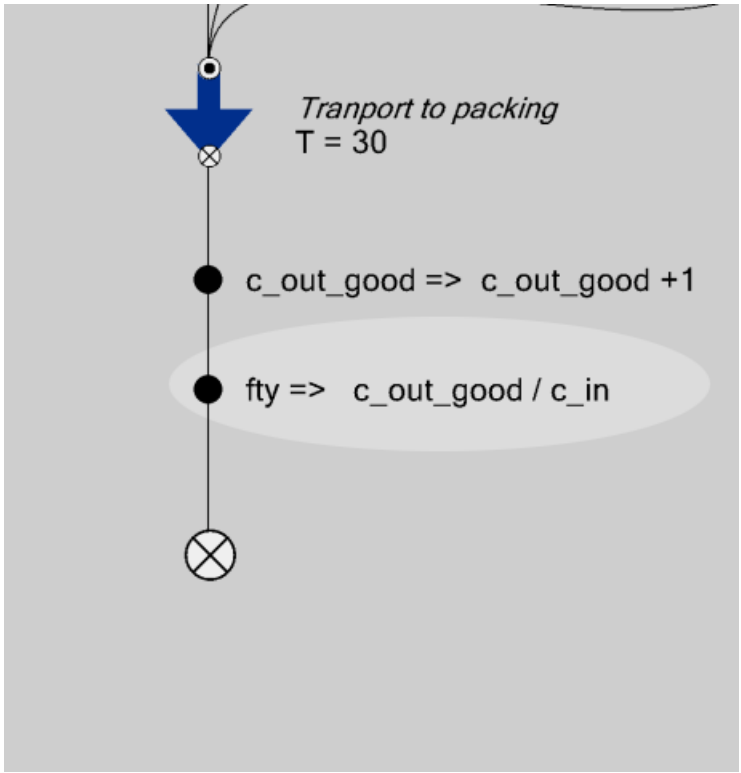


And :



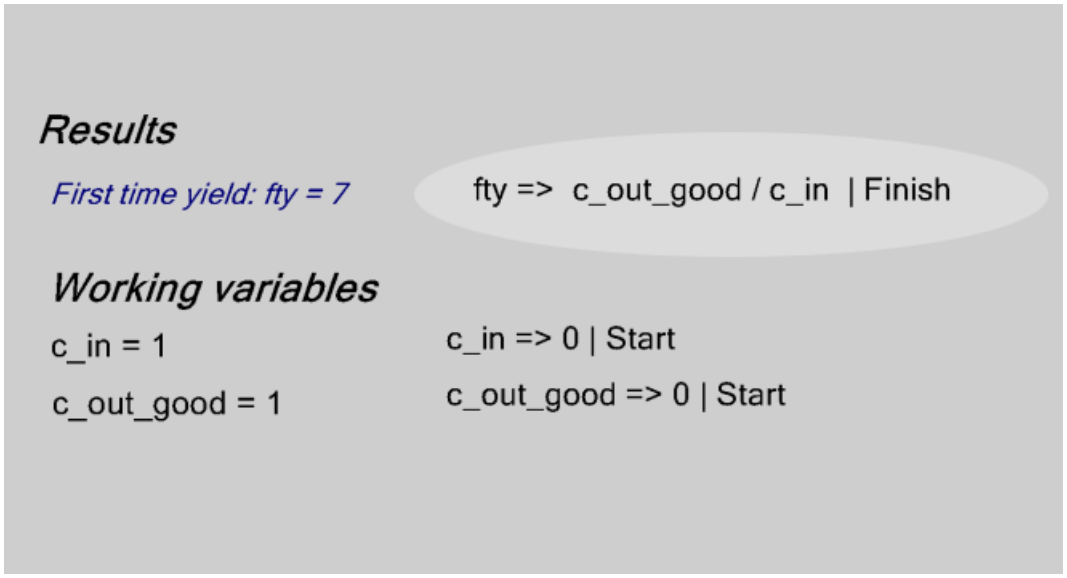
Finally we need to calculate the result..

We need to think about our requirements here. If we calculate the yield as the solve proceeds we could simply do this :

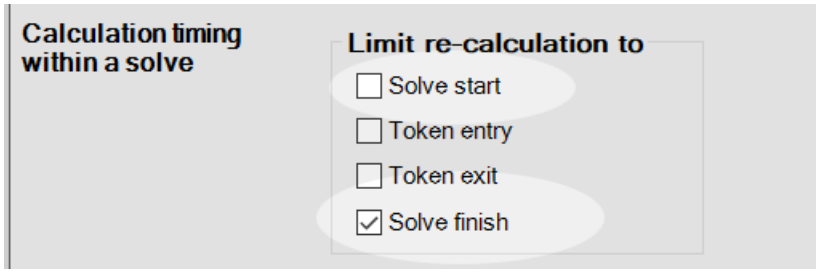


This will calculate a new value for yield every time a token passes. If you want to save the variable it may generate a lot of intermediate values of the yield which you don't want.

If you only want to calculate the yield at the end of the solve do this instead :

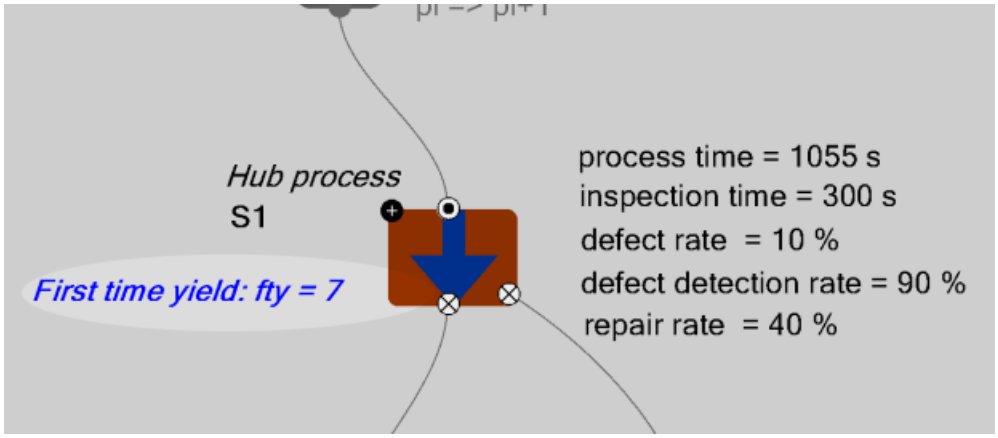


To get the assignment to set when the solve finishes :



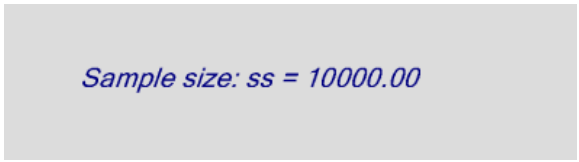
Either way we can now exit the subsystem and bring in the result into the exterior side .

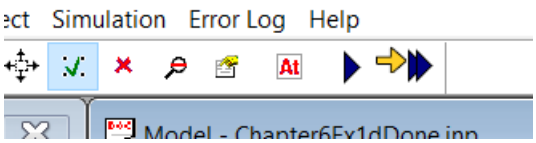
Exercise – display your first time yield result variable like this on the exterior of your subsystem



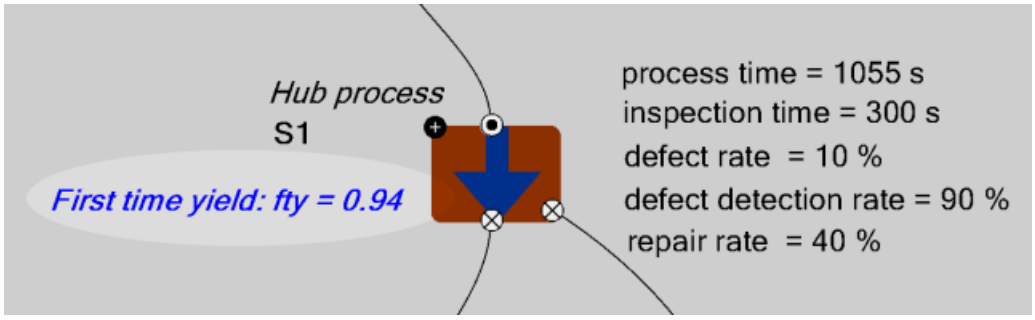
Of course at this stage it is set to 7 because we have never solved it.

Make sure your sample size is reasonable and run a quick solve:





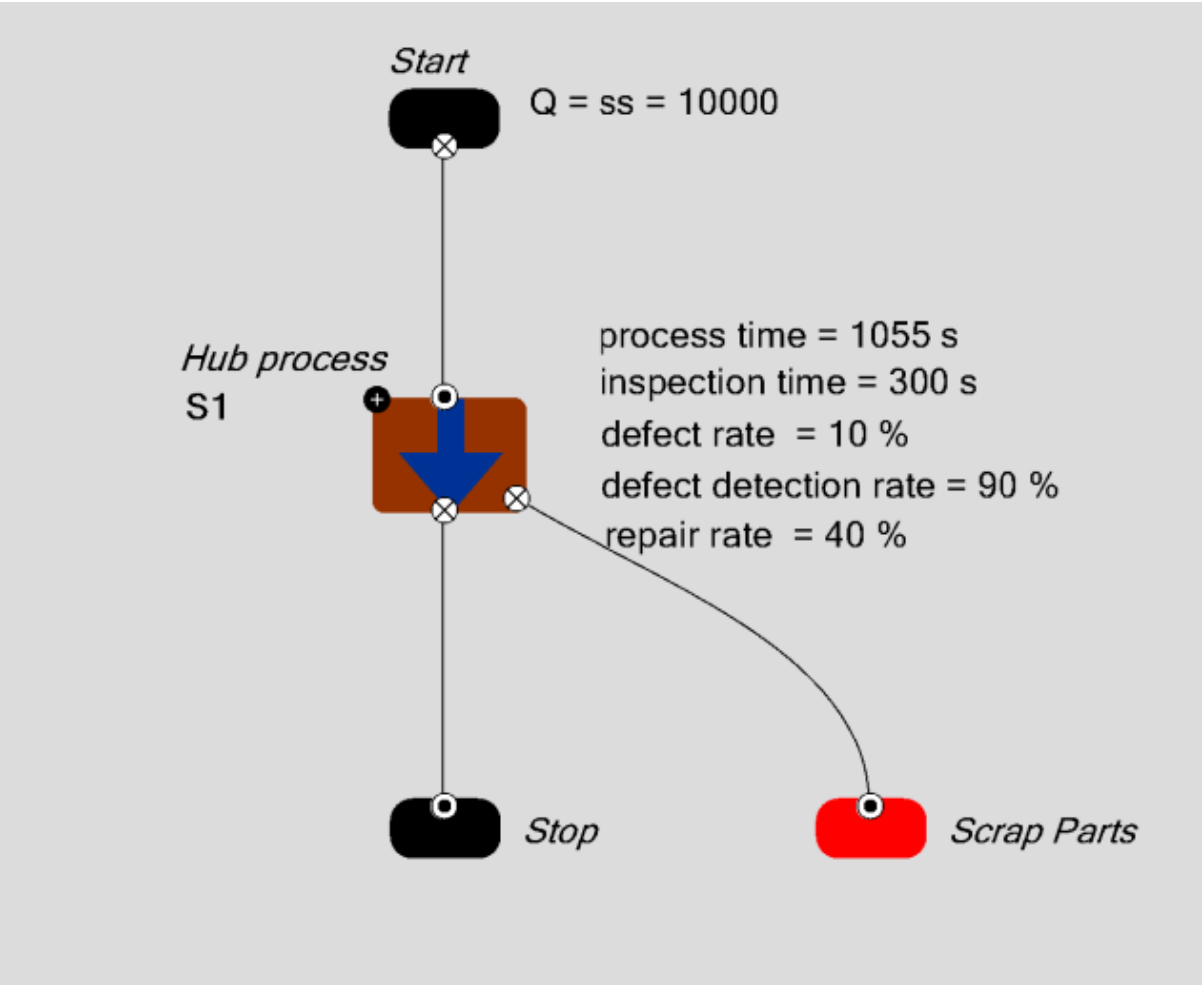
Result :



If you got stuck or need a model to reference you can find it here :*Chapter6Ex1eDone*

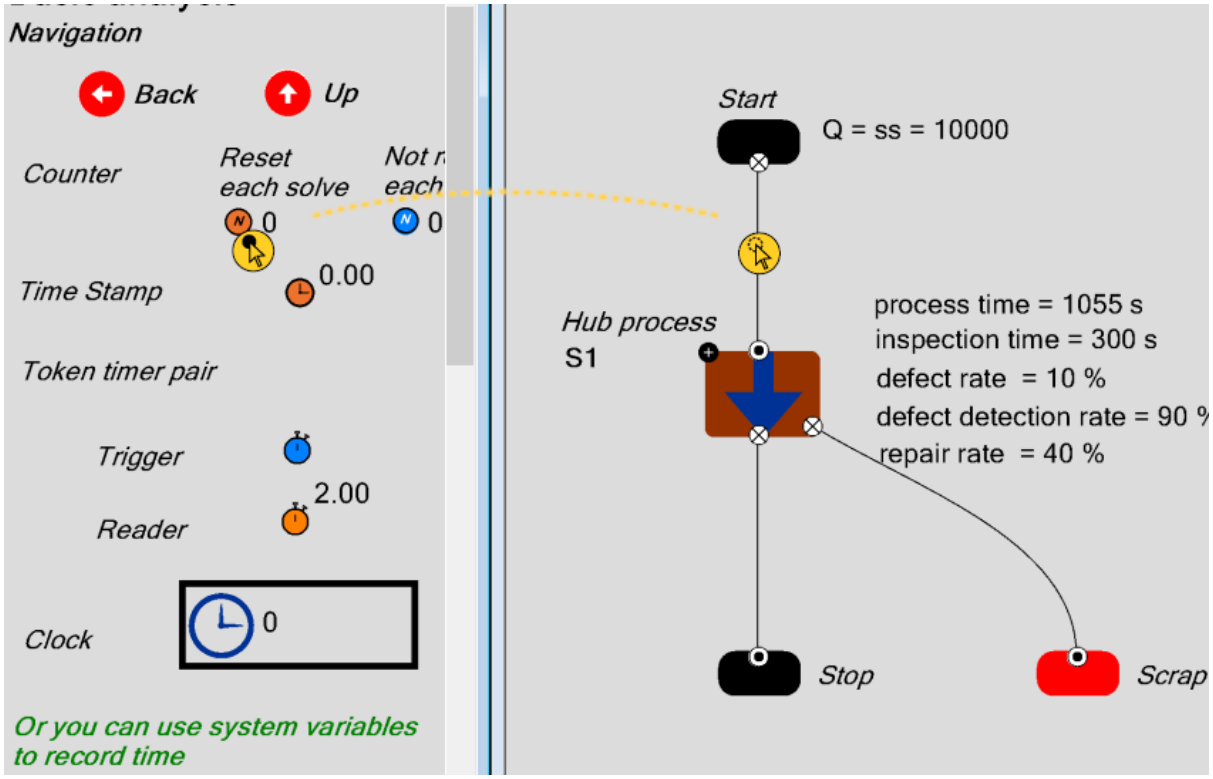
External calculations

Go back to *Chapter6Ex1f*

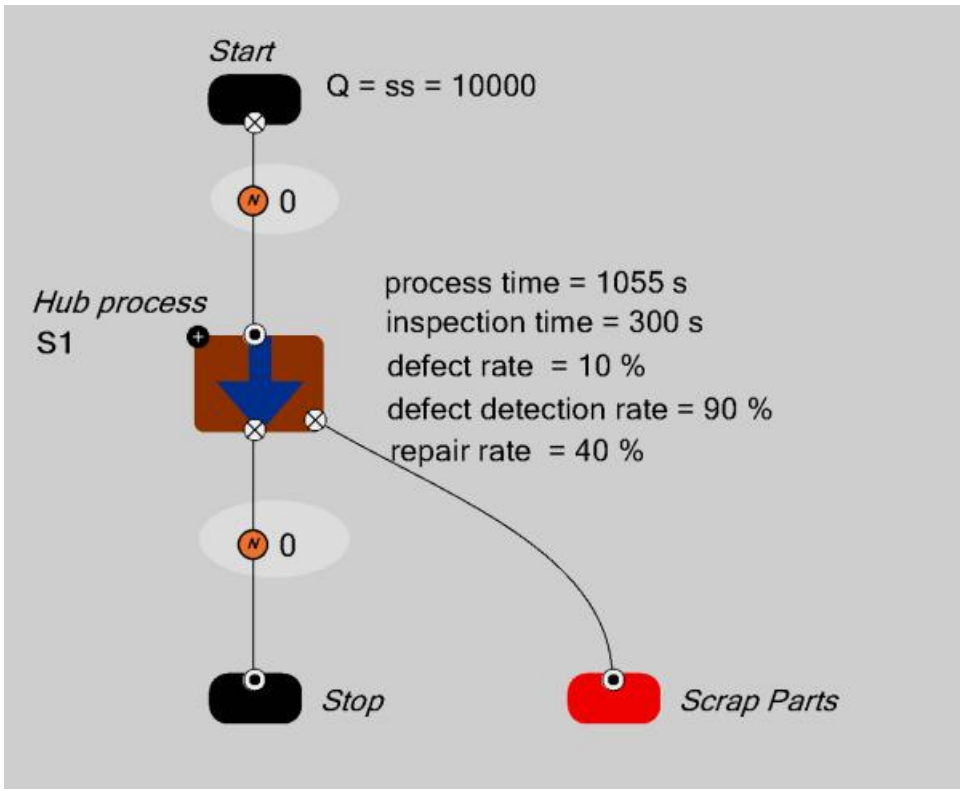


This part has had all external assignments stripped out to make sure you understand how the subsystem can be treated and examined as a black box from scratch…

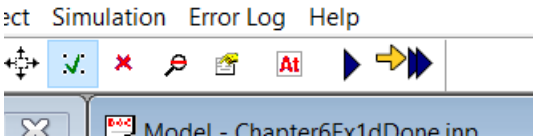
The simplest way of getting an idea of first time yield is simply to put counters on the streams :



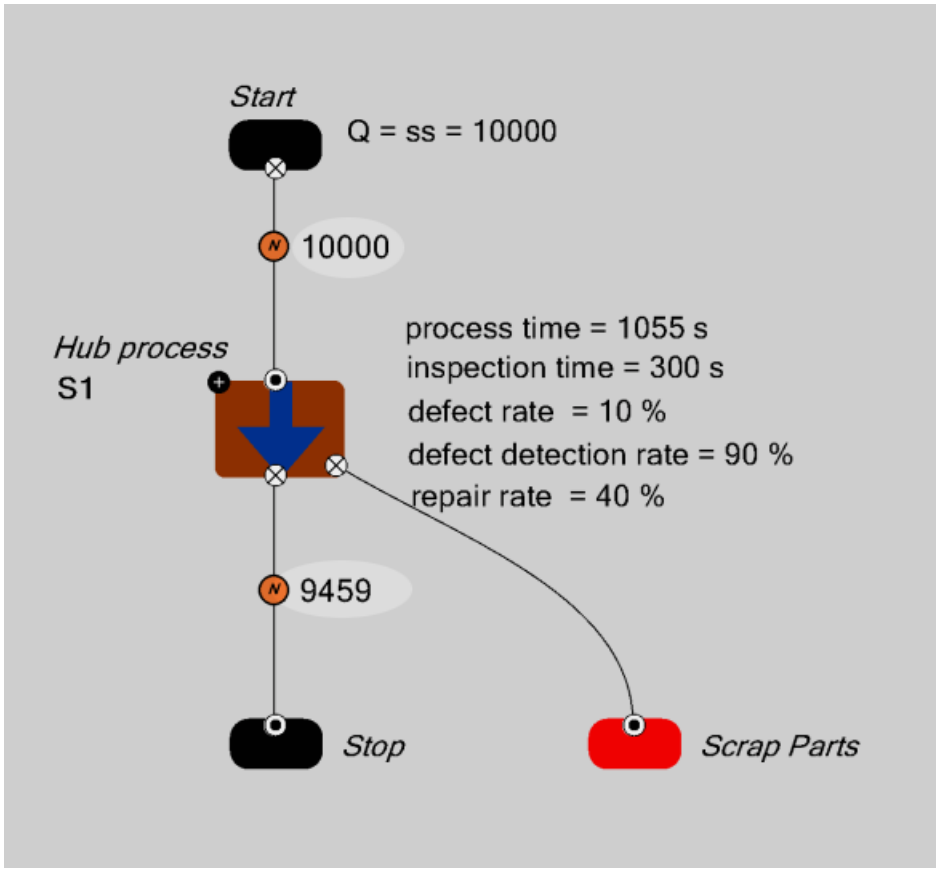
Likewise put the same resetting counter below the subsystem :



Do a quick solve:



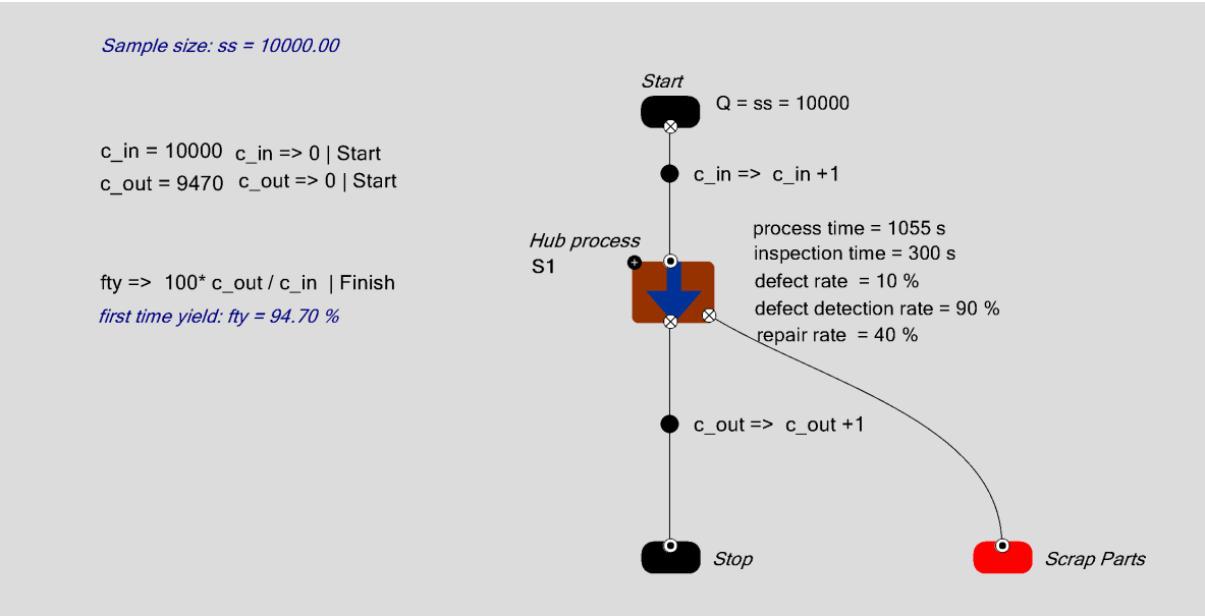
Result :



By inspection we can see that the first time yield looks to be 94.6%

However you might prefer to do the same thing with counters :

Exercise : Calculate the first time yield using numbers :



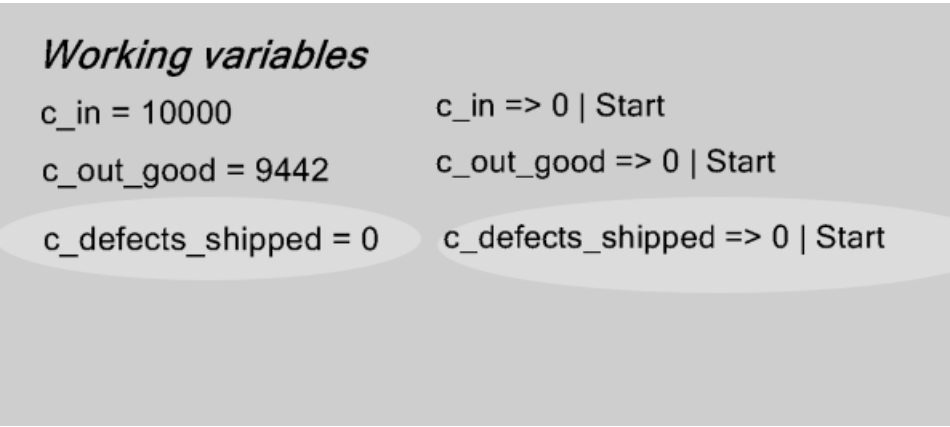
If you have any problems, you can use the model *Chapter6Ex1fdone.inp* for reference.

We now come to a difficulty – some of these good parts are actually defective – its just that the inspection system has not found them. In the real world they would not be counted – until the customer phones up complaining!

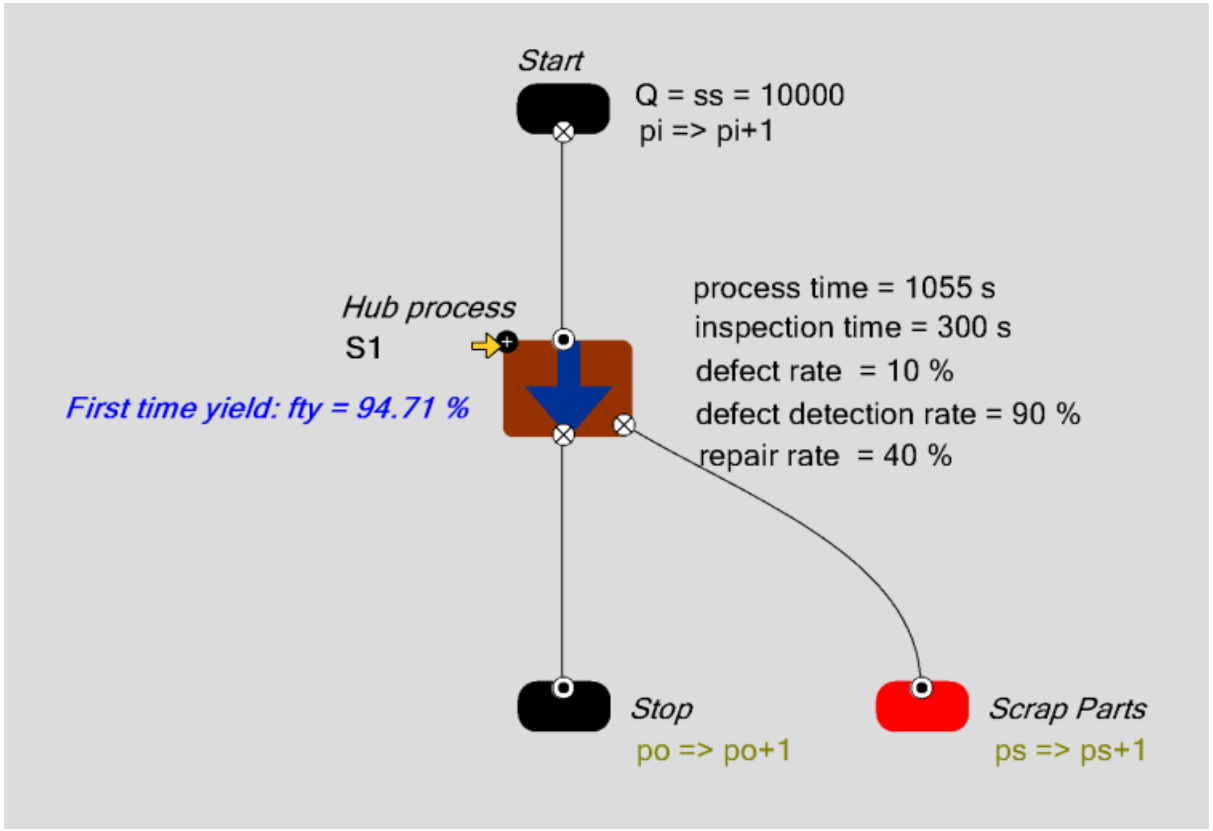
To count them internally is relatively straightforward :

Go back to *Chapter6Ex1eDone* and go into the subsystem:

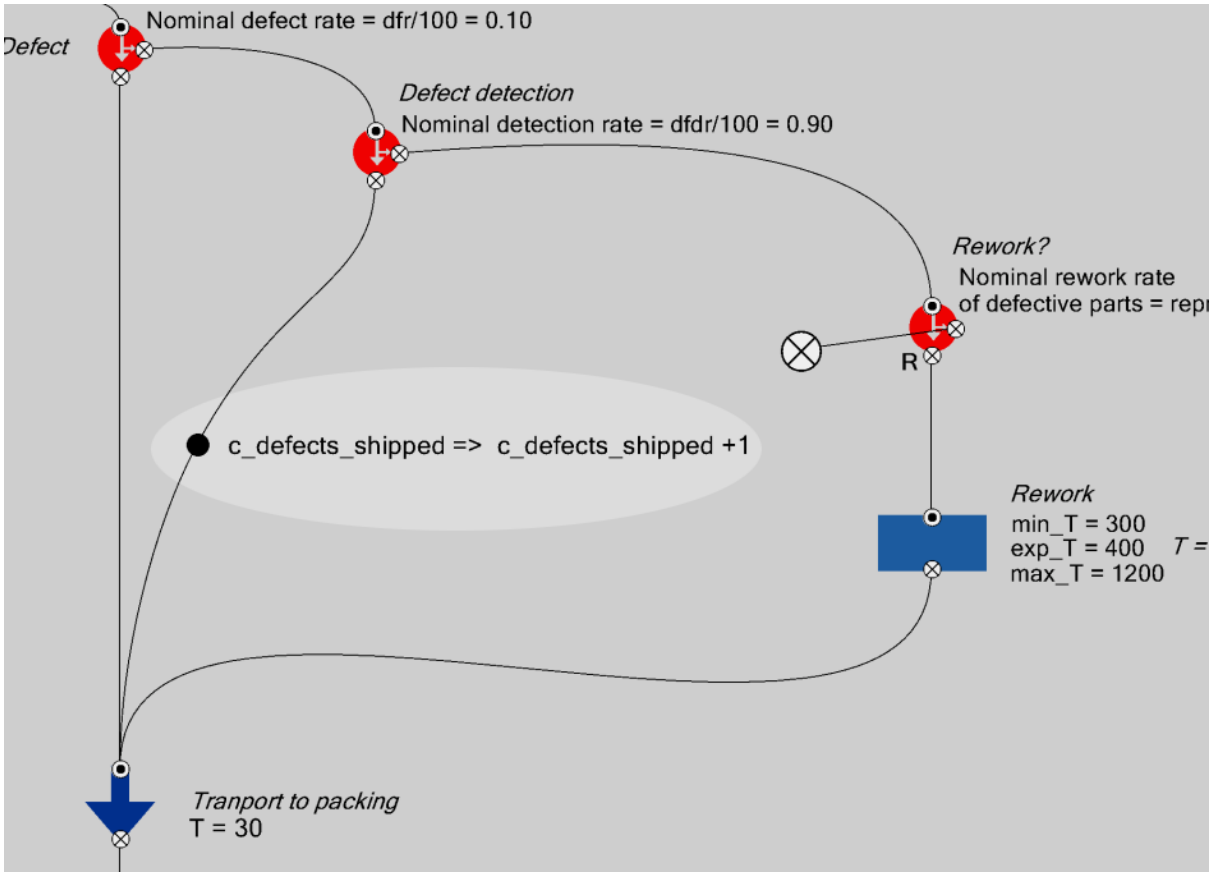
Create a new variable to count the defect parts shipped and initialise it to zero at the start:



Increm

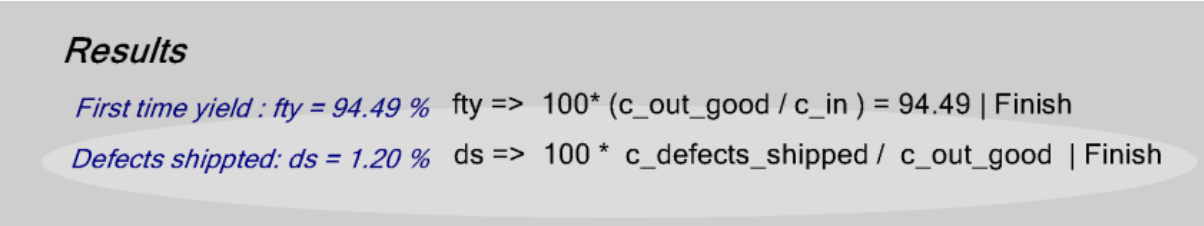


ent it here :

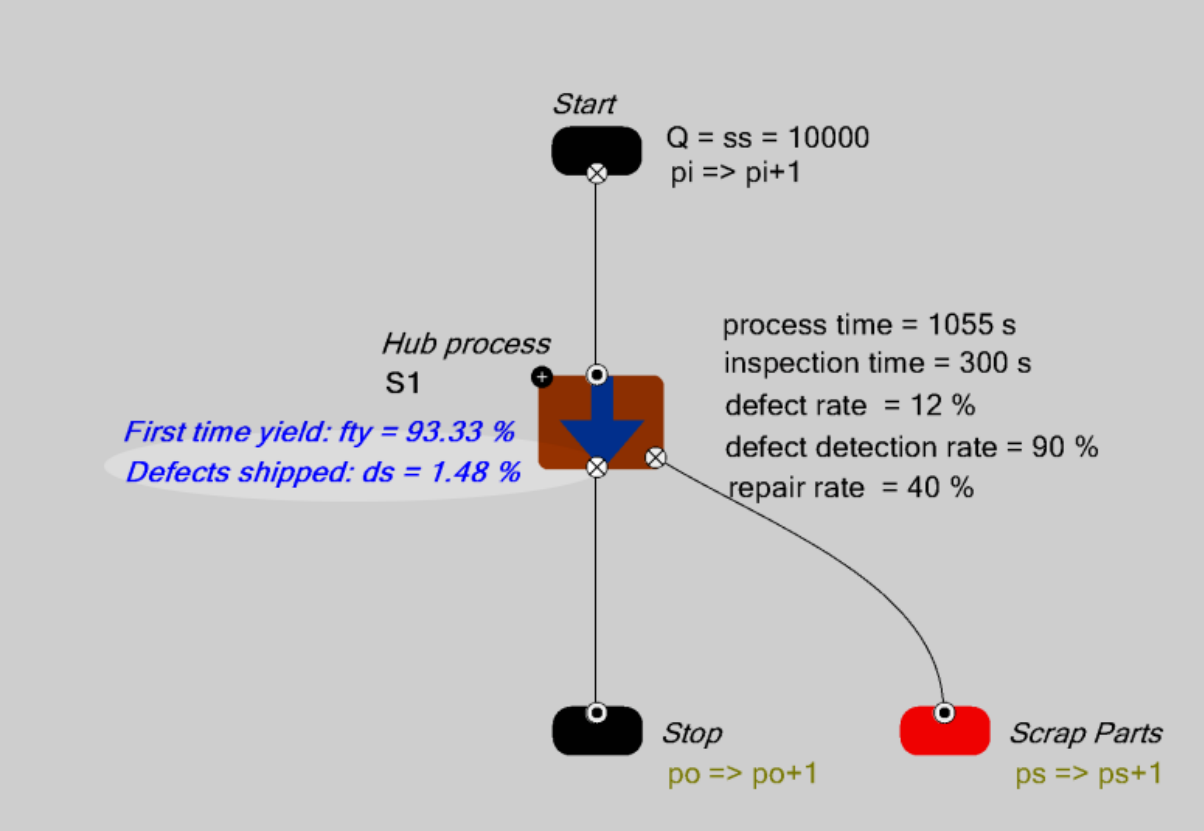


Now create a result for the percentage of defects shipped :

Calculate it at the end of the solve :



Finally Expose it on the outside and solve :



This can be found, for reference in the model *Chapter6Ex1gDone.inp*

Adding intelligence within tokens

Go back to *Chapter6Ex1f.inp*

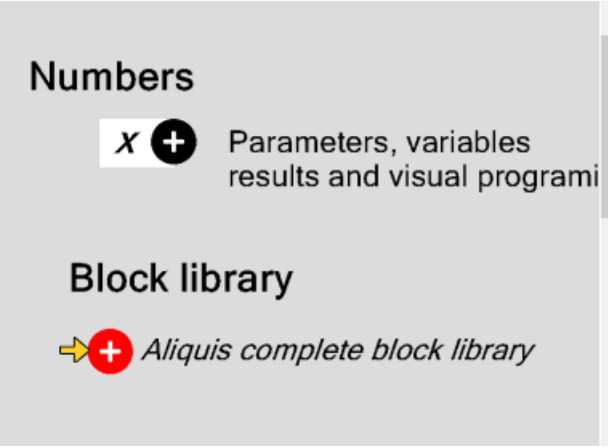
You may remember that in chapter 5 we did an optimisation exercise on how much inspection much inspection was optimum for a high defect rate system. In this case we put on a high cost for every time a defective component got through the system.

We therefore need to identify the defective items which escape to the customer in some way.

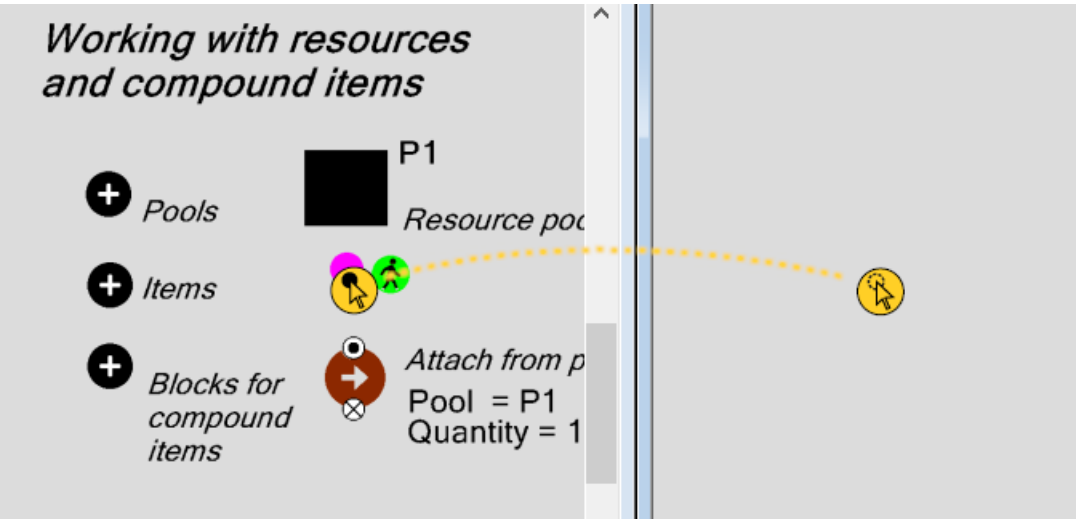
In Aliquis, this can be done by putting data into the tokens. We will go through the technique in isolation first with a simple study then we can apply it to our problem.

This will be the first time you use the full Aliquis block sub library. This library contains everything but it is organised on Aliquis functionality lines rather than rather than the manufacturing planning workstream based lines you have seen so far.

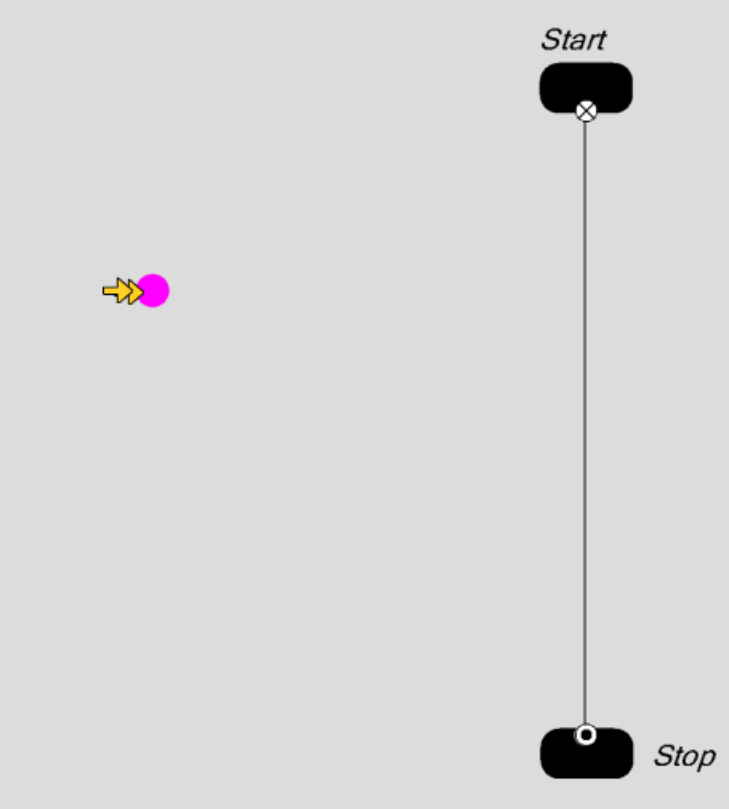
Go into it :



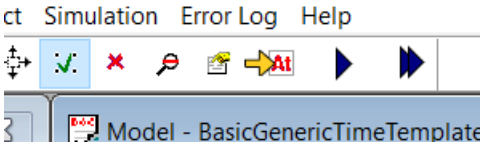
Drag the purple item into your model :



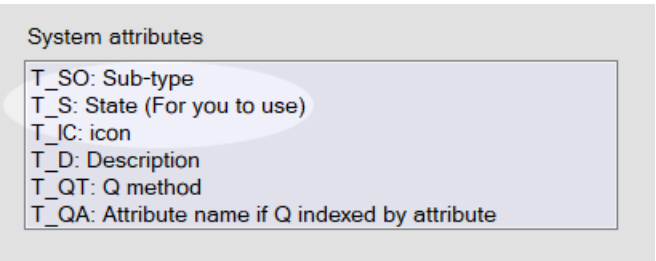
Double click the item in your model to go into it :



Let's have a look at the system attributes which are already in the item :



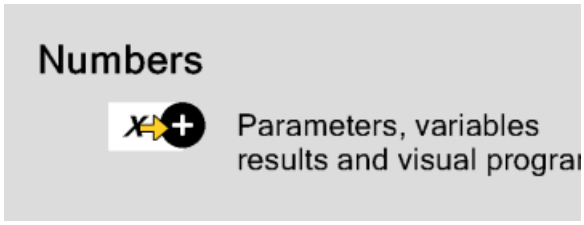
Result :

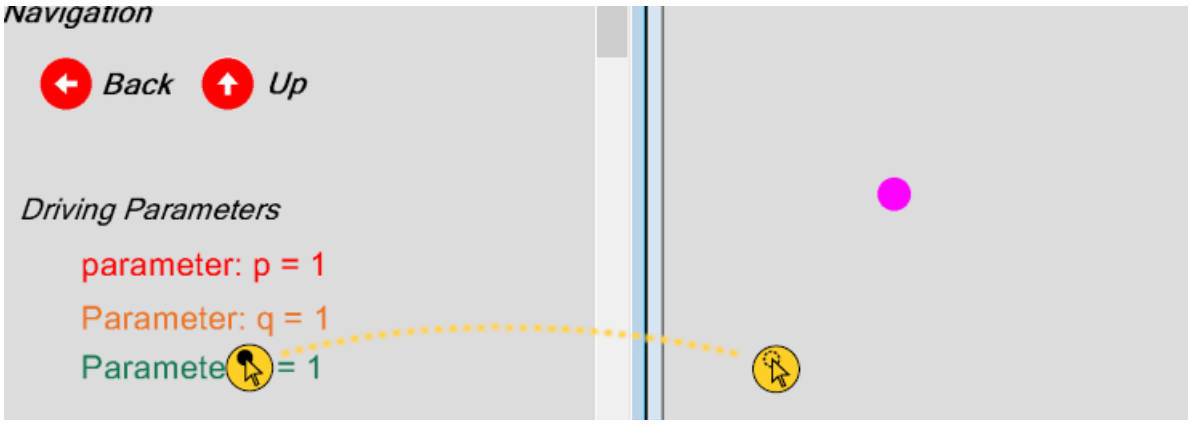


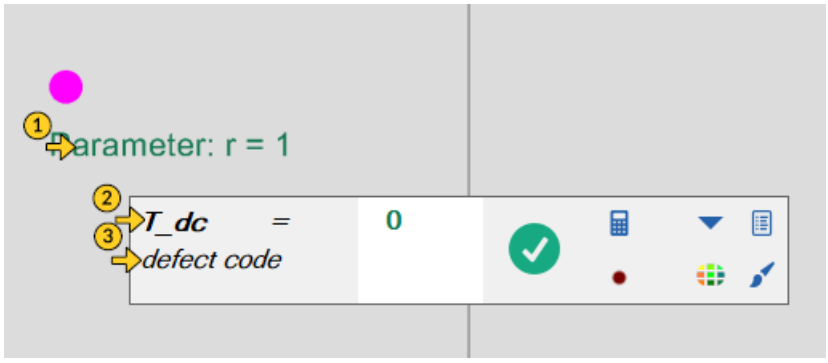
Most of these system attributes are used by Aliquis to determine the behaviour of the item. However , *T_S: State* is an attribute which is reserved for user use.

We have the option of using T_S as a flag for defective parts.

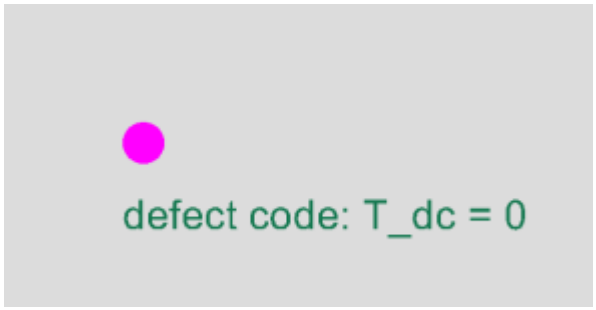
We also have the option of adding user parameters to items eg :



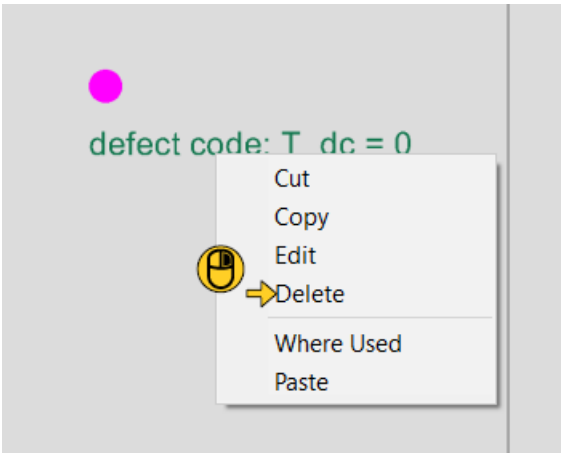




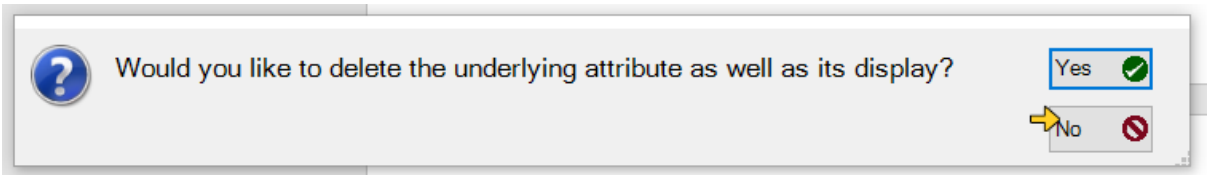
Result :



In this case you would probably not want see the variable's display. You can delete it :



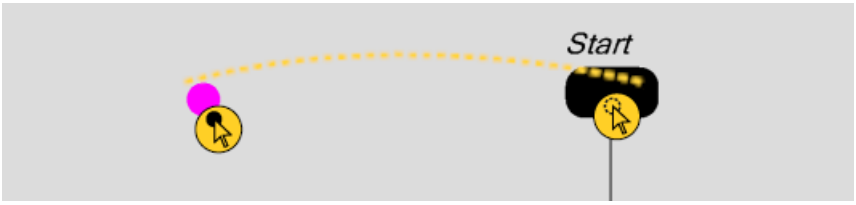
But make sure you do not delete the underlying variable :



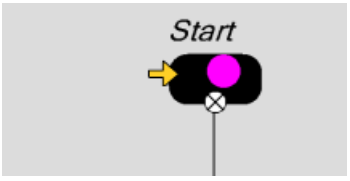
This item can now be used like this…

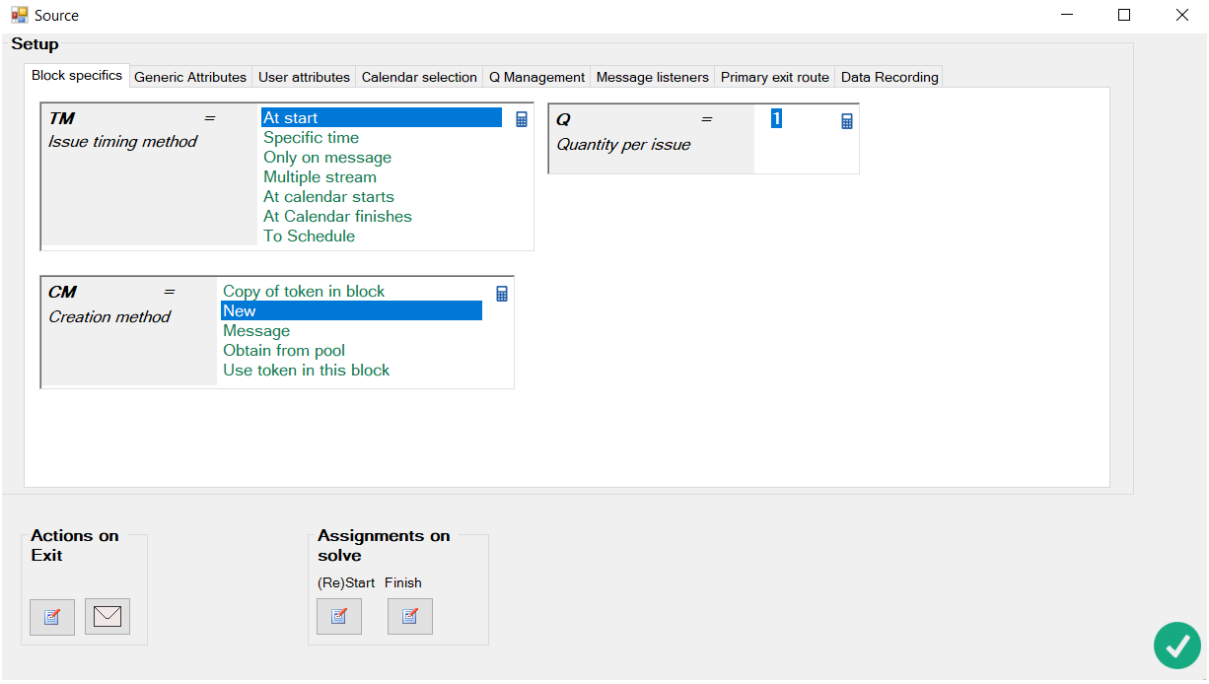
Double click in space to get back into the model.

Drop the item into the source :



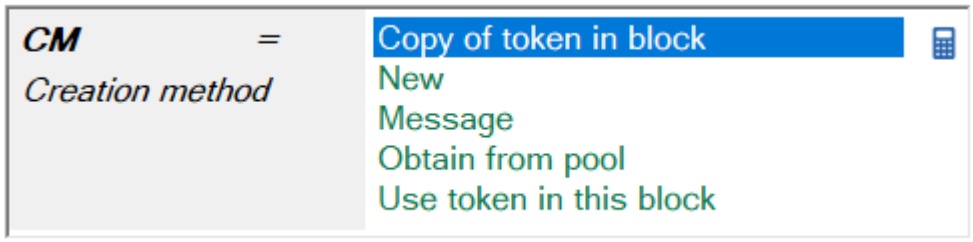
Now edit the source so it issues a copy of the item it contains :





Welcome to the block editor! You may have already seen this if you have accidently clicked a block. It can be used whenever you want to change the way a block works. It is a user interface to the underlying system attributes. Up to now we have learnt how to manipulate the attributes via attribute displays – but this is another way of doing it. Both methods have their place.

Change the creation method to "Copy of token in block"

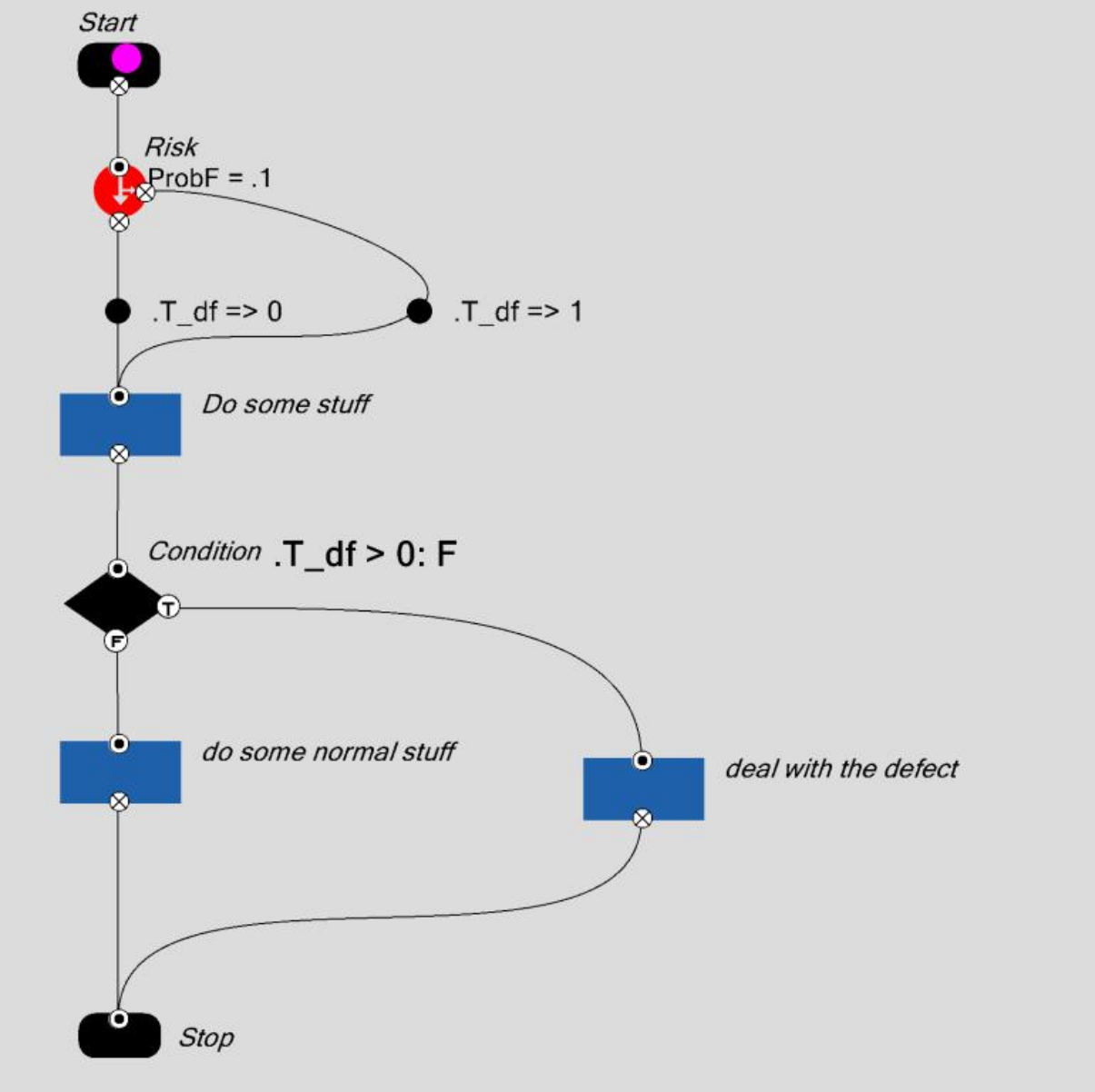


We can now use items with a defect code variable.

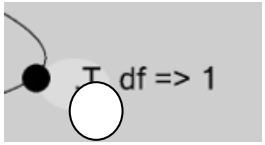
Alternatively you could have not set up the item and the block and just used T_S (the state) as a quick substitute.

Either way, we can use these token variables in modelling. They can be set and interrogated.

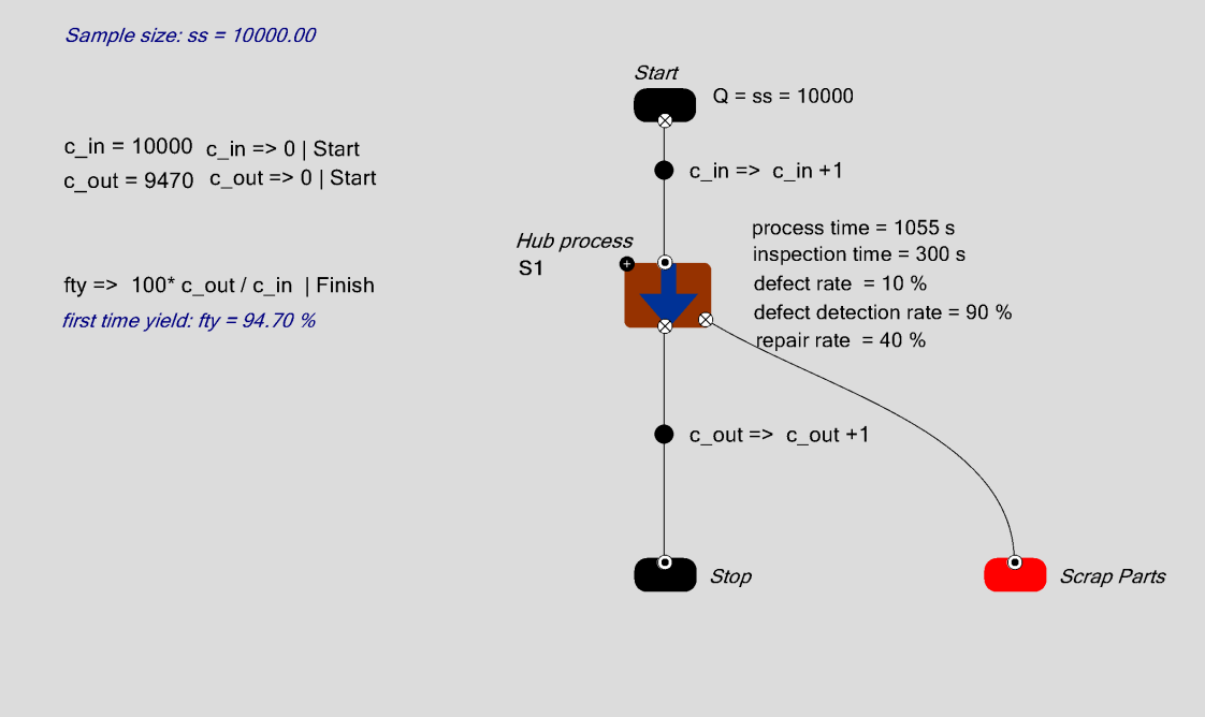
Open up Chapter6Demo2 :



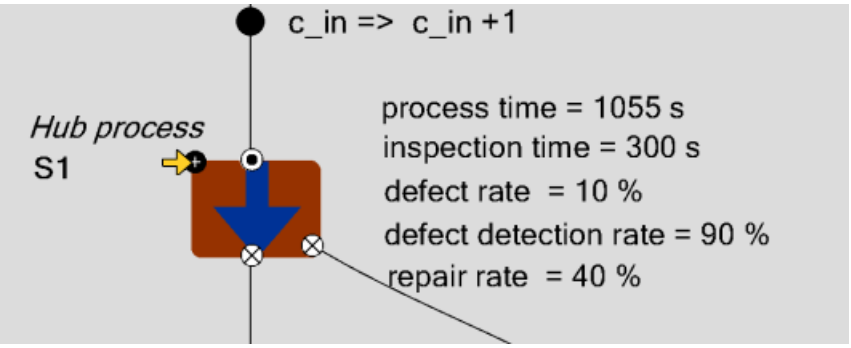
Note : In the main model , data on tokens is referred to by adding "." Before the name:



Lets now apply this idea to our black box model. This time we will use the state variable T_S to record defects . Open the part Chapter6Ex2 :



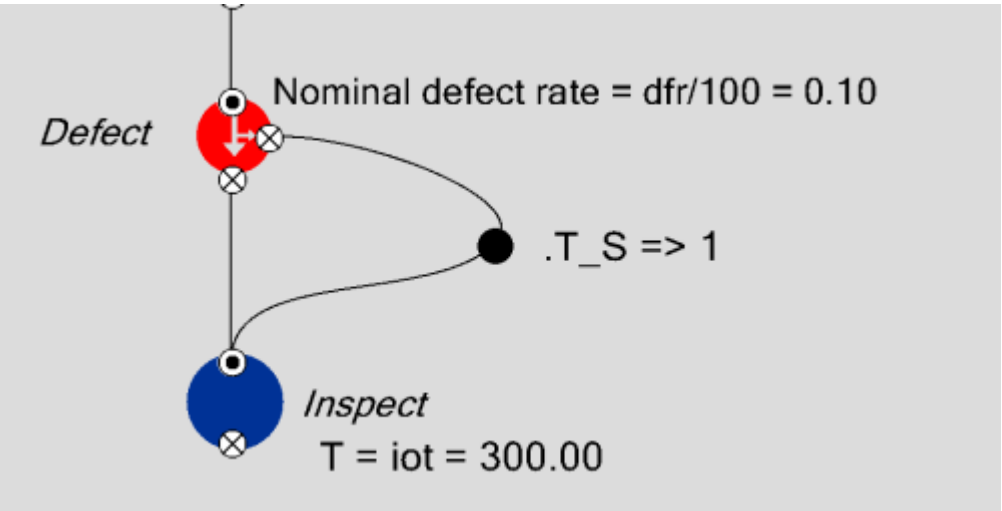
Go into the subsystem :



We are going to change the way the subsystem works a little to enable the detection of detective parts which escape to the customer. The basic approach is to separate the creation of defects from the identification of defects a little… The defects are created before inspection:

Copy the Defect block this block to the position indicated and model the following :

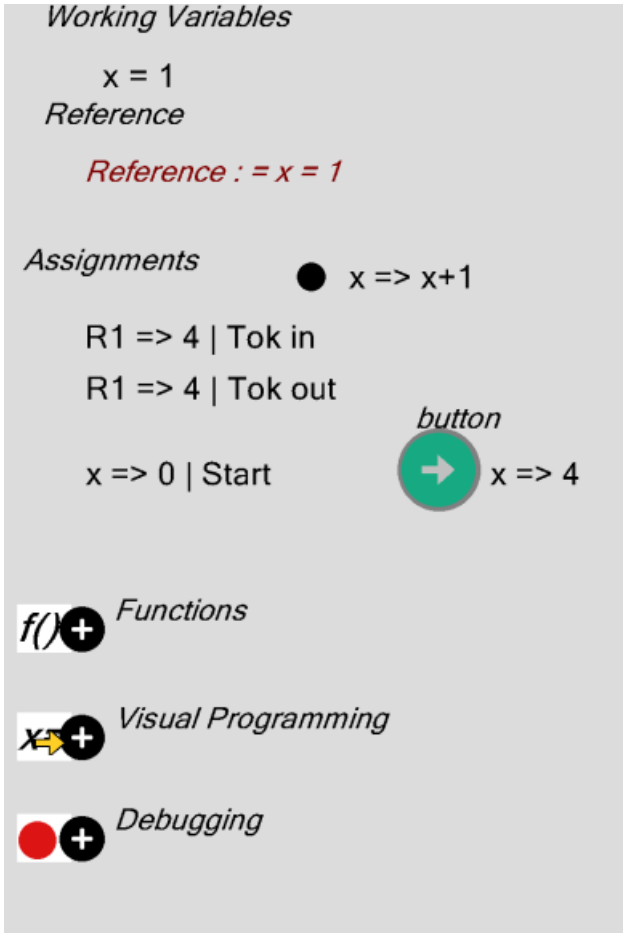
Result :



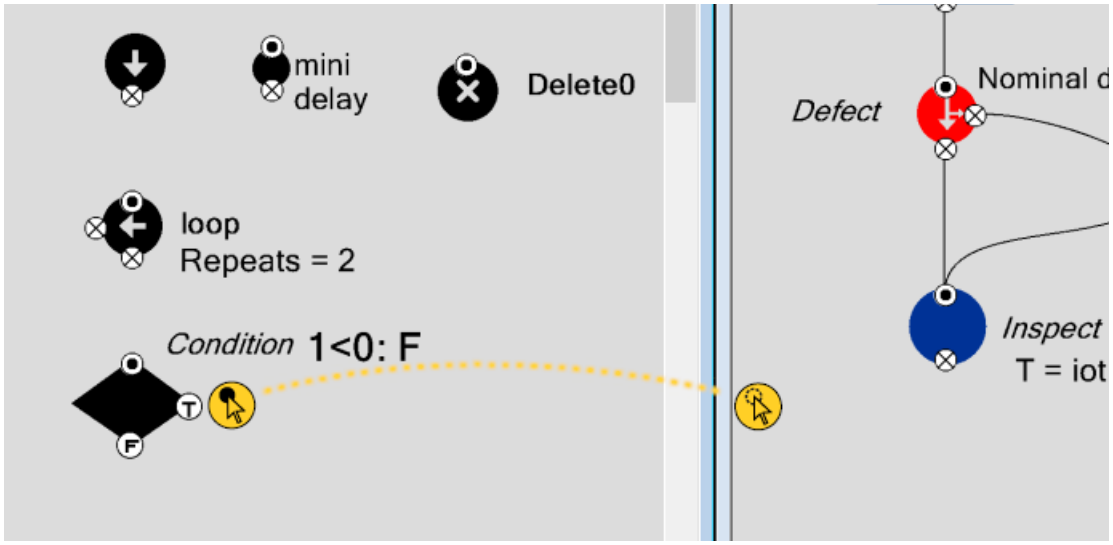
Note .T_S (the token's state) is set to 1 in the case that there is a defect.

This part of the model simply defines that a part has been created with a defect. We now have to look at the inspection system's ability to detect these defects when they occur :

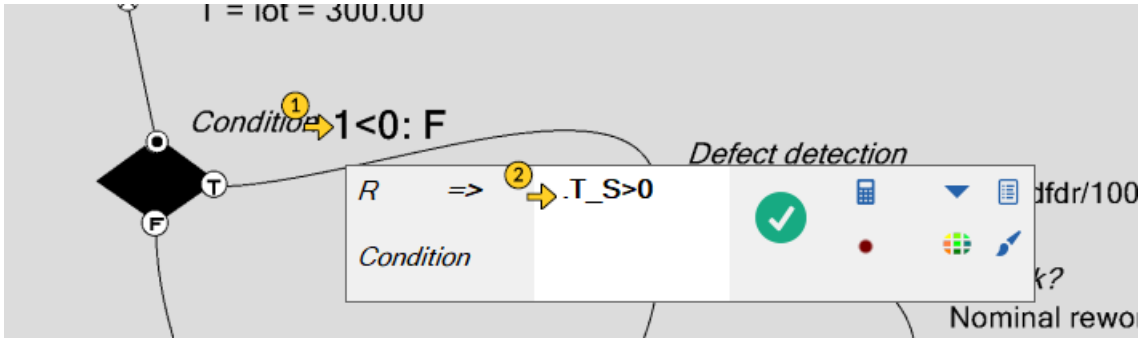
Enter the visual programming block set :



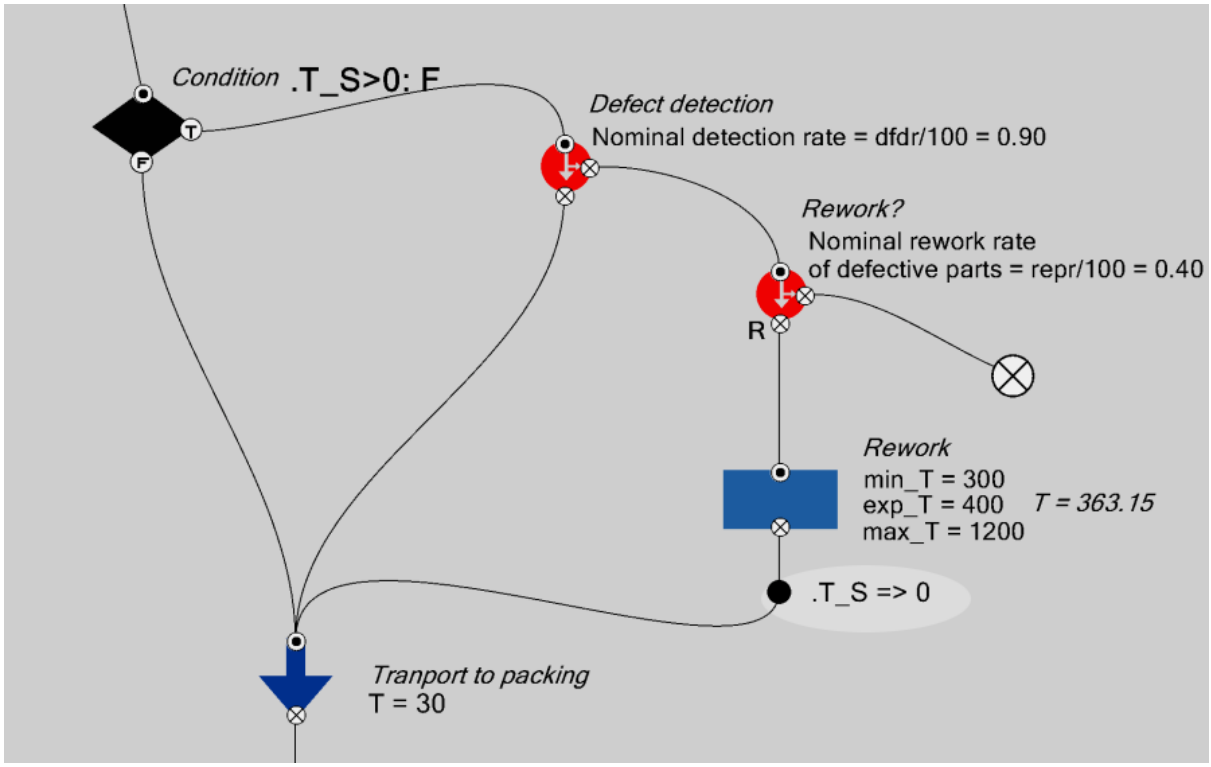
Delete the existing block and use a decision block to sort out the incoming tokens :



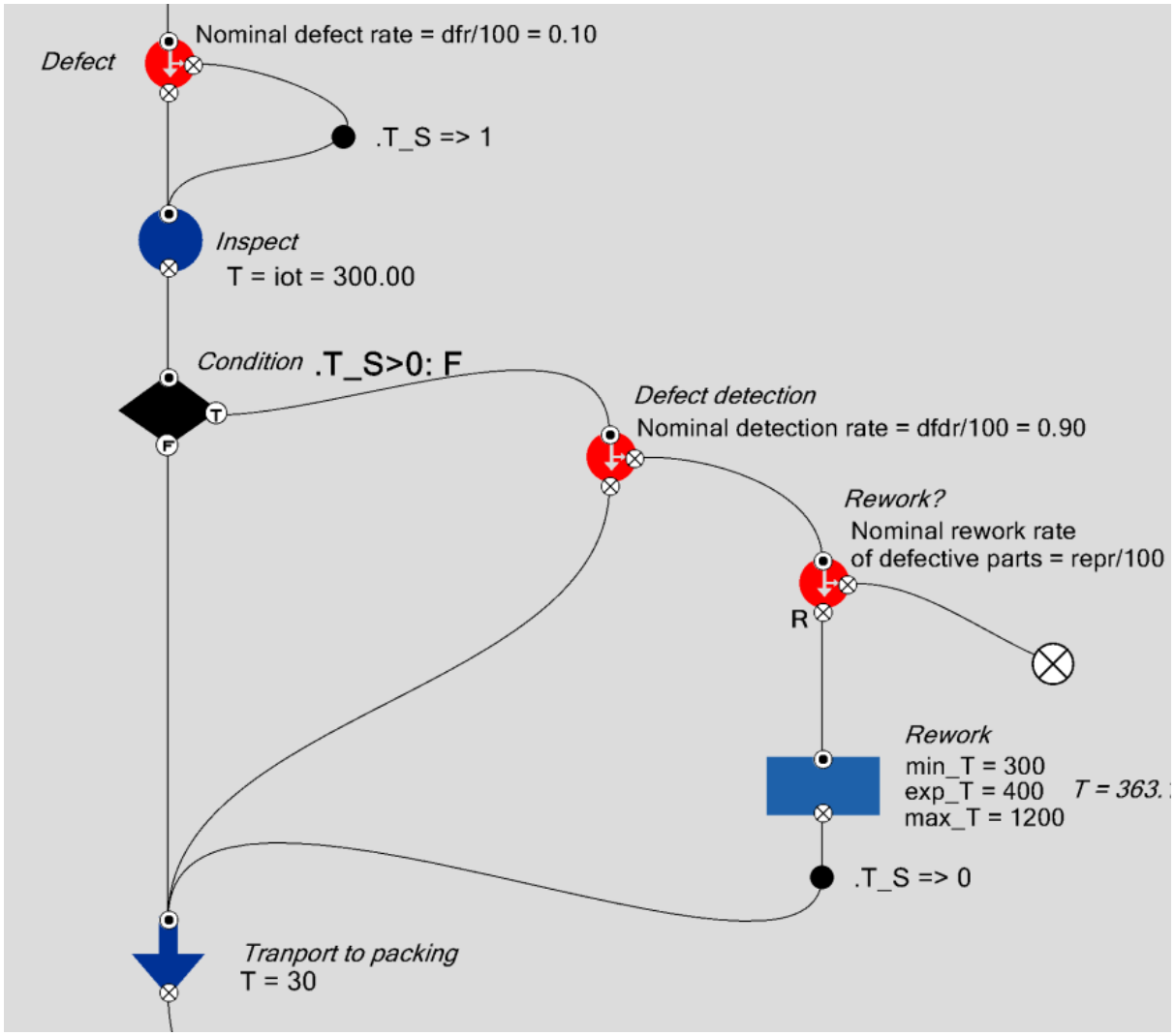
Connect it up and set it as follows :



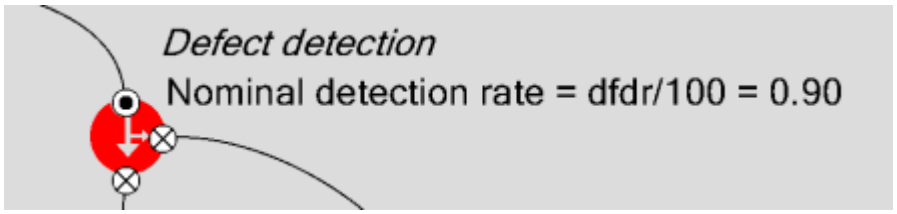
What happens when you rework a defective part. It is probably a reasonable assumption that the defect will be eliminated. The token state must be set to zero :



This is what you are aiming for :



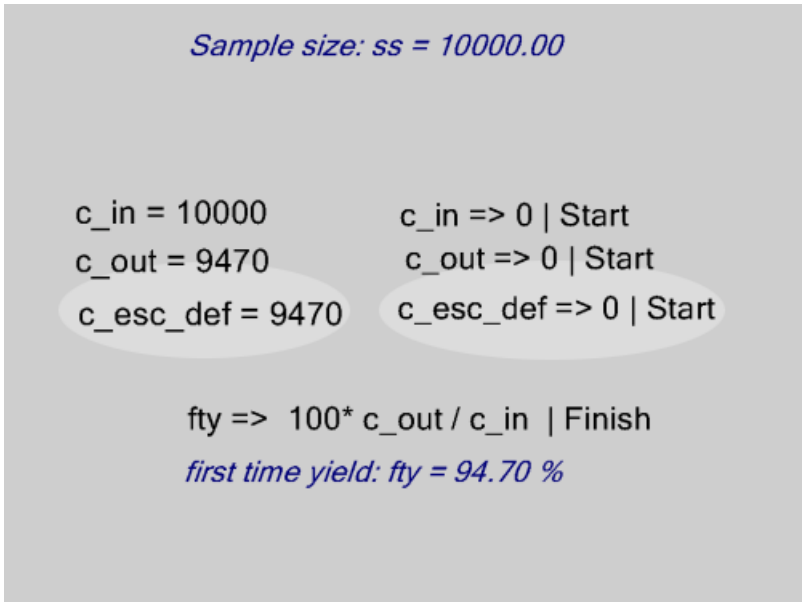
This block –



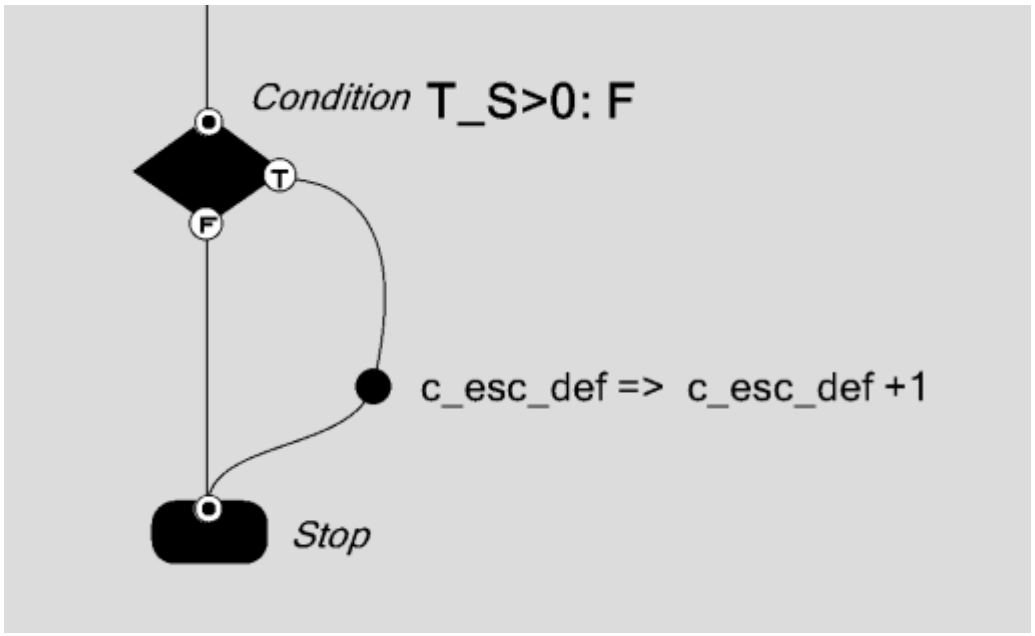
Only receives defective parts – but 10 percent of these will go into the good channel. In others words, if our subsystem is built this way - it will issue some defective parts to the customer. Like a real system with a true 10 percent defect rate but with less than perfect defect detection.

We can now go out of the subsystem and measure the defective parts that escape into the good channel:

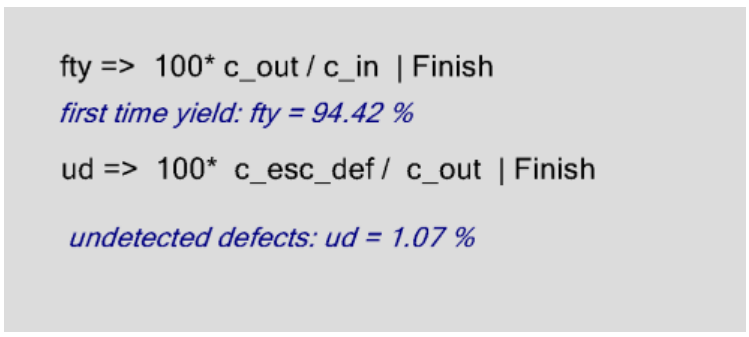
Create a counter and initialise it to 0



Create the following counter mechanism at the end of the model :



Finally compute a result at the end of the solve :



If you have struggled with this at all, refer to *Chapter6Ex2Done.inp*

A note about calculations

In this case it is possible to calculate first time yield from the probabilities set in the model. In our model, for a part to be scrapped it :

- Must be defective
- The defect must be detected
- The defect must not be repaired

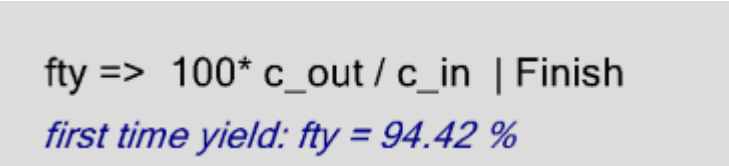
We can calculate the probability of all these occurring by multiplying the individual probabilities :

$$Probability\ of\ scrap = \ \ P(defective) * P(detection) * P(\ NOT\ repaired)$$

In this case it comes to 0.1*0.9*0.6 = 0.054.

We also know that the first time yield is 1 – the scrap rate. In this case 0.946.

This roughly corresponds with our model :



Why is the model value slightly different to the computed value. Well the model is based on a random sample of 10,000 items. The result will be different according to laws of probabilities. If you toss a coin 100 times, it is actually not likely you will get 50 heads!

The number of heads will be subject to a binomial distribution.

Lets consider our system :

If we have a sample size of 1000, we could actually predict the number of items being scrapped with a simple binomial distribution

$$Scrapped\ count = \ \ Binomial(1000,0.054)$$

So why model??

Well there are two reasons.

1. The modelling approach is not just about getting a mathematically correct result. Its about understanding the system and how it works – usually with numerous stake holders. Our defect and defect detection model is probably easier for most stakeholders to understand than the formula.

2. Things will soon get much more complex than this and the maths will get more complex.

That said, once the systems are stable, they can often be significantly simplified if we understand the mathematics of the individual elements. It is worth investing in trying to understand mathematical shortcuts for snippets of model.

Summary

This concludes our initial look At the performance of a process based on a stochastic process model. We still have some way to go. But on the way we have looked at the system as a black box and seen, again, that stochastic systems never produce quite the same results.

In Aliquis, you have been introduced to the ideas of subsystems and how to monitor systems with counters and variables. As we carry on you will find out how powerful this approach is for gaining insight from your models.

Questions :

1. Why is first time yield an important metric in manufacturing
2. Why would you think first time yield is slightly questionable as a metric in a real world scenario?
3. What does the modelling approach give you that simply collecting data in the real world does not?

Answers

1. The most expensive waste in any factory is usually scrap! First time yield and scrap are two sides of the same coin.
2. It does not take rework into account. If the process is not really working but the team is resolving the issue with extensive rework, then significant additional costs may be being added. Later we will look at another measure – Rolled throughput yield.
3. Modelling gives you the ability to conduct millions of virtual tests to make more sense of stochastic systems. It also allows you to conduct “What if” experiments to estimate the actual improvements that a proposed action might bring. On the other hand, if the model is poor, the results will be poor. Over-reliance on modelling has caused much heartache in manufacturing. The key is to understand what the model gives you and what its limitations are .