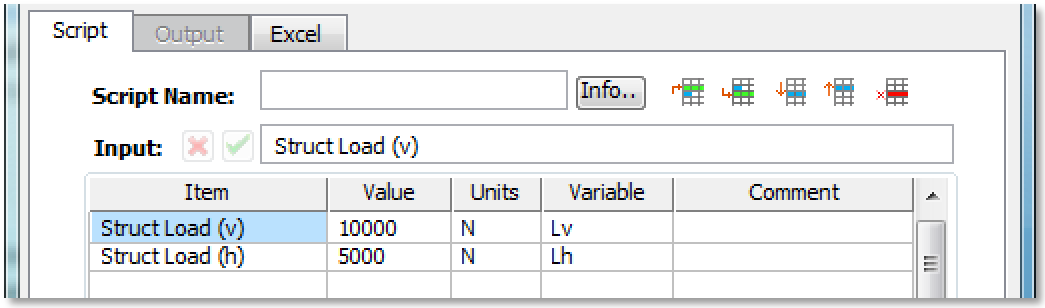
# Salad Tutorial 2 – Writing a Script

Lets say we want to write a script to extract pipe support load information for transmission to the Structural department.

Not all pipe supports need to be designed by a structural engineer however. Our example project procedure states that structural design is needed wherever

- Vertical load is > 10000 N, or
- Horizontal load is > 5000 N

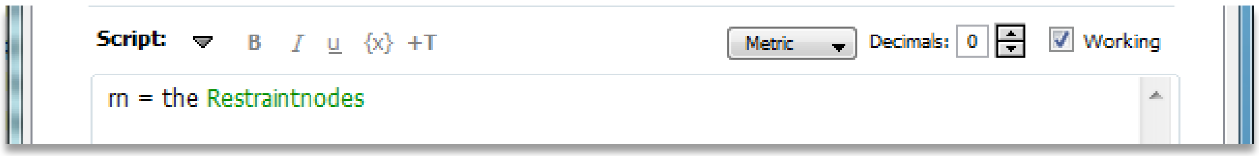
So lets go ahead and define a couple of variables in the Input Variable section, Lv and Lh, to represent these quantities. By defining these here we can quickly see and, if desired, change the figures used for the exercise.



Input: Struct Load (v)

| Item | Value | Units | Variable | Comment |
|---|---|---|---|---|
| Struct Load (v) | 10000 | N | Lv | |
| Struct Load (h) | 5000 | N | Lh | |

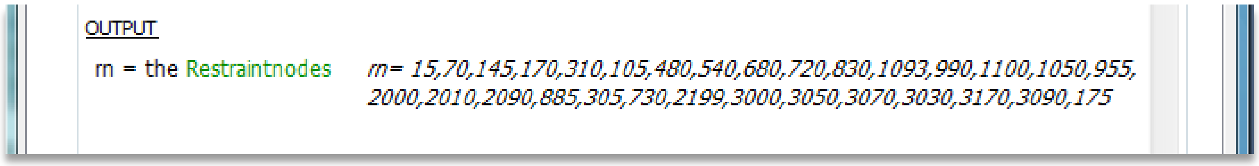
We begin the script by getting a list of pipe supports in the model, by using Salad's *restraintnodes* property. We use a new variable name 'rn' and assign the restraint nodes as below. You can assign variable names 'on the fly', ie without pre-defining them or their type before assigning a value.



Metric, Decimals: 0, ☑ Working

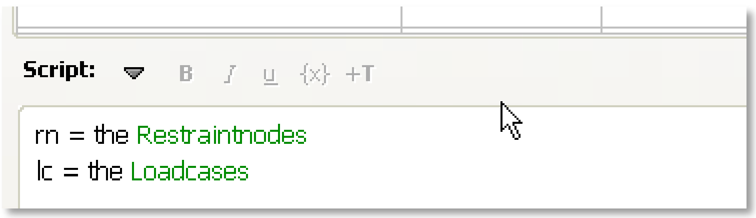
```
rn = the Restraintnodes
```

It is recommended to tick the 'Working' box when programming your script. This shows values assigned in the output screen, which is helpful in error checking and finding problems (aka debugging). Click the 'Execute' button and you will see an output similar to the following. A comma-separated list of restraint node numbers has now been assigned to variable 'rn'.



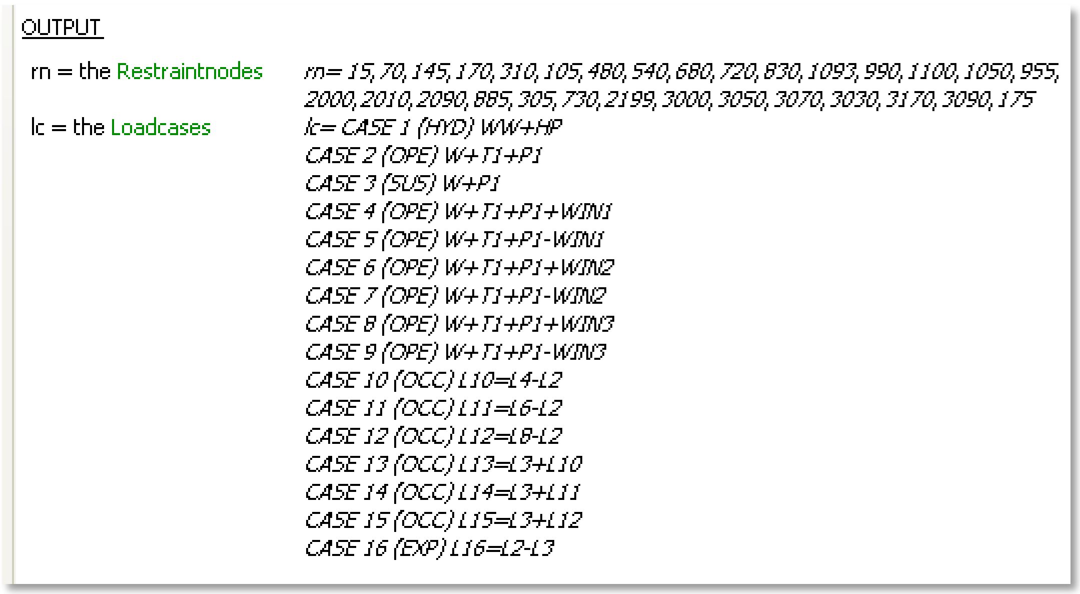
```
OUTPUT
rn = the Restraintnodes     rn= 15,70,145,170,310,105,480,540,680,720,830,1093,990,1100,1050,955,
                            2000,2010,2090,885,305,730,2199,3000,3050,3070,3030,3170,3090,175
```

Now lets go back to the script and add a line to get the load cases –



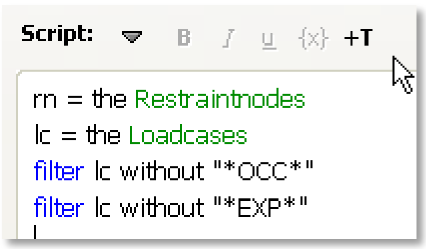
```
rn = the Restraintnodes
lc = the Loadcases
```

Executing gives the following –



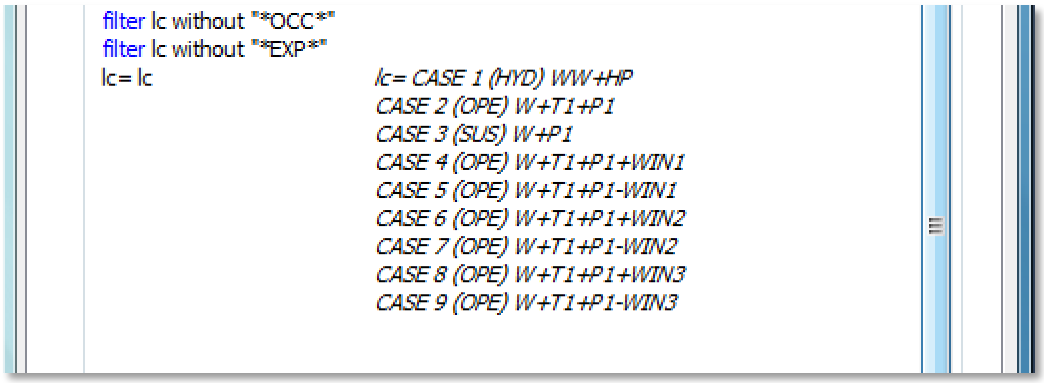
```
OUTPUT
 rn = the Restraintnodes    rn= 15, 70, 145, 170, 310, 105, 480, 540, 680, 720, 830, 1093, 990, 1100, 1050, 955,
                            2000, 2010, 2090, 885, 305, 730, 2199, 3000, 3050, 3070, 3030, 3170, 3090, 175
 lc = the Loadcases         lc= CASE 1 (HYD) WW+HP
                            CASE 2 (OPE) W+T1+P1
                            CASE 3 (SUS) W+P1
                            CASE 4 (OPE) W+T1+P1+WIN1
                            CASE 5 (OPE) W+T1+P1-WIN1
                            CASE 6 (OPE) W+T1+P1+WIN2
                            CASE 7 (OPE) W+T1+P1-WIN2
                            CASE 8 (OPE) W+T1+P1+WIN3
                            CASE 9 (OPE) W+T1+P1-WIN3
                            CASE 10 (OCC) L10=L4-L2
                            CASE 11 (OCC) L11=L6-L2
                            CASE 12 (OCC) L12=L8-L2
                            CASE 13 (OCC) L13=L3+L10
                            CASE 14 (OCC) L14=L3+L11
                            CASE 15 (OCC) L15=L3+L12
                            CASE 16 (EXP) L16=L2-L3
```

Variable 'lc' now contains text consisting of the 16 lines shown. Looking at this output however brings to mind the fact that we don't want to consider certain load cases. Lets say we want to ignore 'OCC' and 'EXP' load cases. We can add a couple of lines to filter these out quickly:



```
rn = the Restraintnodes
lc = the Loadcases
filter lc without "*OCC*"
filter lc without "*EXP*"
```

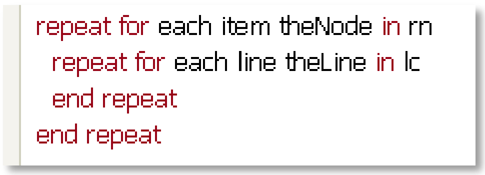
The *filter* command doesn't trigger a re-display of the filtered variable. However we can check it quickly by adding a line to the script: 'lc= lc'. The output is as follows, you will note that 'OCC' and 'EXP' cases have been removed.



```
filter lc without "*OCC*"
filter lc without "*EXP*"
lc= lc                          lc= CASE 1 (HYD) WW+HP
                                CASE 2 (OPE) W+T1+P1
                                CASE 3 (SUS) W+P1
                                CASE 4 (OPE) W+T1+P1+WIN1
                                CASE 5 (OPE) W+T1+P1-WIN1
                                CASE 6 (OPE) W+T1+P1+WIN2
                                CASE 7 (OPE) W+T1+P1-WIN2
                                CASE 8 (OPE) W+T1+P1+WIN3
                                CASE 9 (OPE) W+T1+P1-WIN3
```

The *filter* command is a Revolution-native command. You can find more about Revolution terms from the Help menu under ‘Language Reference’. (See also the futher note at end of this tutorial).

Now lets loop through all the restraints and the load cases to look for restraints which qualify as structural supports. There are a number of ways to initiate a repeat loop, but perhaps the easiest is the *repeat for each* form. Recalling that the restraint nodes are stored as a comma-separated list in variable ‘rn’, we can make use of the concept of *items*. Items are simply chunks of text or numbers in a list which are separated by a common character known as the *itemdelimiter*, in this case a comma. The comma is the default character for the *itemdelimiter*. Therefore we use the *repeat for each item* form for restraint nodes, and the *repeat for each line* form for load cases:

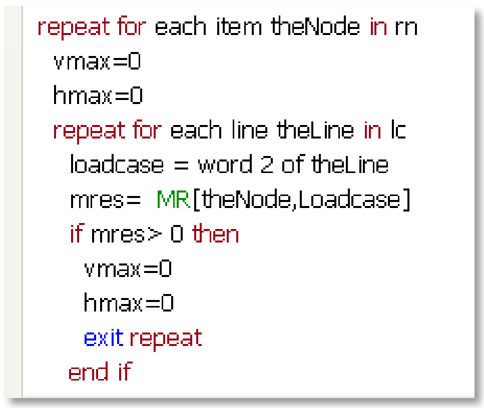


```
repeat for each item theNode in rn
  repeat for each line theLine in lc
  end repeat
end repeat
```

This is a preliminary ‘skeleton’ showing how the repeat loops are constructed – at this stage no useful output is created. The outer repeat loop goes sequentially through each restraint node in the list variable ‘rn’ and assigns the node number to the variable ‘theNode’. The inner loop goes through each line of the loadcase list ‘lc’ and assigns the line to the variable ‘theLine’.

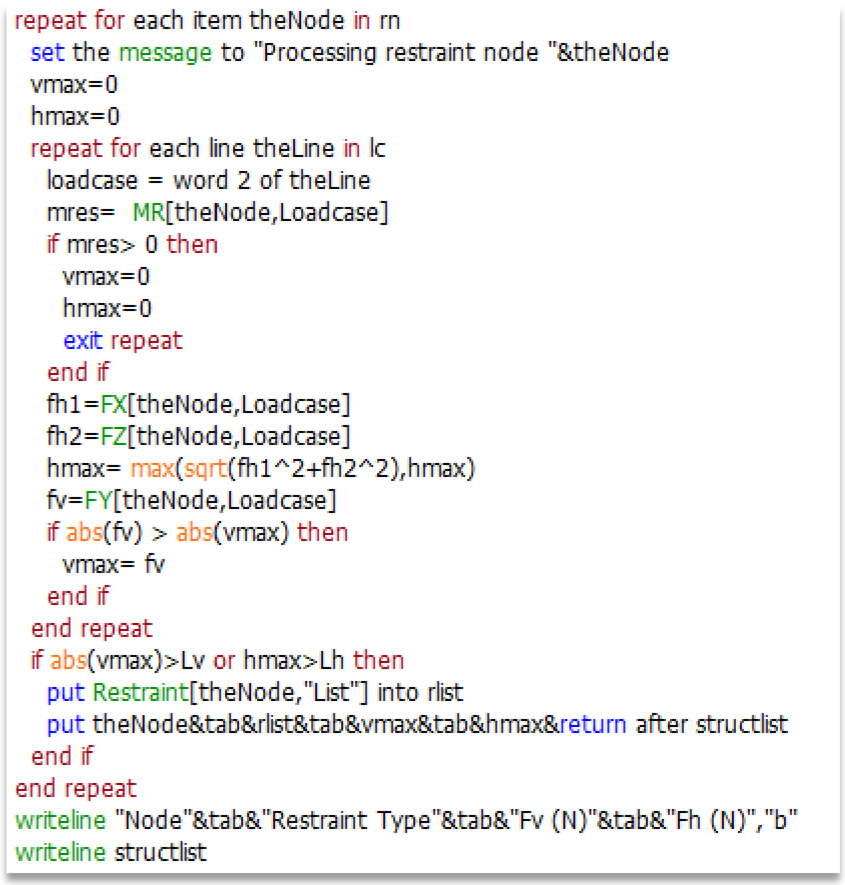
Now lets add script to extract the data. It will be assumed that we are not interested in supports which have moment restraint, these are normally internal to the piping system or imposed on equipment.

We will use variables ‘vmax’ and ‘hmax’ to track the maximum load for each support. These need to be reset to zero for each restraint. We then get the load case number for each load case, recalling that this is the second word in the load case description, eg “case 6 (OPE) W + T1 + P1 + WIN2”. Then we use the *MR[node,loadcase]* data variable to check for bending moments and exit the loop if they are found.



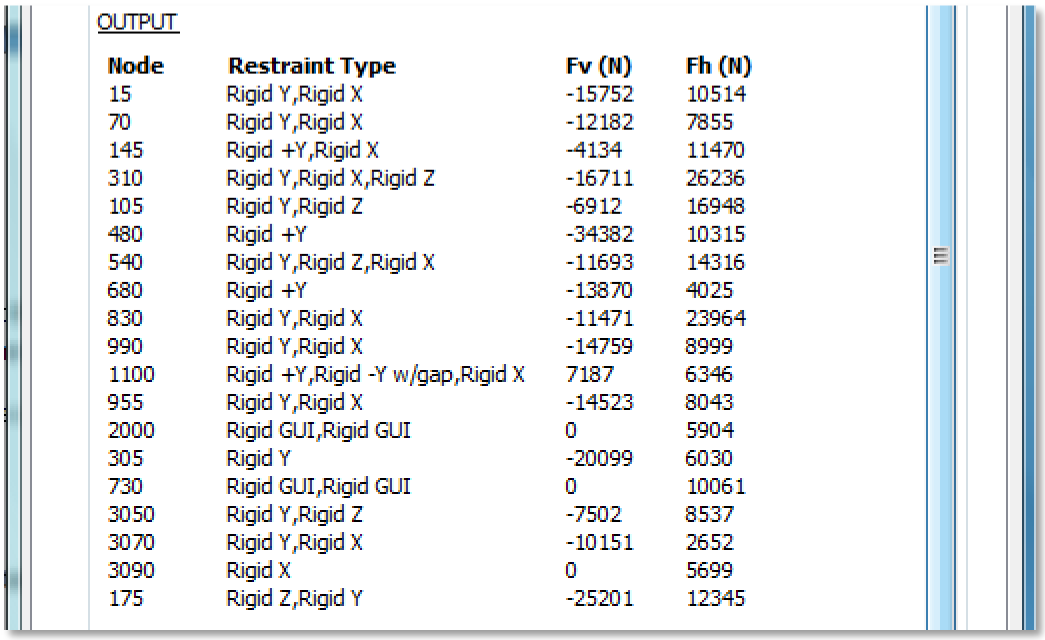
```
repeat for each item theNode in rn
  vmax=0
  hmax=0
  repeat for each line theLine in lc
    loadcase = word 2 of theLine
    mres=  MR[theNode,Loadcase]
    if mres> 0 then
      vmax=0
      hmax=0
      exit repeat
    end if
```

We complete the first draft script as follows, getting the maximum horizontal and vertical loads for each support and then comparing to the structural support load levels ‘Lv’ and ‘Lh’.   A list of structural supports is created with tabs separating column data.  It is also recommended to untick the ‘Working’ box at this stage to prevent slow execution with the large number of repeat loops taking place.



```
repeat for each item theNode in rn
  set the message to "Processing restraint node "&theNode
  vmax=0
  hmax=0
  repeat for each line theLine in lc
    loadcase = word 2 of theLine
    mres=  MR[theNode,Loadcase]
    if mres> 0 then
      vmax=0
      hmax=0
      exit repeat
    end if
    fh1=FX[theNode,Loadcase]
    fh2=FZ[theNode,Loadcase]
    hmax= max(sqrt(fh1^2+fh2^2),hmax)
    fv=FY[theNode,Loadcase]
    if abs(fv) > abs(vmax) then
      vmax= fv
    end if
  end repeat
  if abs(vmax)>Lv or hmax>Lh then
    put Restraint[theNode,"List"] into rlist
    put theNode&tab&rlist&tab&vmax&tab&hmax&return after structlist
  end if
end repeat
writeline "Node"&tab&"Restraint Type"&tab&"Fv (N)"&tab&"Fh (N)","b"
writeline structlist
```

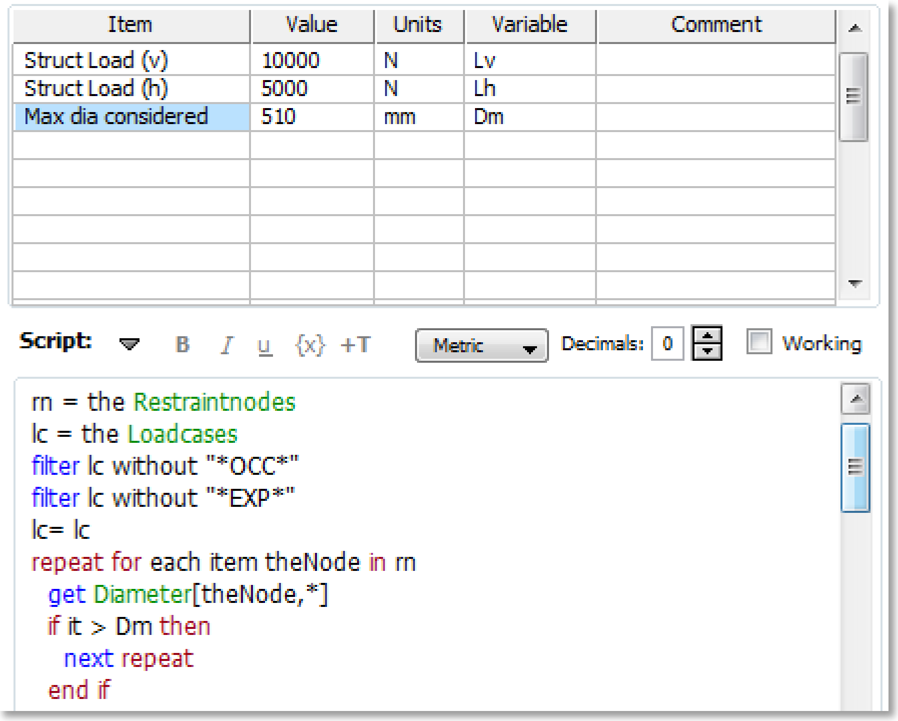
The output is similar to the following:



OUTPUT

| Node | Restraint Type | Fv (N) | Fh (N) |
|---|---|---|---|
| 15 | Rigid Y,Rigid X | -15752 | 10514 |
| 70 | Rigid Y,Rigid X | -12182 | 7855 |
| 145 | Rigid +Y,Rigid X | -4134 | 11470 |
| 310 | Rigid Y,Rigid X,Rigid Z | -16711 | 26236 |
| 105 | Rigid Y,Rigid Z | -6912 | 16948 |
| 480 | Rigid +Y | -34382 | 10315 |
| 540 | Rigid Y,Rigid Z,Rigid X | -11693 | 14316 |
| 680 | Rigid +Y | -13870 | 4025 |
| 830 | Rigid Y,Rigid X | -11471 | 23964 |
| 990 | Rigid Y,Rigid X | -14759 | 8999 |
| 1100 | Rigid +Y,Rigid -Y w/gap,Rigid X | 7187 | 6346 |
| 955 | Rigid Y,Rigid X | -14523 | 8043 |
| 2000 | Rigid GUI,Rigid GUI | 0 | 5904 |
| 305 | Rigid Y | -20099 | 6030 |
| 730 | Rigid GUI,Rigid GUI | 0 | 10061 |
| 3050 | Rigid Y,Rigid Z | -7502 | 8537 |
| 3070 | Rigid Y,Rigid X | -10151 | 2652 |
| 3090 | Rigid X | 0 | 5699 |
| 175 | Rigid Z,Rigid Y | -25201 | 12345 |

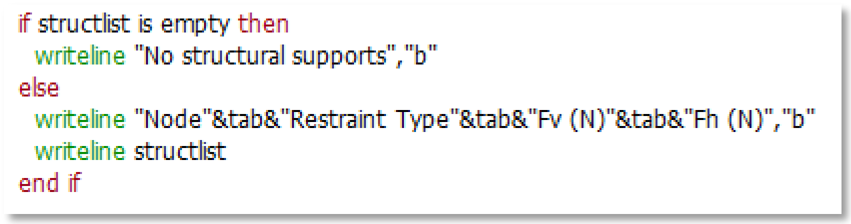
Also it now occurs to us that we might be including in the list some restraints that are part of vessels or other equipment. Let's filter those out by limiting the maximum diameter of piping considered. We add another Input Variable 'Dm' to set this quantity. We add a *get* statement in the main repeat loop to check the diameter. If it exceeds 'Dm' we go on to the next restraint. The *get* statement can be used when we don't want to keep a value for long – it sets the value of a special reserved variable named *it*.



| Item | Value | Units | Variable | Comment |
|---|---|---|---|---|
| Struct Load (v) | 10000 | N | Lv | |
| Struct Load (h) | 5000 | N | Lh | |
| Max dia considered | 510 | mm | Dm | |

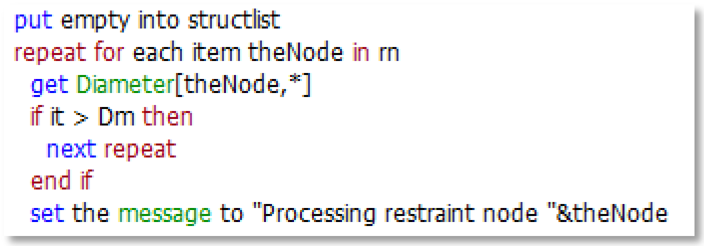
```
rn = the Restraintnodes
lc = the Loadcases
filter lc without "*OCC*"
filter lc without "*EXP*"
lc= lc
repeat for each item theNode in rn
  get Diameter[theNode,*]
  if it > Dm then
    next repeat
  end if
```

Thinking of another possible outcome, we check for the condition where there are no structural supports.  At the moment the statement ‘writeline structlist’ would just cause Salad to output the word ‘structlist’ if no value had been assigned to that variable.  So we output an alternative message if structlist is empty at the end.



```
if structlist is empty then
  writeline "No structural supports","b"
else
  writeline "Node"&tab&"Restraint Type"&tab&"Fv (N)"&tab&"Fh (N)","b"
  writeline structlist
end if
```

We also need to set structlist to *empty* before the first repeat loop, otherwise it would be considered undefined rather than empty in the case of no structural supports.  Finally, we add a message to show progress, which can be desireable for larger models:



```
put empty into structlist
repeat for each item theNode in rn
  get Diameter[theNode,*]
  if it > Dm then
    next repeat
  end if
  set the message to "Processing restraint node "&theNode
```

**Further Improvements**

The script currently  gives a list of all restraints qualifying as structural supports for the model.  The script could be further refined and improved, for instance:

- to list Fx and Fz for each support
- To extract supports from more than one file (hint - use the *Datafiles* function, another repeat loop and the *Datafile* property)

**Note regarding the Language Reference (found under Help-> Language Reference):**

The reference material has been extracted directly from Revolution’s help documentation.  It refers to a number of objects which are ‘building blocks’ of the Revolution programming environment, including *stacks, cards, groups, buttons, fields* and *files*.  These are not relevant to the use of Salad.