Using Loop module in BEAMER

This application note details the usage of the Loop module. The loop module in BEAMER is like any other 'LOOP' operation used in other programming languages where a certain variable is defined and iterated based on given conditions. In BEAMER, the Loop module works seamlessly with other process modules allowing for iterative modification of layout or process parameters.

INTRODUCTION

Many devices require multiple fabrication iterations before getting the optimal conditions for a final product. This can be time and cost consuming. The simulation of such devices gives the possibility to modify a layout providing relevant information in a fraction of time. Particularly, the loops give a simple way to introduce small changes in every layout iteration. BEAMER provides a 'loop module' which can be used with other modules, and results in a much faster and wider range of testing conditions for a single layout. In this document we show all the alternatives that the loop module provides.

THE LOOP MODULE GENERALITIES

The *Loop* module can be found within the control tab. It can be put into action by left clicking the mouse and dragging it into the flow area or by double clicking on the loop module. This opens a loop-dialogue window where the 'Variables', 'Advanced' and 'Comment' tabs are modified to set the module.

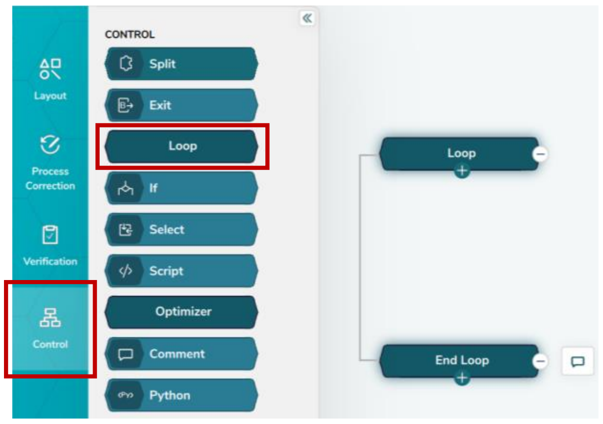


Figure 1. The Loop module is found in the control tab in BEAMER.

1. VARIABLES

The conditions that establish the iterative process of the loop are typed within the 'Variables' tab. The variable name is defined with the format %VarName%. There are three *Loop modes* available for variable definition:

- *Generic Loop*: The user creates a list either by direct tipping into the table (alphanumerical) or by using a wizard).
- *Loop Over Layer*: The list for the loop is obtained from the list of layers in the layout, i.e., number of iterations in the loop is the same as the number of layers.
- *Loop Over Cell*: The list for the loop is selected from the cells forming the layout.

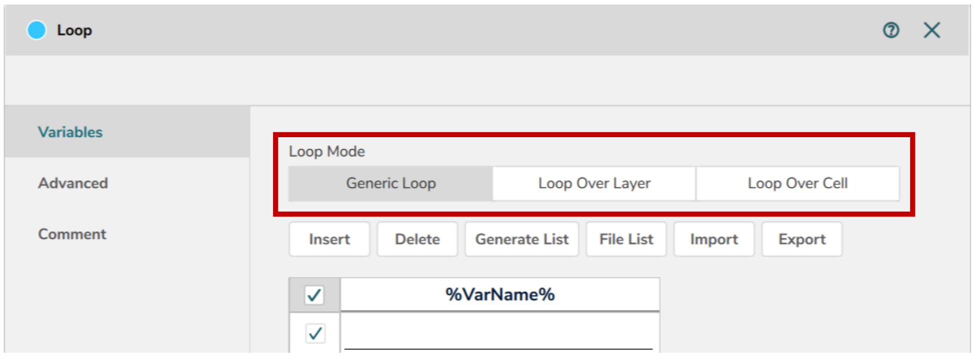


Figure 2. Exist three modes to set the loop cycle.

a. *Generic loop:* When the *Generic Loop Mode* is chosen, there are four alternatives to define the variables: manual value entering one by one, 'Generate List', 'File List' and 'Import'. The *Generate List* dialogue allows to introduce a start value and end value of a list as well as the step. Additionally, the user can create its list by exporting it to a text file (this can also be used as a template for further reference).

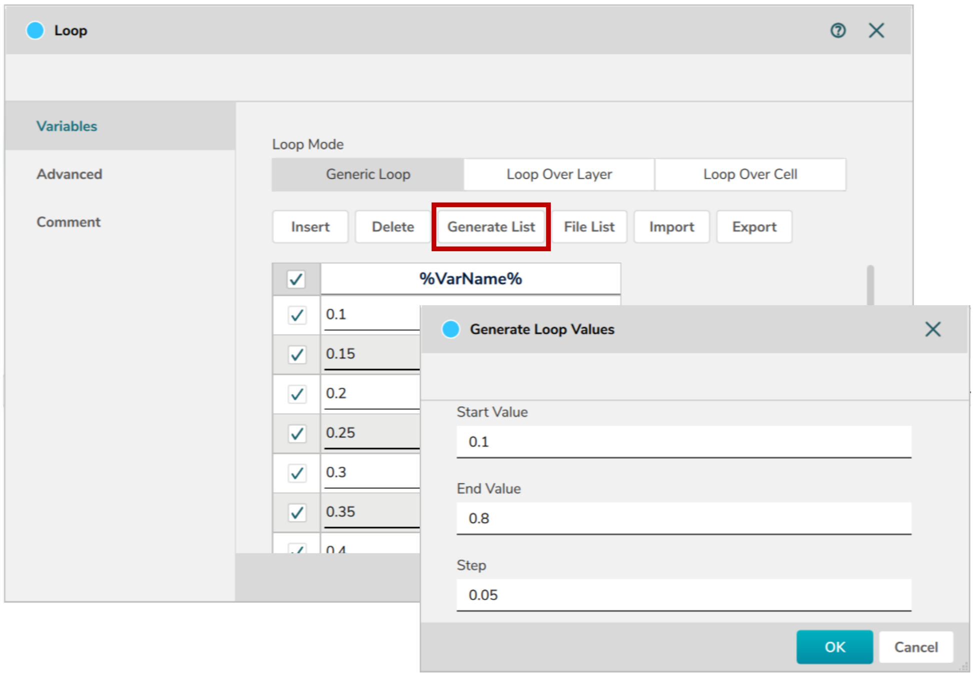
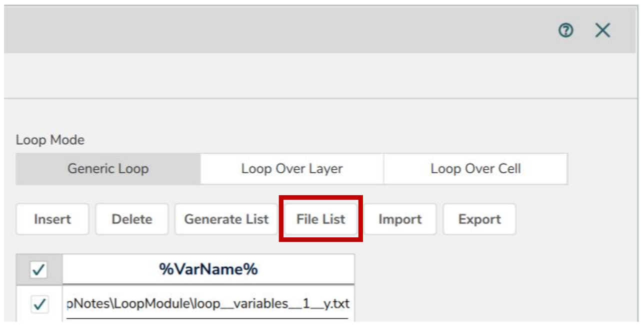


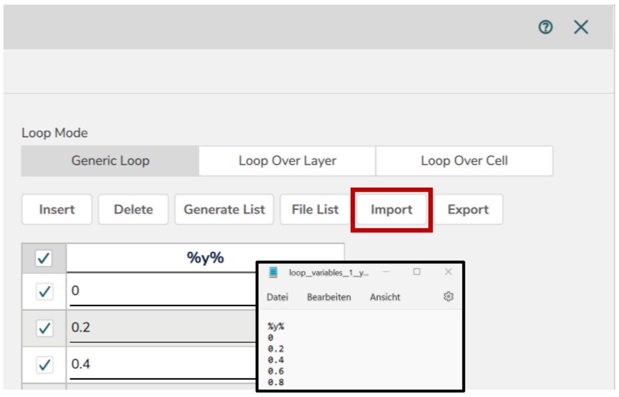
Figure 3. Numeric list sorting using 'Generate List' option.

The *File List* and *Import* buttons either creates a list of files or loads an external file that contains a list, respectively. In the *File List* option, the user needs to change the name of the variable (%VarName%), while in *Import* the header of the file passes as the variable name. Importantly, the symbol '%' is part of BEAMER format and must remain regardless of the name selection.



✓	%VarName%
✓	pNotes\LoopModule\loop__variables__1__y.txt

Figure 4. Filling the loop using files.

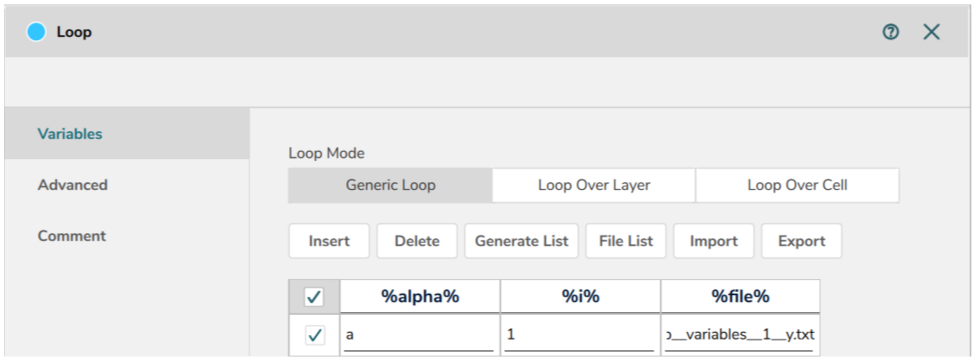


✓	%y%
✓	0
✓	0.2
✓	0.4

```
%y%
0
0.2
0.4
0.6
0.8
```

Figure 5. Importing a file that contains a numeric list.

Additionally, it is possible to create multiple variables, each of them with their own typed, generated, or imported list.

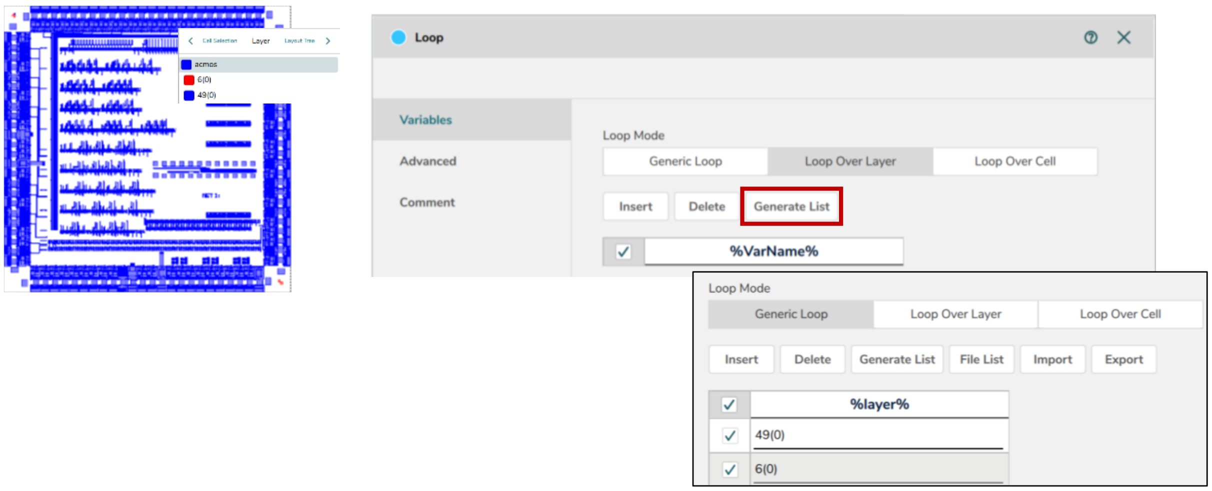


✓	%alpha%	%i%	%file%
✓	a	1	›__variables__1__y.txt

Figure 6. Different types of variables used within the Loop module.

b. *Loop Over Layer:* The next Loop Mode is the 'Loop Over Layer'. This option only enables the *Generate List* button. When the *Generate List* button is pressed, it reads the number of layers available in the layout and generates a list with these layers. Afterwards, the mode is automatically changed to a *Generic Loop Mode* where the other options are enabled once more. For instance, in the image is shown a design that contains 2 layers (6(0) and 49(0)). When the loop mode is *Loop Over Layer*, those two layers are used to generate a list.

a) b)



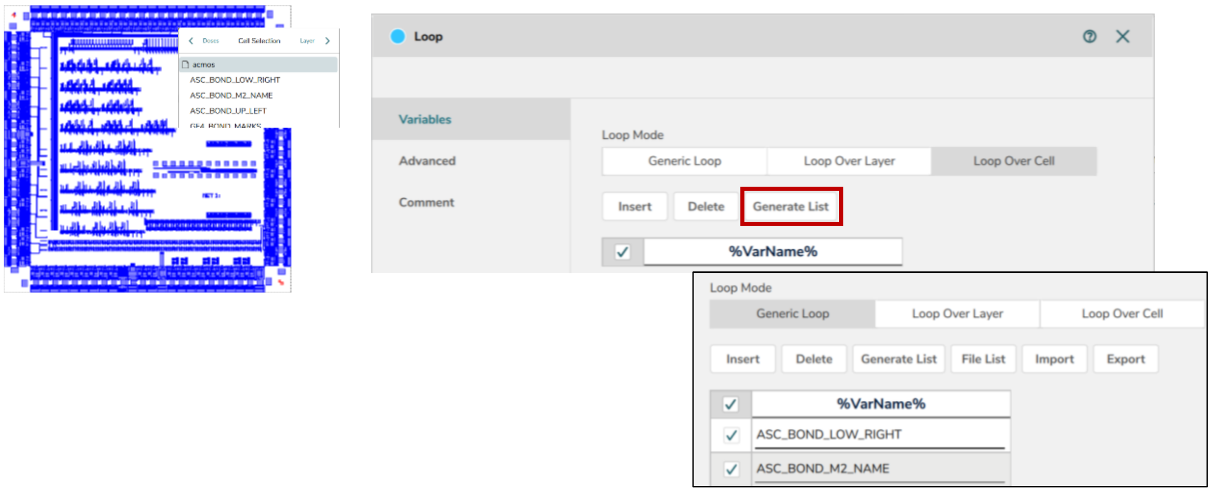
%VarName%

%layer%
49(0)
6(0)

Figure 7. Layout with two layers used to create the loop variables.

c. Loop Over Cell: The 'Loop Over Cell' takes the cells that form the layout to generate a list. In the figure, the layout contains many cells (only four cells are shown). When using the button *Loop Over Cell* followed by the *Generate List* button, all the cells are used to generate a list. Importantly, less cells can be used in the list if the extract module is used before the Loop Module to take only the cells of interest (see App Note for Extract Module for more details).

a) b)



%VarName%

%VarName%
ASC_BOND_LOW_RIGHT
ASC_BOND_M2_NAME

Figure 8. Layout with many cells used to create the loop variables.

Finally, the *Insert* and *Delete* buttons, which are available to the three *Loop Modes* mentioned before, are used to further modify an existing list by adding or removing values, respectively. Moreover, user can choose to select/unselect a value by ticking/unticking the square box in front of each column.

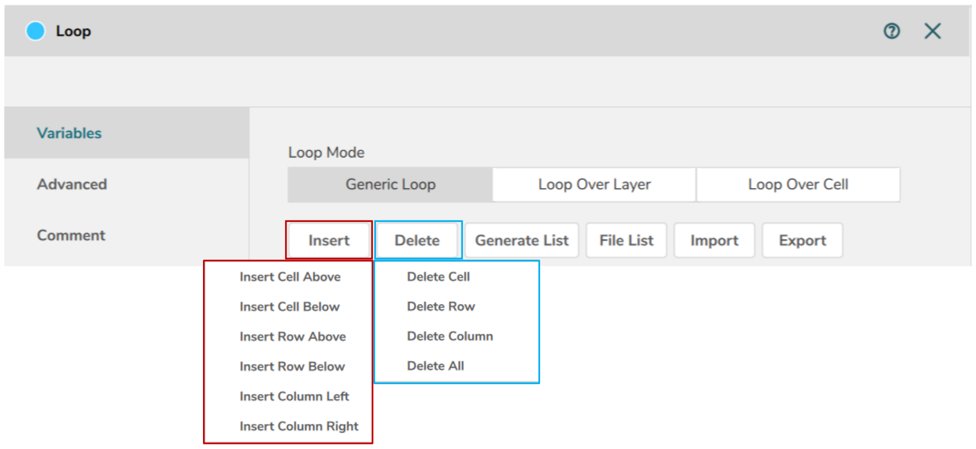
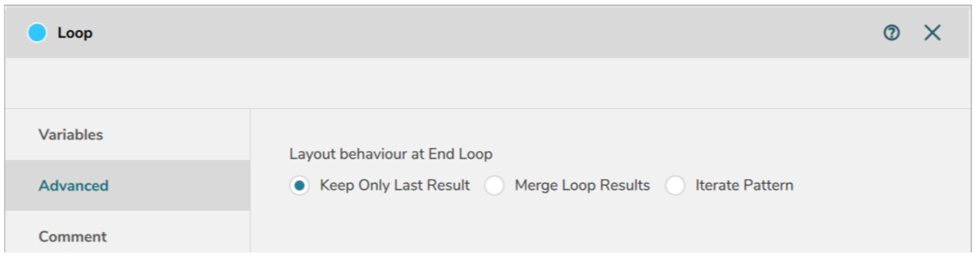


Figure 9. Insert and Delete buttons with their respective options.

2. ADVANCED

In the 'Advanced' tab, the user decides over the Loop Module results. Here, three options are given:



Layout behaviour at End Loop: [✓ Keep Only Last Result] Merge Loop Results Iterate Pattern

Figure 10. Loop module behaviour at the end of the cycle.

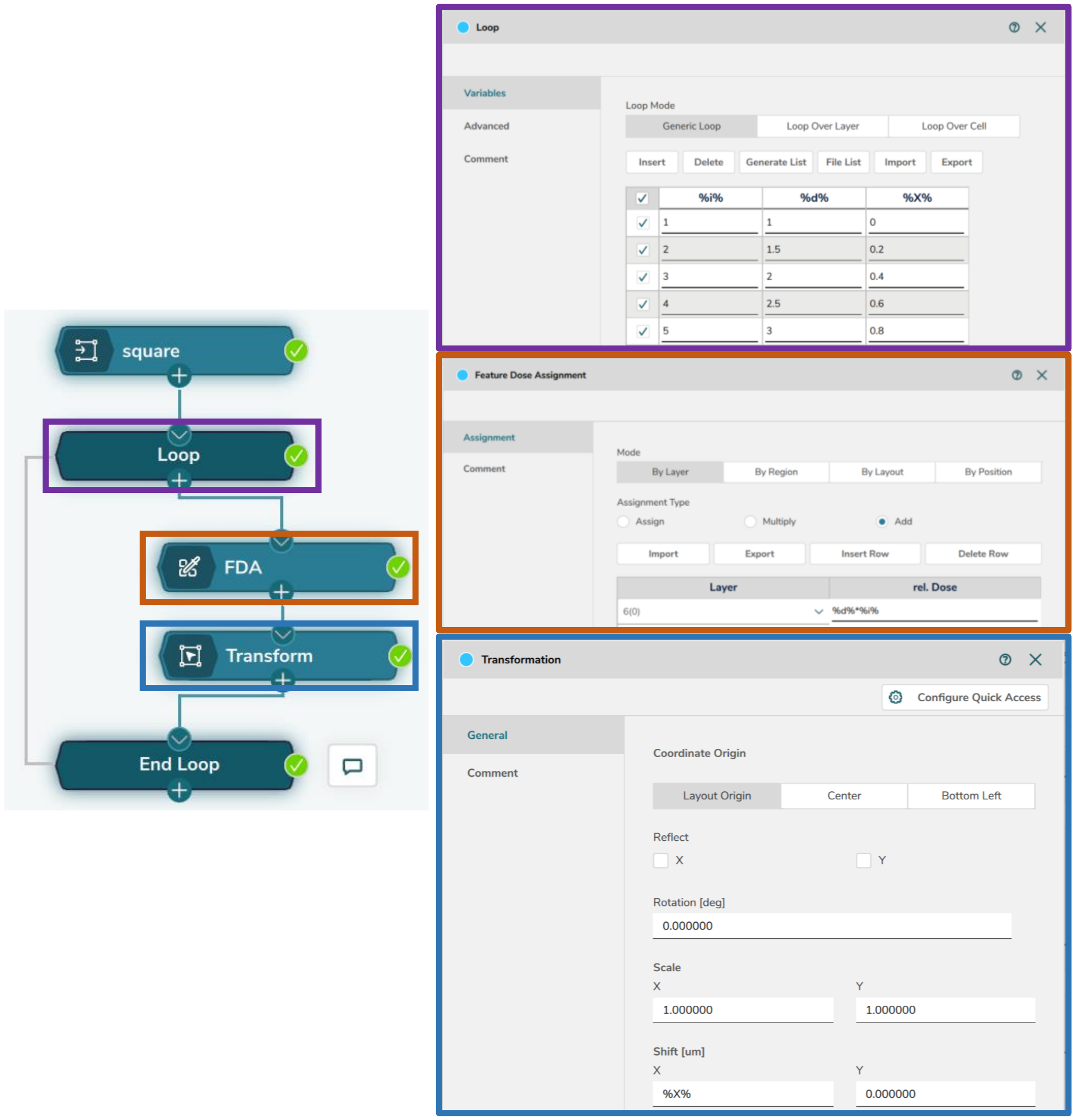
- *Keep Only Last Result*: Regardless of the number of iterations, this option only displays the result of the final iteration. The input of every iteration in the loop is the original layout.
- *Merge Loop Results*: The result of every iteration is stored. At the end of the loop, all the results are displayed in the viewer. The input of every iteration in the input of the loop is the original layout.
- *Iterate Pattern*: The result of an iteration is used as an input layout for the following iteration. At the end of the loop, it is displayed the last result but modified for every iteration.

The usage of the tab 'Advanced' will be clarified with the following example that shows the function of the Loop Module

Loop Module example

The following image exhibits a *Loop Module* with the variables %i%, %d% and %X%. These variables are used by a module to modify the dose of the layout (*FDA*) and by a module to modify the position of the layout (*Transform*). The flow image needs to be updated and enlarge the image for each module so that it is easy to read them.

The *FDA* module operates over the module variables (%i% * %d%) to change the dose of the layout in every iteration. The *Transform* module uses the variable %X% to set a new position of the layout. When the flow is run, the loop module assigns the values of the first row to each of the variables. When it finishes the first cycle, for the second iteration it assigns the values of the second row, and so on. Importantly, the modules can run independently to the *Loop* module; therefore, before running the full flow, it is recommended to run each module separately ensuring the variables assignment works.



Loop

Variables
Advanced
Comment

Loop Mode: [✓ Generic Loop] Loop Over Layer | Loop Over Cell

Insert | Delete | Generate List | File List | Import | Export

✓	%i%	%d%	%X%
✓	1	1	0
✓	2	1.5	0.2
✓	3	2	0.4
✓	4	2.5	0.6
✓	5	3	0.8

Feature Dose Assignment

Assignment
Comment

Mode: [✓ By Layer] By Region | By Layout | By Position

Assignment Type: Assign | Multiply | [✓ Add]

Import | Export | Insert Row | Delete Row

Layer	rel. Dose
6(0)	%d%*%i%

Transformation

Configure Quick Access

General
Comment

Coordinate Origin: [✓ Layout Origin] Center | Bottom Left

Reflect: X ☐ Y ☐

Rotation [deg]: 0.000000

Scale: X 1.000000 Y 1.000000

Shift [um]: X %X% Y 0.000000

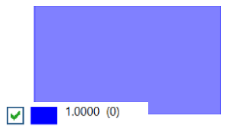
Figure 11. Exemplary flow using the Loop, dose assignment and transform modules.

Accordingly, the following image exhibits the result for every option in the tab 'Advanced'. For an input layout consisting of a single rectangle of dose 1 (see Fig 12a):

- In the *Keep Only Last Result*, the cycle runs over all the iterations and only displays the last result of the loop. In this case %d% = 3 and %i% = 5; therefore, the final dose of the loop is 15 and then is added to the input layout. The result is a single rectangle with dose of 16 (see Fig 12b.1).

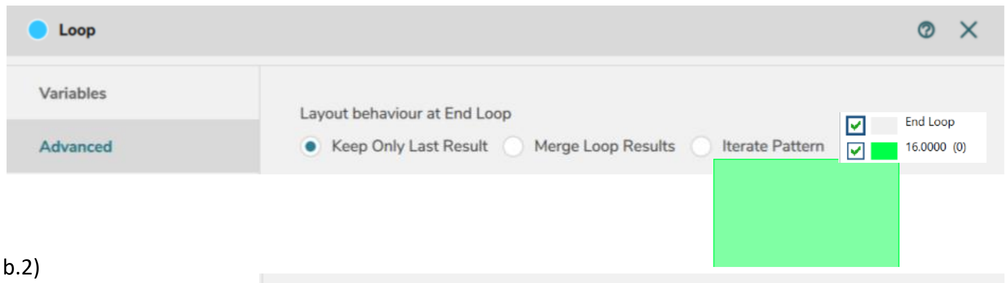
- In the *Merge Loop Results*, the loop stores the results of every iteration. For the first iteration the dose assignment is %i% * %d% = 1 * 1 = 1 and added to the input layout, resulting in a final dose of 2. This process continues and for the last iteration the dose assignment is %i% * %d% = 5 * 3 = 15 and added to the input layout, resulting in a final dose of 16. Finally, each of these layouts are placed according to the variable %X%. The result of the loop are 5 rectangles with doses from 2 to 16 placed one next to each other (see Fig 12b.2).
- In the *Iterate Pattern*, the input layout is changed in every iteration. The rectangle of dose 1 is the input of the first cycle, where its dose is modified to 2. For the second iteration, the input layout is a rectangle of dose 2, the new dose assignment %i% * %d% = 2 * 1.5 = 3 modifies the dose to 5. The rectangle of dose 5 becomes the new loop input. This process is repeated as such as the result of the loop is a single rectangle with dose of 36 (see Fig 12b.3)

a)



✓ 🟦 1.0000 (0)

b.1)



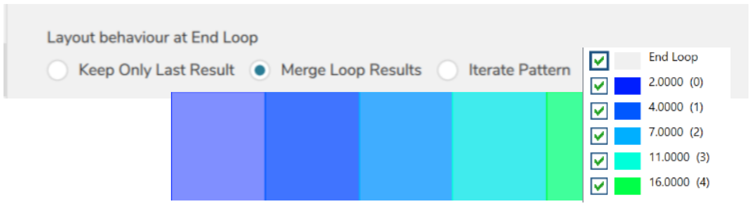
Loop

Layout behaviour at End Loop

[✓ Keep Only Last Result] Merge Loop Results Iterate Pattern

✓ End Loop
✓ 🟩 16.0000 (0)

b.2)

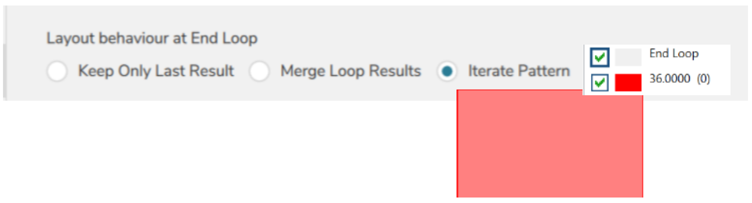


Layout behaviour at End Loop

Keep Only Last Result [✓ Merge Loop Results] Iterate Pattern

✓ End Loop
✓ 🟦 2.0000 (0)
✓ 🟦 4.0000 (1)
✓ 🟦 7.0000 (2)
✓ 🟦 11.0000 (3)
✓ 🟩 16.0000 (4)

b.3)



Layout behaviour at End Loop

Keep Only Last Result Merge Loop Results [✓ Iterate Pattern]

✓ End Loop
✓ 🟥 36.0000 (0)

Figure 12. a) Layout that enters to the loop module with a dose of 1, b) result of the layout using different ending loop behaviour.