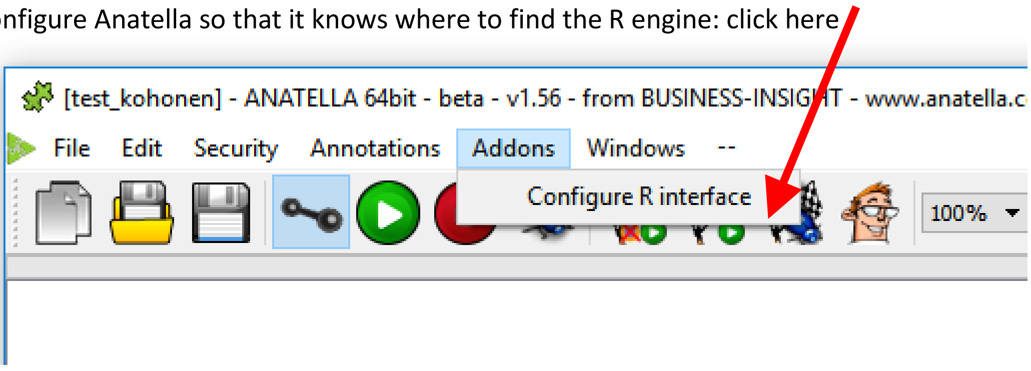
A brief introduction to coding in R with Anatella

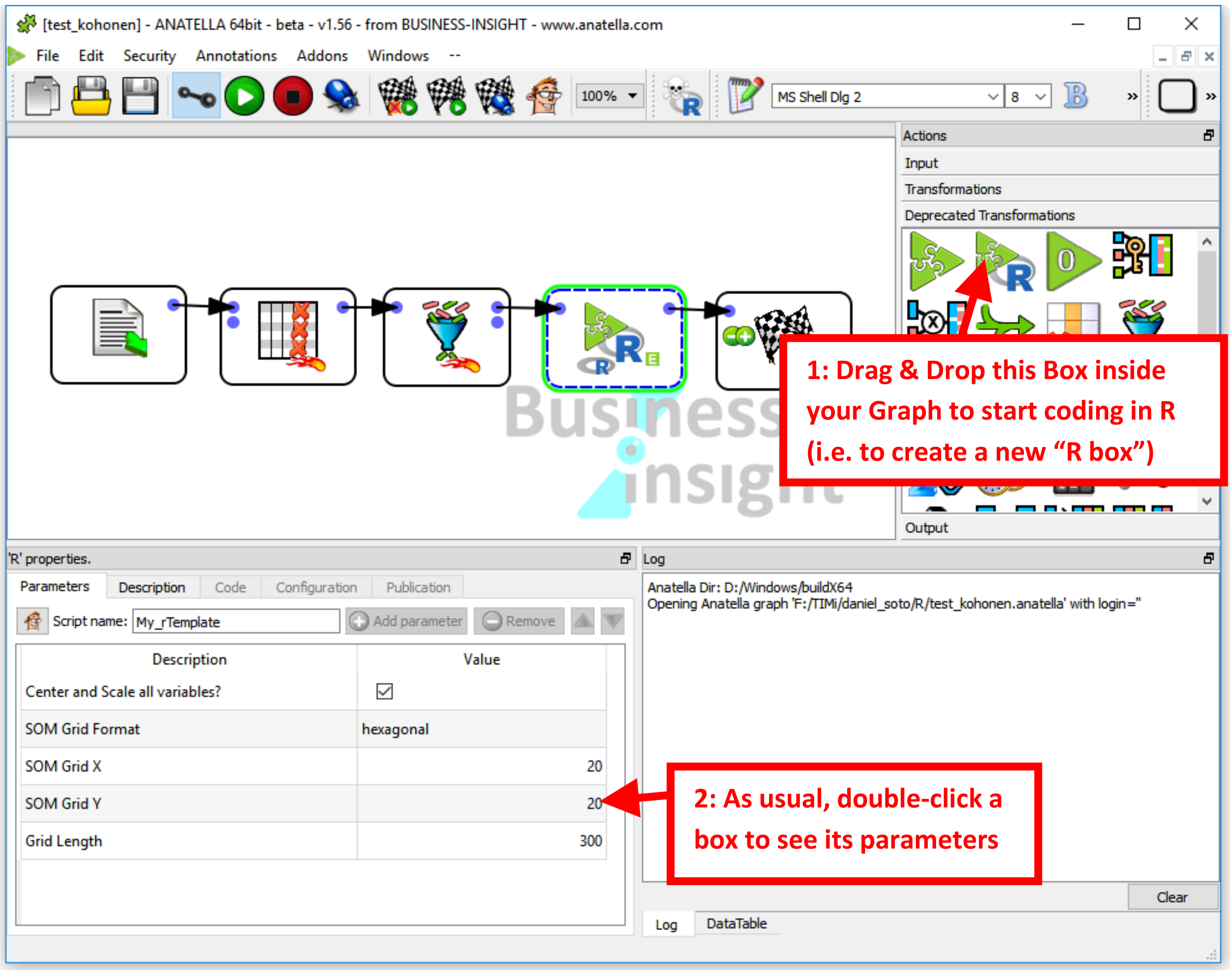
Before using the R engine within Anatella, you must first:

1. install & download the R engine from here:
   https://cran.r-project.org/bin/windows/base/
2. configure Anatella so that it knows where to find the R engine: click here



[test_kohonen] - ANATELLA 64bit - beta - v1.56 - from BUSINESS-INSIGHT - www.anatella.c
File
Edit
Security
Annotations
Addons
Windows
--
Configure R interface
100%

Here is the standard Anatella Window:



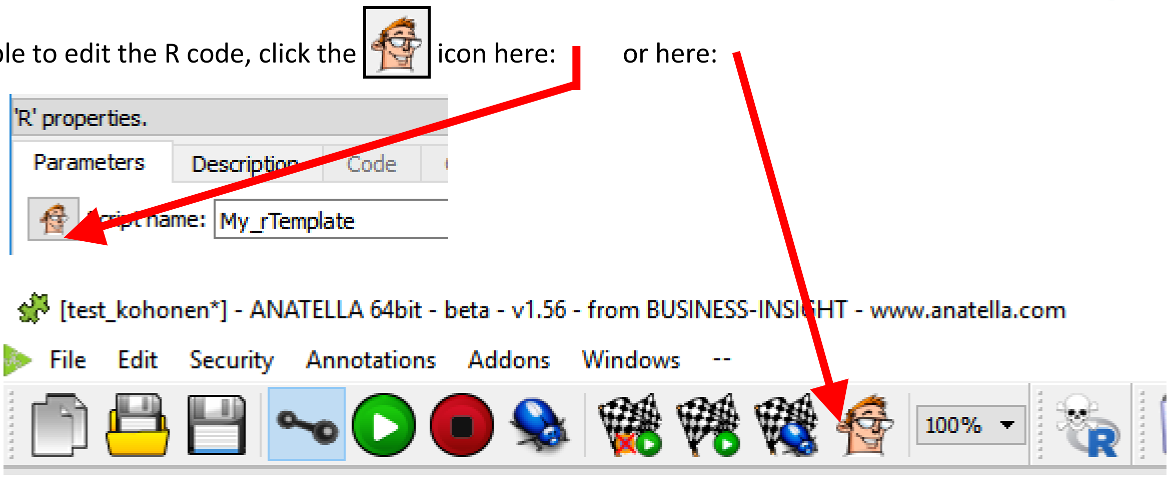
[test_kohonen] - ANATELLA 64bit - beta - v1.56 - from BUSINESS-INSIGHT - www.anatella.com
File
Edit
Security
Annotations
Addons
Windows
--
100%
MS Shell Dlg 2
8
Actions
Input
Transformations
Deprecated Transformations
Output
1: Drag & Drop this Box inside your Graph to start coding in R (i.e. to create a new "R box")
2: As usual, double-click a box to see its parameters
'R' properties.
Parameters
Description
Code
Configuration
Publication
Script name:
My_rTemplate
Add parameter
Remove
Log
Clear
Log
DataTable

Log:

```
Anatella Dir: D:/Windows/buildX64
Opening Anatella graph 'F:/TIMi/daniel_soto/R/test_kohonen.anatella' with login=''
```

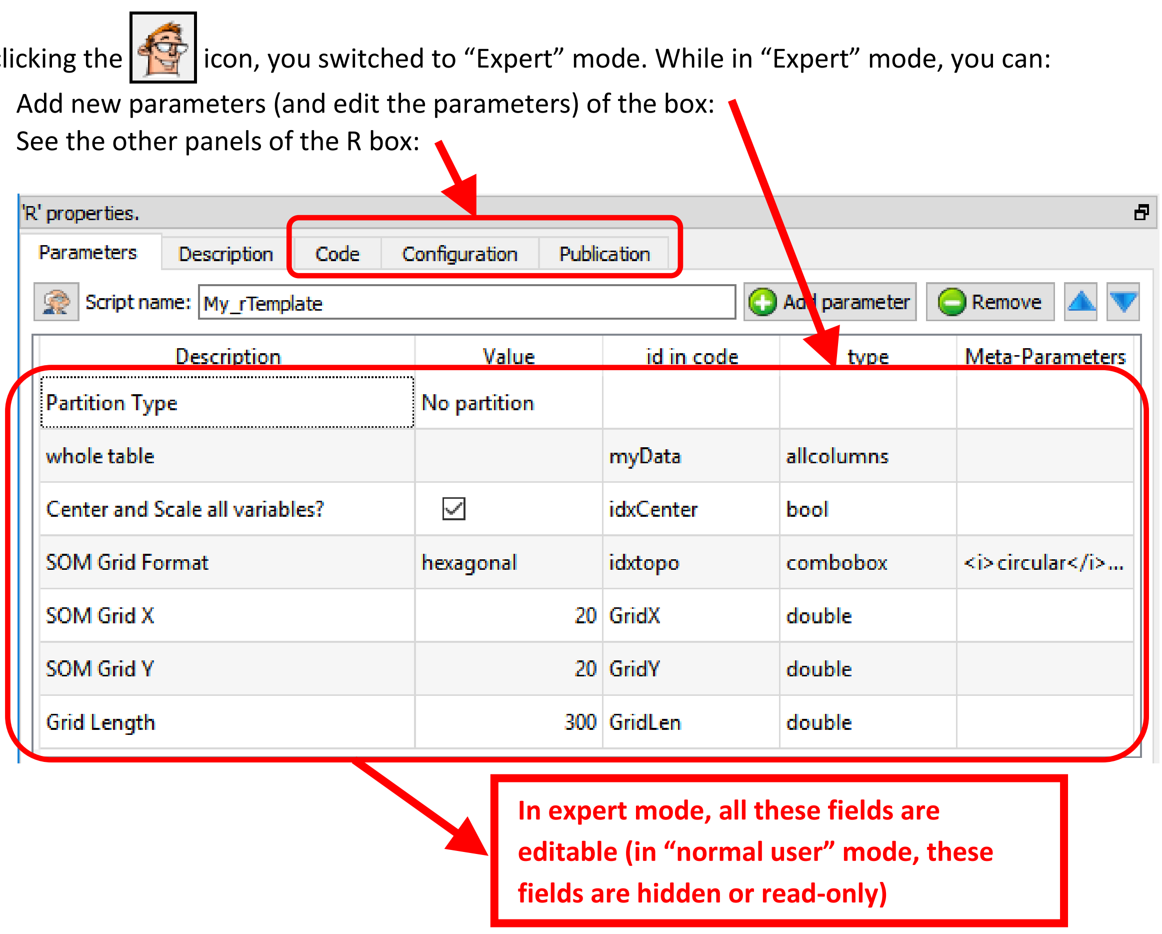
Description	Value
Center and Scale all variables?	☑
SOM Grid Format	hexagonal
SOM Grid X	20
SOM Grid Y	20
Grid Length	300

To be able to edit the R code, click the icon here: or here:



After clicking the icon, you switched to "Expert" mode. While in "Expert" mode, you can:

- Add new parameters (and edit the parameters) of the box:
- See the other panels of the R box:

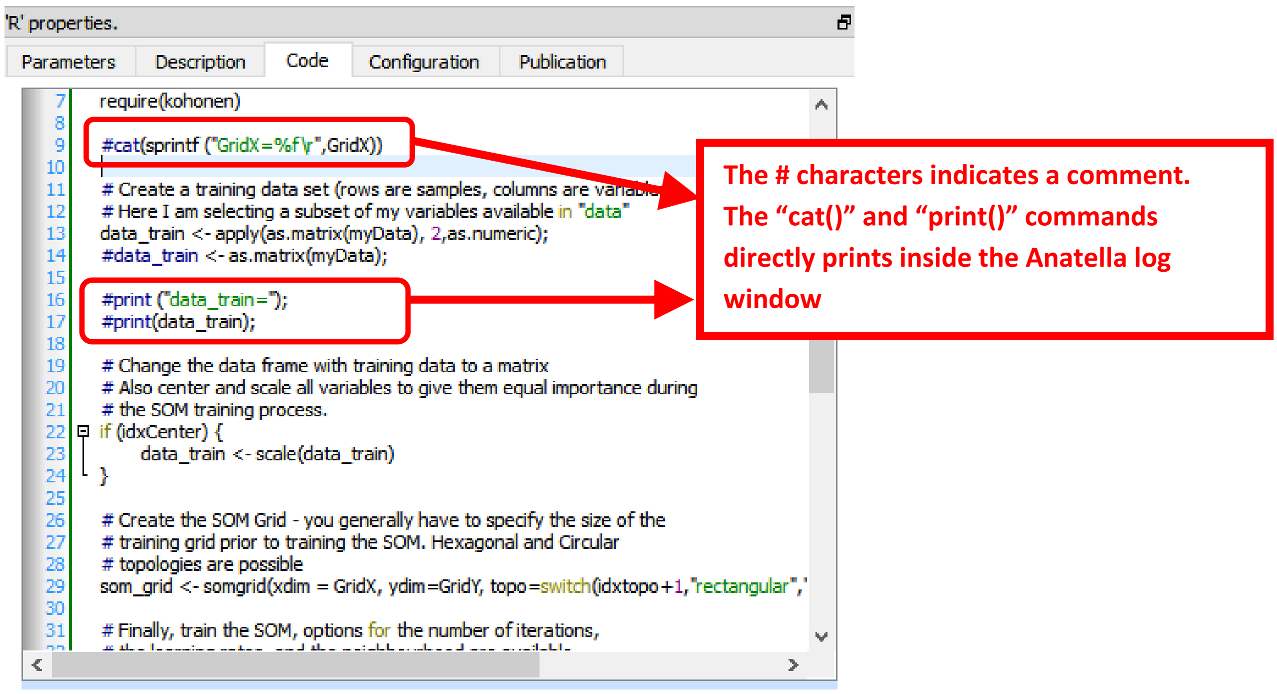


Description	Value	id in code	type	Meta-Parameters
Partition Type	No partition			
whole table		myData	allcolumns	
Center and Scale all variables?	✓	idxCenter	bool	
SOM Grid Format	hexagonal	idxtopo	combobox	<i>circular</i>...
SOM Grid X	20	GridX	double	
SOM Grid Y	20	GridY	double	
Grid Length	300	GridLen	double	

In expert mode, all these fields are editable (in "normal user" mode, these fields are hidden or read-only)

In the above example, we can see that Anatella will initialize the R engine with 6 variables (these 6 variables are named "myData", "idxCenter", "idxtop", "GridX", "GridY, "GridLen") before running the R code. All the parameters that are tables (coming for the input pins) are injected inside the R engine as "data frames".

In particular, the “Code” panel is interesting: it contains the R code: Here is an example of R code:



```
require(kohonen)

#cat(sprintf ("GridX=%f\r",GridX))

# Create a training data set (rows are samples, columns are variable
# Here I am selecting a subset of my variables available in "data"
data_train <- apply(as.matrix(myData), 2,as.numeric);
#data_train <- as.matrix(myData);

#print ("data_train=");
#print(data_train);

# Change the data frame with training data to a matrix
# Also center and scale all variables to give them equal importance during
# the SOM training process.
if (idxCenter) {
      data_train <- scale(data_train)
}

# Create the SOM Grid - you generally have to specify the size of the
# training grid prior to training the SOM. Hexagonal and Circular
# topologies are possible
som_grid <- somgrid(xdim = GridX, ydim=GridY, topo=switch(idxtopo+1,"rectangular",'

# Finally, train the SOM, options for the number of iterations,
```

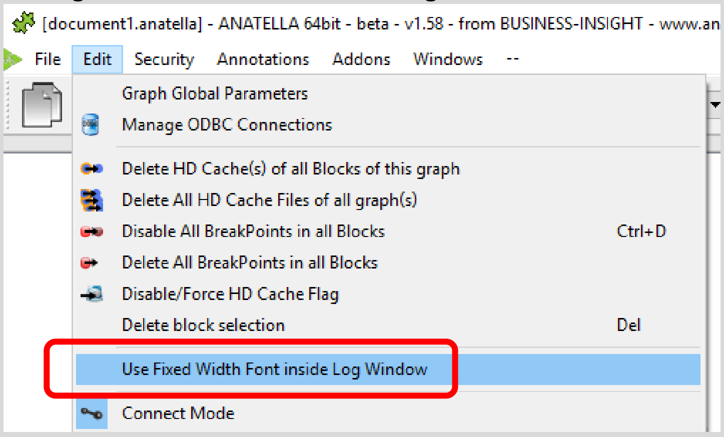
The # characters indicates a comment. The “cat()” and “print()” commands directly prints inside the Anatella log window

You can use the "print()" or the “cat()” command to display some values inside the Anatella Log window (this is useful for debugging your code).

When the R engine prints some results inside the Anatella Log Window, it assumes that the font used to display the results is of **Constant-Width** (e.g. so that, we you print an array, the different columns from your array are correctly aligned on each row).

By default, Anatella uses a Variable-Width font inside the Log Window (and thus the array’s are not displayed very nicely).

You can change the font used inside the Log Window here:


[document1.anatella] - ANATELLA 64bit - beta - v1.58 - from BUSINESS-INSIGHT - www.an
File
Edit
Security
Annotations
Addons
Windows
--
Graph Global Parameters
Manage ODBC Connections
Delete HD Cache(s) of all Blocks of this graph
Delete All HD Cache Files of all graph(s)
Disable All BreakPoints in all Blocks Ctrl+D
Delete All BreakPoints in all Blocks
Disable/Force HD Cache Flag
Delete block selection Del
Use Fixed Width Font inside Log Window
Connect Mode

(You can also use “CTRL+Wheel” to zoom in/out the text inside the Log Window)

The # characters at the start of a line indicates a comment. Inside R, there are no easy way to comment several lines of codes (i.e. there are no /*…*/ as in other languages such as Javascript, C, C#, Java, etc.).

Here is a special extension that is only available inside Anatella: Use the string "##astop" (without the quotes at the beginning and the end) at the very start of a line to prevent Anatella to execute any R code located beyond the "##astop" flag. This is handy when developing new R Actions.

Here are some "good practice" rules to follow when creating a new R code:

1. It might be easier to use an interactive tool such as "R-Studio" to develop the first version of your R code (i.e. during the first "iterations" of code development). Once your R code is working more-or-less properly, you can fine-tune its integration inside an Anatella box using the Anatella GUI. Once the integration inside Anatella is complete, you'll have a block of R code (i.e. an Anatella box) that you can re-use easily everywhere with just a simple drag&drop! (…and without even looking at the R code anymore!)

When developing a new code in R, it happens very often that the R engine computes for a very long time (for example, because you didn't define properly a parameter) and the whole data-transformation is "freezing" abnormally for very long period of time. In such situation, don't hesitate to click the ⏹ button inside the toolbar to abort all the computations prematurely. The Anatella GUI remains stable even if you cancel all the time the R engine. This allows to make many iterations, to quickly arrive to a working code.

2. If you need a specific data-type to run your R computations, ensure that you convert to this specific data-type before doing any computation. For example, don't assume that you'll always receive a matrix full of numbers in input: More precisely: Always force the conversion to the "number type", if you specifically need "numbers" to do your computations. To convert a data frame (received as input) that is named "myDataFrame" into an Array of numbers (and get rid of the strings!) that is named "myArray":

```
myArray <- apply (as.matrix (myDataFrame), 2, as.numeric);
```

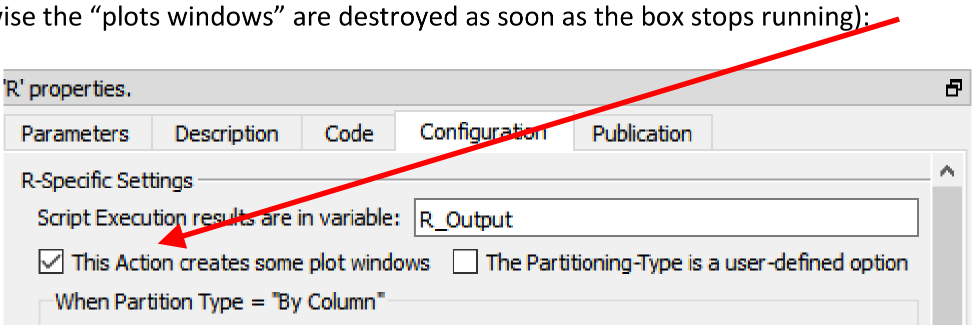
3. If you need some specific packages to run your R code, add a few lines of code (at the top of your R script) that installs the required packages if they are not there yet. For example, this install the "kohonen" package if it's not there yet:

```
if("kohonen" %in% rownames(installed.packages()) == FALSE)
{
    print("installing cohonen package for first time use\r");
    install.packages("kohonen",repos=RemoteRepository)
}
```

(don't forget to use the option "`repos=RemoteRepository`" inside the above "`install.packages`" command)

When creating a R box that displays some plot window:

1. in the "Configuration" panel, check the option "*This Action creates some plot windows*" (otherwise the "plots windows" are destroyed as soon as the box stops running):



'R' properties.

Parameters | Description | Code | Configuration | Publication

R-Specific Settings

Script Execution results are in variable: R_Output

[✓] This Action creates some plot windows [ ] The Partitioning-Type is a user-defined option

When Partition Type = "By Column"

2. Use "x11();" to open new "plot windows" (otherwise all plots ends up inside the same plot-window and you only see the last plot because it has "overwritten" all the previous ones)

3. **<u>Optional:</u>** To avoid consuming much RAM memory for nothing (while R is just busy showing your plots), add at the end of your R code a few lines to destroy all large matrices stored in RAM. For example:

```
#free up memory:
myDataFrame=0; # replace the large data-frame named "myDataFrame" with
               # a single number (0) to reclaim RAM
gc();          # run garbage collector to force R to release RAM
```

To pass some table-results as output of the R box, use the "R_Output" variable. The data-type of the variable used as output is very precise: it must be a data frame (and not an array). To convert your variables to data frames, use the following command:

```
myDataFrame <- data.frame( MyVariable, stringsAsFactors=FALSE)
```

(don't forget the option "`stringsAsFactors=FALSE`", otherwise R does strange things!!).

Here are some example of usage of the output variable named "R_Output":

1. To pass on output pin 0 the data frame named "myDataFrame", simply write:

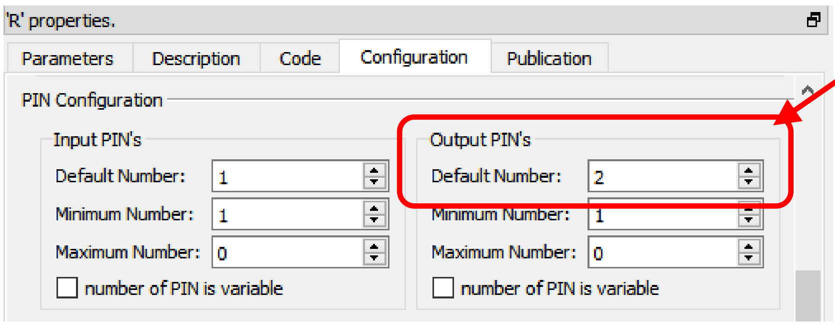
   ```
   R_Output <- myDataFrame
   ```

   Please ensure that the type of the variable named "myDataFrame" is indeed a "data frame" (and not an "Array"), otherwise it won't work.

2. To pass on output pin 0 the data frame named "myDataFrame1" and to pass on output pin 1 the data frame named "myDataFrame2", simply write:

   ```
   R_Output <- list(myDataFrame1, myDataFrame2)
   ```

For the above code to work, you must also setup Anatella to have 2 output pins on your box:

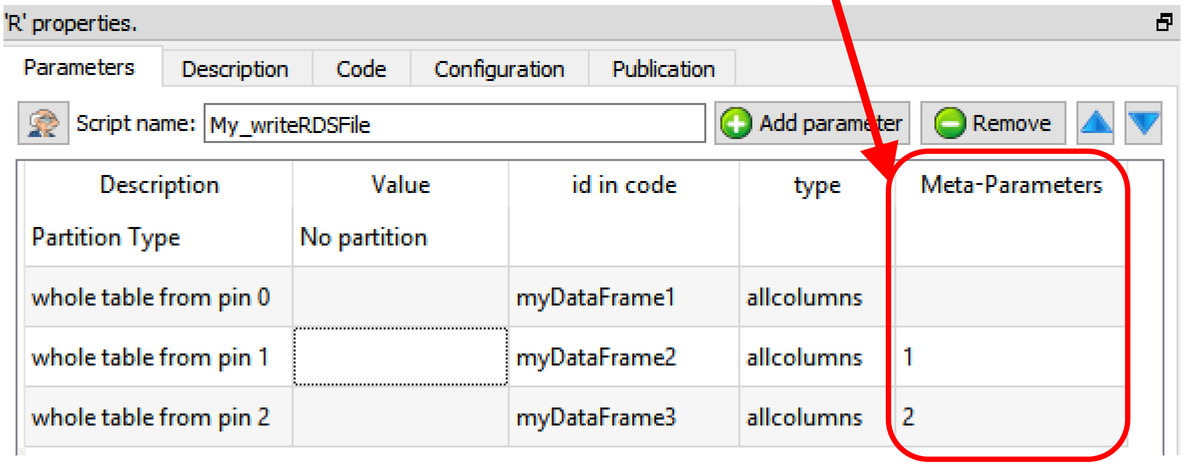

'R' properties.

Parameters | Description | Code | Configuration | Publication

PIN Configuration

Input PIN's		Output PIN's	
Default Number:	1	Default Number:	2
Minimum Number:	1	Minimum Number:	1
Maximum Number:	0	Maximum Number:	0
☐ number of PIN is variable		☐ number of PIN is variable	

To get inside the R environment the input tables available on the second, third, fourth,... pins, use the column "Meta–Parameters" inside the "Parameters" panel: For example, this define 3 data frames (named "myDataFrame1", "myDataFrame2" and "myDataFrame3") that contains the tables on the input pins 0,1 & 2:

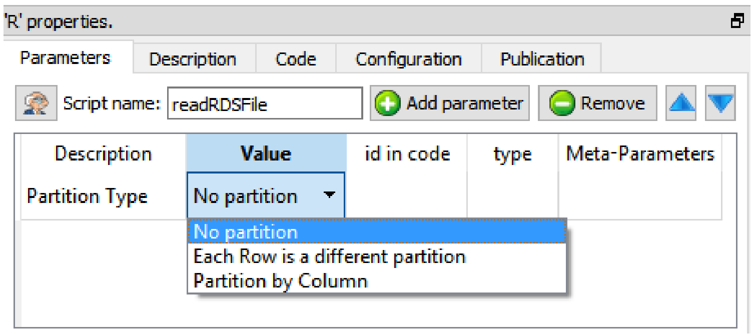

'R' properties.

Parameters | Description | Code | Configuration | Publication

Script name: My_writeRDSFile

Description	Value	id in code	type	Meta-Parameters
Partition Type	No partition			
whole table from pin 0		myDataFrame1	allcolumns	
whole table from pin 1		myDataFrame2	allcolumns	1
whole table from pin 2		myDataFrame3	allcolumns	2

The R engine is quite limited in terms of the size of the data it can analyze because all the data must fit into the (small) RAM memory of the computer. To alleviate this limitation of R, you can ask to Anatella to partition your data. There are currently 3 different partitioning options inside Anatella:


'R' properties.

Parameters | Description | Code | Configuration | Publication

Script name: readRDSFile

Description	Value	id in code	type	Meta-Parameters
Partition Type	No partition			

No partition
Each Row is a different partition
Partition by Column

How does it work? The table on the first input pin (i.e. on pin 0) is "splitted" in many different "little" tables (the tables on the other pins – pin 1, pin 2, pin 3, etc. – are always injected completely inside the R engine without any "splitting"). The R engine can process easily each of this "little" table because they only consumes a small quantity of RAM memory. After the split, Anatella calls the R engine "in-a-loop", several times: At each iteration, the R engine process one different "little" table (and it might also produce some output).

There are three "Partition Types":

- **No partition** (the default option): self-explaining.
- **Each Row is a different partition**: self-explaining.
- **Partition by Column**: When using this option you must select a "Partitioning Column". For example, if you select as "Partitioning Column" the column "Age", then each partition will containing all the people (i.e. all the rows) with the same "Age".

  The concept of "Partition" is used many times inside Anatella: e.g. See the sections 4.8.5.3. (Partitioned Sort), 4.8.11.4 (Time Travel), 4.8.7.9. (Quantile), 4.8.5.9. (Flatten), 4.8.3.2.5. (Multithread Run), 4.8.11.2. (Smoothen Rows) of the "AnatellaQuickGuide.pdf" where the same partitioning concept is also used.

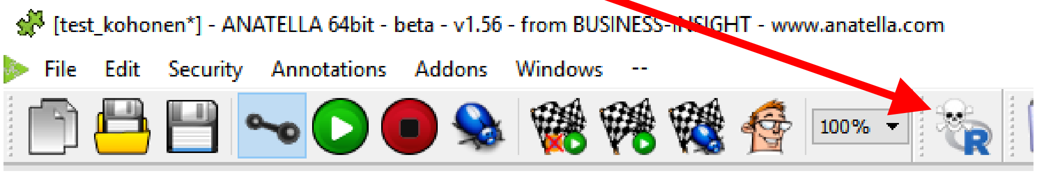
More precisely, when using partitioning, Anatella does the following:

1. Split the table on pin 0 into many different "little" tables (one table for each different partition).
2. Inject into R the variables based on the complete tables available on pin 1, pin 2, pin 3, etc.
3. Inject into R the variable named "partitionType" (whose value is 0, 1 or 2, depending on which "Partition Type" you are using).
4. Inject into R the variable named "iterationCount" with the value 0.
5. Inject into R the variable named "finished" with the value "false".
6. Run the loop (i.e. process each partition):
   - Inject into R one of the "little" tables (that are coming from the "big" table available on the first input pin).
   - Run your code inside the R engine.
   - Get back some output results to forward onto the output pin.
   - Increase by one the value of the variable "iterationCount".
   - Execute the next iteration of the loop until there are no more "little" tables to process.
7. Set the variable named "finished" to the value "true".
8. Run the R engine one last time.
9. Get back some output results to forward onto the output pin.

One example where partitioning "makes sense" is the following: Let's assume that you are working for a large retailer (such as Carrefour, Wallmart, etc) and you need to manage the stocks of all your different products (the products are also named "SKU"=Stock Keeping Unit) at all your different Point-Of-Sales (POS). One important part of stock management involves predicting what will be the demand of each (SKU;POS) pair in the fore coming weeks. If you predict a high demand for a specific (SKU;POS) pair, then you'd better have a significant stock for this same (SKU;POS) pair (unless you want to lose sales because of "out-of-stock" conditions). A typical large retailer has around 20,000 SKU's at each of their POS. Let's assume that we have 200 POS. We thus have 20,000x200=4,000,000 (SKU;POS) pairs. This means that we'll have to compute 4,000,000 predictions (one for each (SKU;POS) pair). This also means that we'll typically have a matrix with 4,000,000 rows: Each row contains information about the past demand for a (SKU;POS) pair. Using the values available on the current row (about past demands), we'll typically use some "time series" algorithm to predict the future demand (for the fore coming weeks). Actually, to compute the prediction for a specific (SKU;POS) pair, **you only need one row of the**

table. In other words, all computation can be done row-by-row. This means that we can use the "Partition Type" named "*Each Row is a different partition*". Because of the Anatella-Partitioning-Algorithm, we can handle any number of SKU or POS without being limited by the R engine when it comes to handle large matrices.

There is also a new button inside the toolbar here:

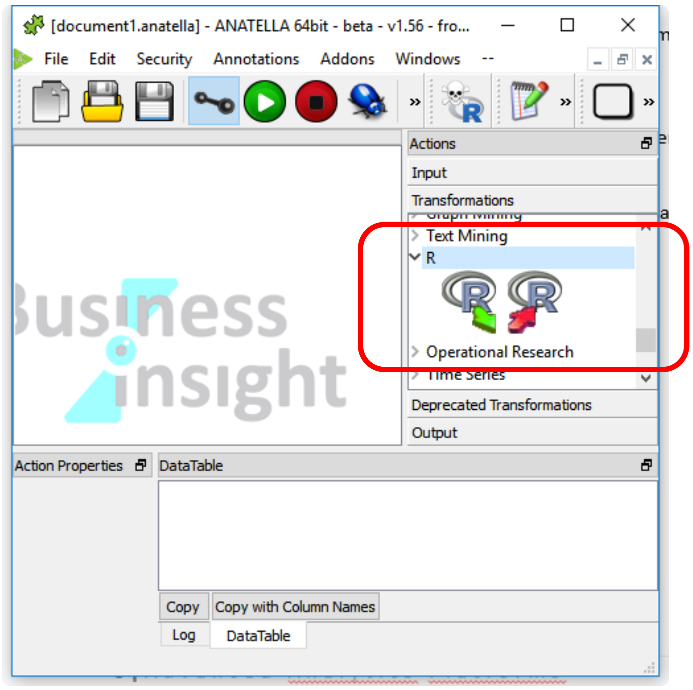

[test_kohonen*] - ANATELLA 64bit - beta - v1.56 - from BUSINESS-INSIGHT - www.anatella.com
File Edit Security Annotations Addons Windows --
100%

This new button kills all the R engines currently running. This means that, when you click this button:

- All plot-windows are closed
- All R computations are stopped (i.e. don't click this button when your graph is running!).

Use the button inside the toolbar to close in one click all the plot-windows.

Once you are satisfied with your box, you can "publish it" so that it always becomes available inside the "common" re-usable actions: For example:



Please refer to the sections 6.2.8 and 6.3 inside the "AnatellaQuickGuide.pdf" to know more about the process of publishing (and sharing with your colleagues) new boxes developed in R (or developed in Javascript: i.e. this is the same process).