Methods of data generation with Curiosity Software

When using the Curiosity Platform, there are multiple methods available to populate the rule sets needed to generate data. There are also different methods of activating synthetic data generation, and users can use a combination of these methods to configure the rule needed for each of their use cases.

It's also possible to execute the synthetic data generation from multiple routes. These include: using an API call such as from an automation framework or CI/CD pipeline, using a submit form from the self-service portal, or using an iFrame to integrate into another platform such as a Confluence.

Section 3 – Using defaults, data painter and rule set accelerators

There are several ways to configure rule sets in the Curiosity Platform, and the three most common are covered in this training. You will learn how to use 'Defaults', 'Data painter' and 'Rule set accelerators'.

Method 1:

Defaults

These are great for standard synthetic data generation rules that are commonly seen within organisations and can be used to solve common use cases.

Method 2:

Data Painter

This allows the user to create or modify functions typically for advanced or rare use cases.

Method 3:

Rule set accelerators

These can be used to quickly configure a rule set to accomplish specific tasks, such as to clone a specific set of data and create copies from it.

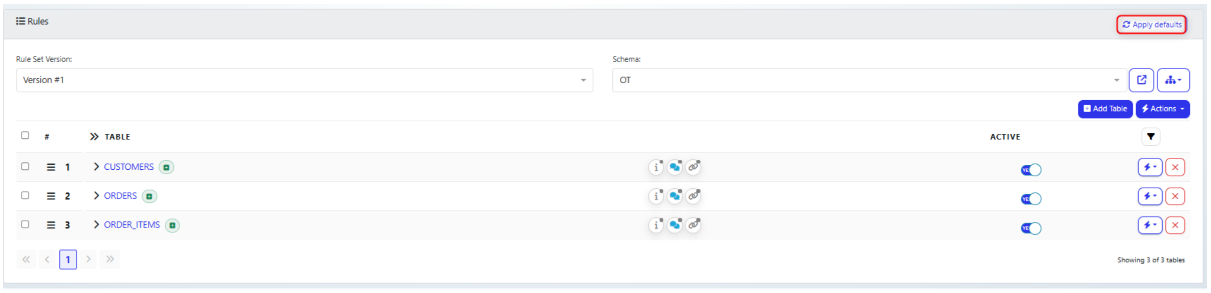
Method 1: Using defaults

The Curiosity platform comes with defaults already available, or you can create your own. To learn more about defaults, please visit our knowledge base: Defaults

Applying defaults is normally the first way to populate a rule set with synthetic data generation rules. Using the extensive analysis undertaken in the profiling and discovery training (Module 1 of this course), the default rules can be applied depending on criteria such as tags, column names and the content of the data.

1. Apply global defaults

   In the rule set screen you can see the '**Apply Defaults'** button that allows users to apply global defaults.

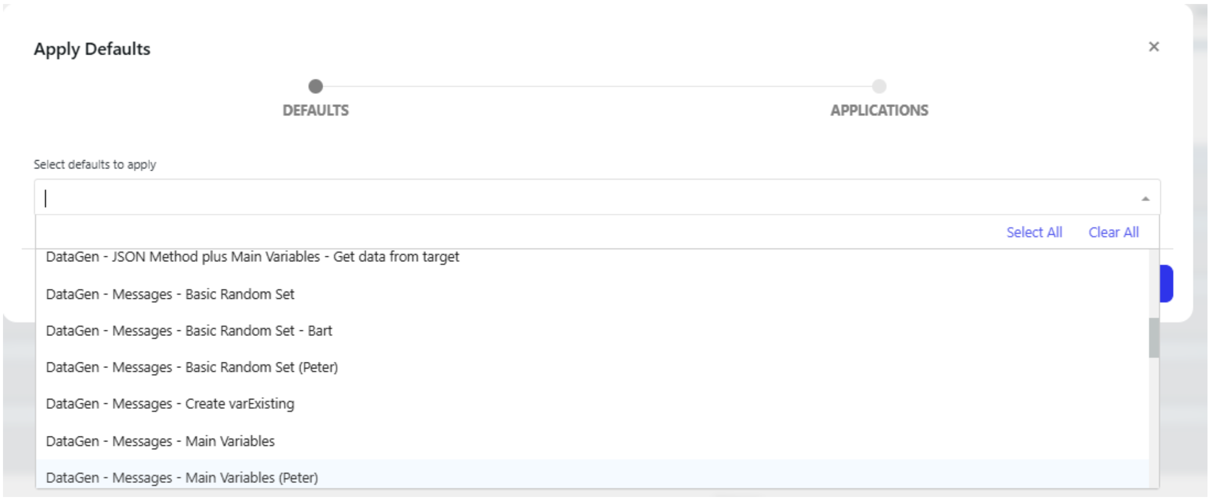


Rule Set Version: Version #1
Schema: OT

	#	TABLE	ACTIVE
☐	1	CUSTOMERS	●
☐	2	ORDERS	●
☐	3	ORDER_ITEMS	●

Showing 3 of 3 tables

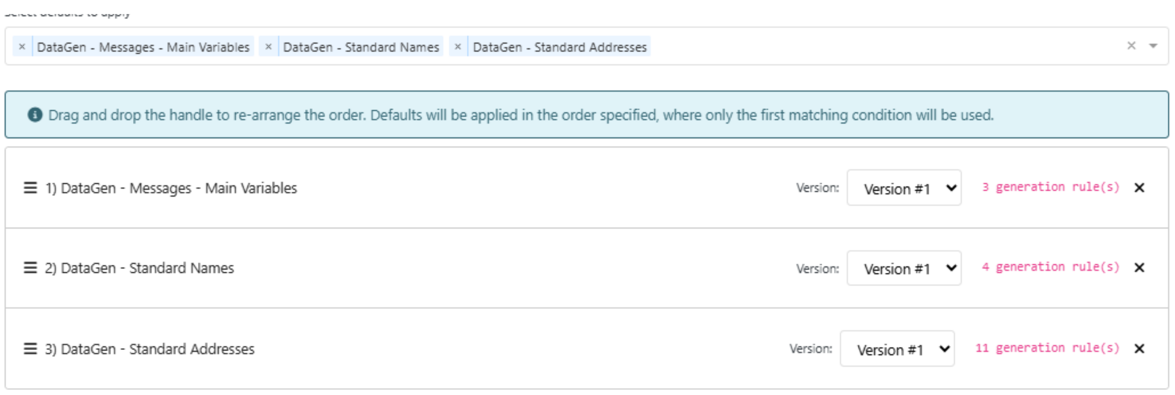
2. When you click the '**Apply Defaults'** button the following screen will open:



DataGen - JSON Method plus Main Variables - Get data from target
DataGen - Messages - Basic Random Set
DataGen - Messages - Basic Random Set - Bart
DataGen - Messages - Basic Random Set (Peter)
DataGen - Messages - Create varExisting
DataGen - Messages - Main Variables
DataGen - Messages - Main Variables (Peter)

The drop down displays the different default rule sets you can apply to the data generation rule set. You can select multiple default rule sets and order their priority in the screen, as only one generation rule can be applied to each column.

3. Once the default rule sets have been chosen, the tool will ingest the information that has been gathered about the rule set such as tags, column names and the data to identify which default rules can be applied.


× DataGen - Messages - Main Variables × DataGen - Standard Names × DataGen - Standard Addresses

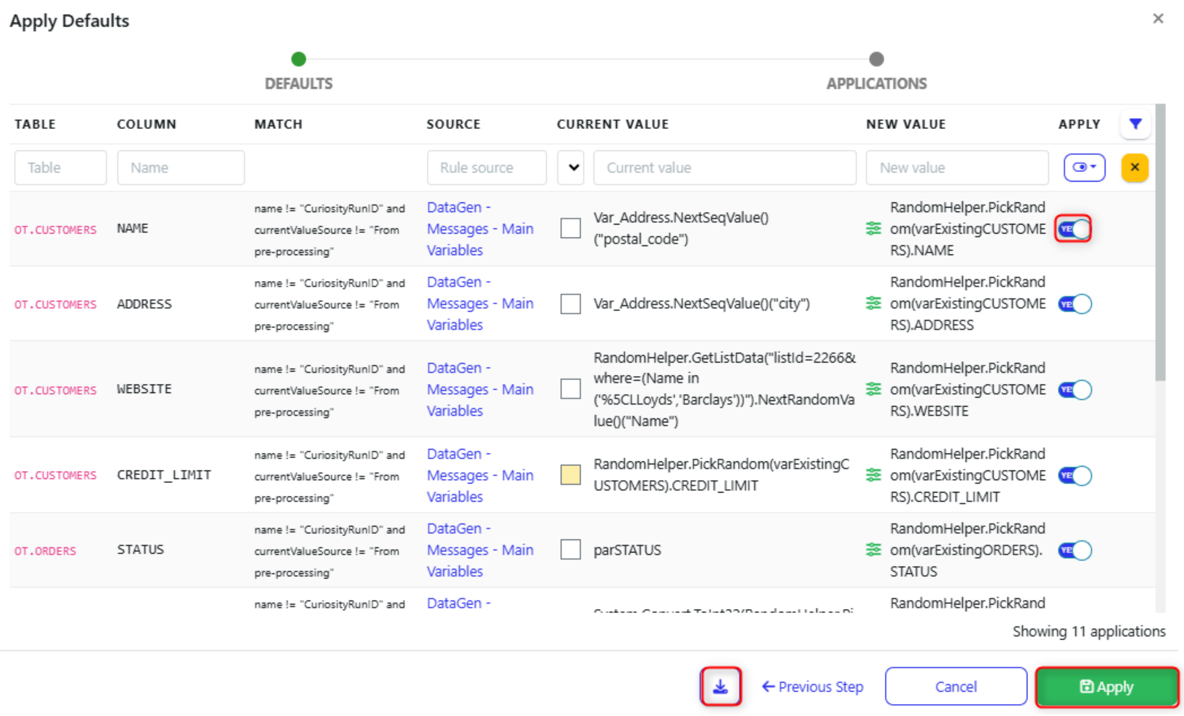
Drag and drop the handle to re-arrange the order. Defaults will be applied in the order specified, where only the first matching condition will be used.

1) DataGen - Messages - Main Variables	Version: Version #1	3 generation rule(s)	×
2) DataGen - Standard Names	Version: Version #1	4 generation rule(s)	×
3) DataGen - Standard Addresses	Version: Version #1	11 generation rule(s)	×

The next step is to decide which of the default rules to apply to the generation rule set, by using the toggle next to each rule. The screen shows which rules will be applied to which column and what the current rule is and what it will be changed to. It also tells you the source of the previous rule to aid a decision in whether to replace it or not

You can also use the 'Download' button to download the changes into a CSV file and view them.

4. Click **'Apply'** to implement the selected rules.


Apply Defaults

DEFAULTS — APPLICATIONS

TABLE	COLUMN	MATCH	SOURCE	CURRENT VALUE	NEW VALUE	APPLY
OT.CUSTOMERS	NAME	name != "CuriosityRunID" and currentValueSource != "From pre-processing"	DataGen - Messages - Main Variables	Var_Address.NextSeqValue()("postal_code")	RandomHelper.PickRandom(varExistingCUSTOMERS).NAME	YES
OT.CUSTOMERS	ADDRESS	name != "CuriosityRunID" and currentValueSource != "From pre-processing"	DataGen - Messages - Main Variables	Var_Address.NextSeqValue()("city")	RandomHelper.PickRandom(varExistingCUSTOMERS).ADDRESS	YES
OT.CUSTOMERS	WEBSITE	name != "CuriosityRunID" and currentValueSource != "From pre-processing"	DataGen - Messages - Main Variables	RandomHelper.GetListData("listId=2266& where=(Name in ('%5CLLoyds','Barclays'))").NextRandomValue()("Name")	RandomHelper.PickRandom(varExistingCUSTOMERS).WEBSITE	YES
OT.CUSTOMERS	CREDIT_LIMIT	name != "CuriosityRunID" and currentValueSource != "From pre-processing"	DataGen - Messages - Main Variables	RandomHelper.PickRandom(varExistingCUSTOMERS).CREDIT_LIMIT	RandomHelper.PickRandom(varExistingCUSTOMERS).CREDIT_LIMIT	YES
OT.ORDERS	STATUS	name != "CuriosityRunID" and currentValueSource != "From pre-processing"	DataGen - Messages - Main Variables	parSTATUS	RandomHelper.PickRandom(varExistingORDERS).STATUS	YES
		name != "CuriosityRunID" and	DataGen -		RandomHelper.PickRand	

Showing 11 applications

Exercise 4

1. Run 'Apply defaults' on your rule set

Method 2: Using data painter

The ‘**Data Painter’** enables users to create their own rules using a selection of pre-built functions that can be modified using VB.Net in-built functions. The rules are in-line VB.Net functions.

This section provides a collection of ready-to-use functions designed for common data generation scenarios, designed to help users get started quickly. For specific data generation needs, users can use the ‘**Function Editor’** to write custom logic using VB.Net.

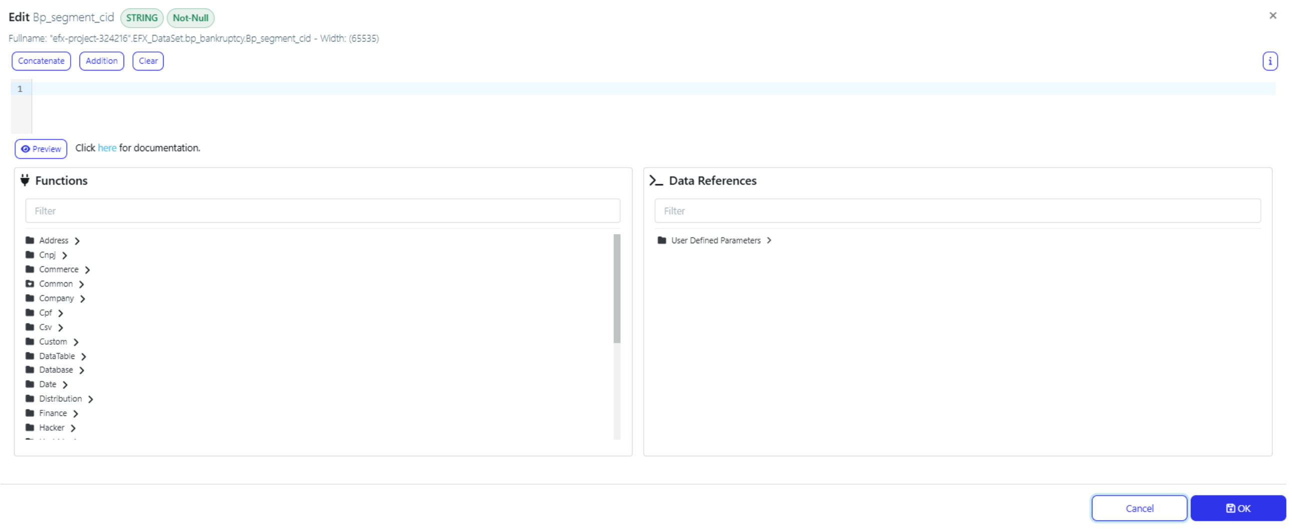
‘Step 4 – Data References’ allows for parameterisation of data using local and global variables, supporting referential integrity and dynamic data generation.

Step 1 - Data painter synthetic functions

The Curiosity Platform has an extensive library of synthetic data generation functions that can be used to create the data and use the VB.net syntax to modify.

These can be found in the **‘Functions’** section and you can apply a filter to help you find the function you would like to use. Most of the functions have parameters which control the output of the function.

For instance, the ‘RandomHelper.GetRandomDateTime()’ function takes two parameters for the minimum date and maximum date that can be generated, like so: ‘RandomHelper.GetRandomDateTime("2021-01-01", "2012-01-01")’


Edit Bp_segment_cid STRING Not-Null
Fullname: "efx-project-324216".EFX_DataSet.bp_bankruptcy.Bp_segment_cid - Width: (65535)
Click here for documentation.

Exercise 5

1. For one of the columns in your rule set, choose a function that will populate the column with realistic data

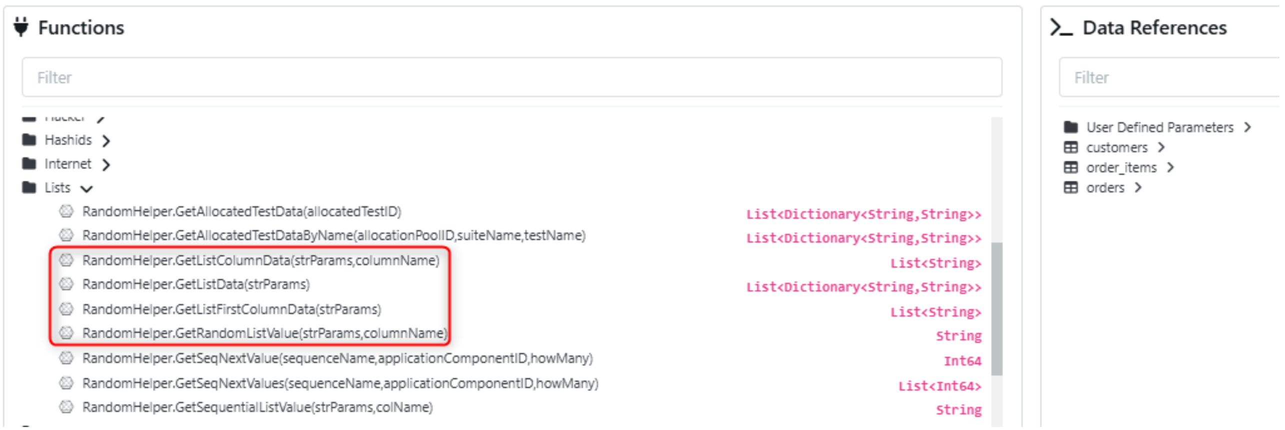
Step 2 – Data list wizards

There are also functions that return values from assets in the Curiosity Platform such as data lists and sequences. Each of these has a wizard that is designed to help the user configure the correct function to return the desired data. The functions that return data from a list all use the ‘Resolve List’ wizard, whereas the sequence functions all use the ‘Select Sequence’ wizard.

The Curiosity Platform is able to store data and make this easily accessible and editable to the user in a **Data List.** Further information can be found in our knowledge base: [Data List Information]

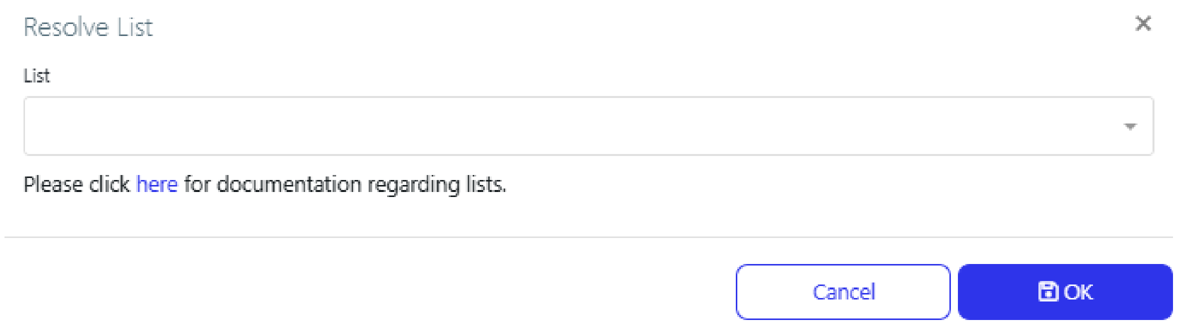
The difference between the data list functions is in the data types they return, for instance one will return a list of strings whilst another will return just a single string. Make sure you review the name of the function and the data type to ensure the data returned is what you expect.

1. Clicking on the desired list function opens the 'Resolve List' wizard. The first step is to choose the list to return data from.



```
Functions                                                                Data References
Filter                                                                   Filter
Hashids >                                                                User Defined Parameters >
Internet >                                                               customers >
Lists v                                                                  order_items >
                                                                         orders >
  RandomHelper.GetAllocatedTestData(allocatedTestID)                              List<Dictionary<String,String>>
  RandomHelper.GetAllocatedTestDataByName(allocationPoolID,suiteName,testName)    List<Dictionary<String,String>>
  RandomHelper.GetListColumnData(strParams,columnName)                            List<String>
  RandomHelper.GetListData(strParams)                                             List<Dictionary<String,String>>
  RandomHelper.GetListFirstColumnData(strParams)                                  List<String>
  RandomHelper.GetRandomListValue(strParams,columnName)                           String
  RandomHelper.GetSeqNextValue(sequenceName,applicationComponentID,howMany)       Int64
  RandomHelper.GetSeqNextValues(sequenceName,applicationComponentID,howMany)      List<Int64>
  RandomHelper.GetSequentialListValue(strParams,colName)                          String
```

2. Click **'OK'** to progress the wizard to the next screen where the inputs to the wizard configure the function created.



Resolve List

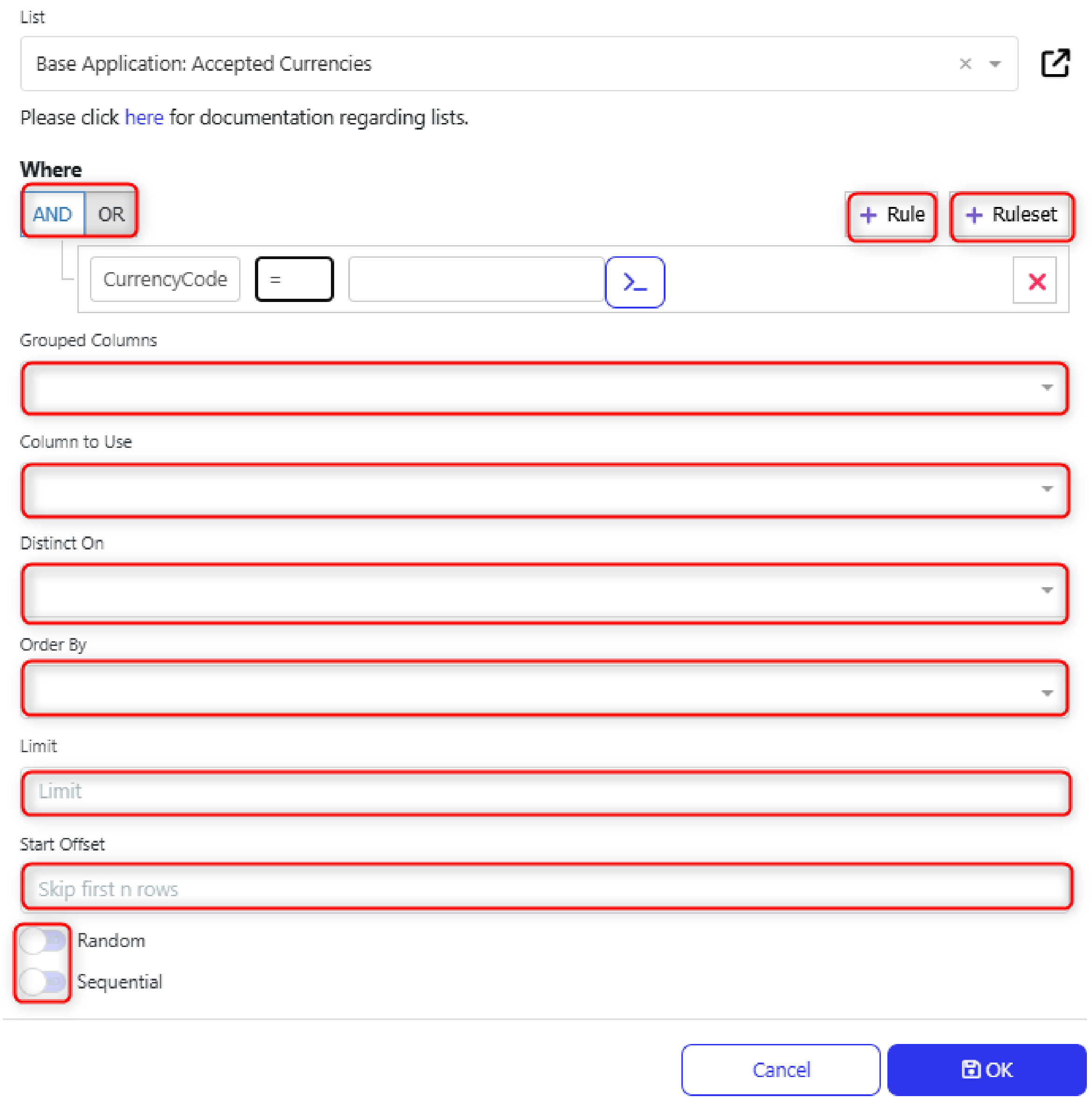
List

Please click here for documentation regarding lists.

These options include:

- **Where –** select the operator either "**AND**", "**OR**" and then select the field you want this **'where'** condition to be against. You can then choose an operator such as equals or not equals. The >_ button allows you to run these conditions against the value of a column or data variable at runtime.
- **Grouped Columns –** This allows you to choose the columns that the data will return, if you leave blank it will select all the columns
- **Column to Use –** This is the column that dictates the order for either random or sequential
- **Distinct On –** The columns to ensure unique values are returned
- **Order By –** The column to order the data returned by
- **Limit –** The number of rows returned
- **Start Offset –** The number of rows to skip and start returning data by

The **'random'** and **'sequential'** options can be toggled on or off to dictate the way data is returned to you.



List
Base Application: Accepted Currencies
Please click here for documentation regarding lists.
Where
AND OR
+ Rule + Ruleset
CurrencyCode =
Grouped Columns
Column to Use
Distinct On
Order By
Limit
Limit
Start Offset
Skip first n rows
Random
Sequential
Cancel OK

Exercise 6

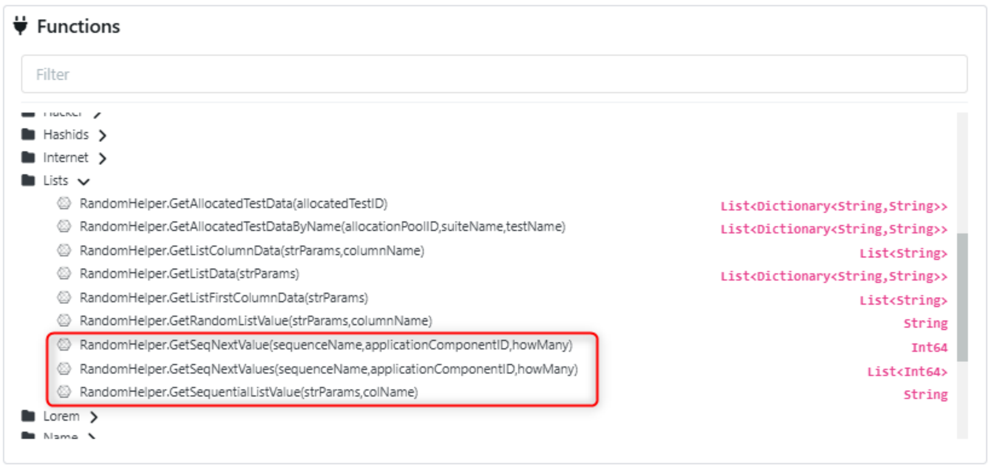
1. Use one of the data list wizards to select an address from one of the standard data lists included in the product into a user-defined variable
2. Try one of the **'where'** clauses as well to only bring in an address for a certain post code

Step 3 – Sequence wizard

With data generation, there are times where we'll need consecutive files or row values to be taken from a sequence that is handled within the Curiosity Platform. There is a wizard that helps users get one or many values from a sequence, to use for data generation.

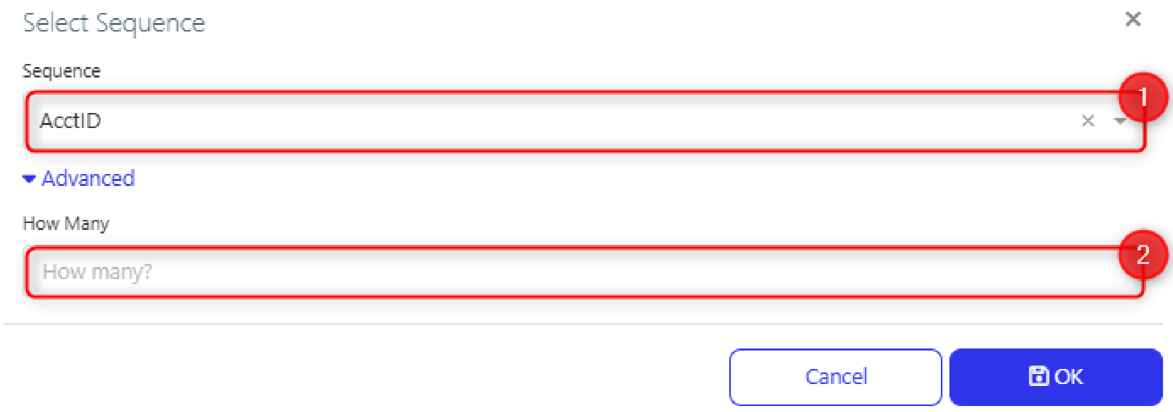
Further information can be found in our knowledge base: Sequences

1. Click on one of the functions to start the wizard

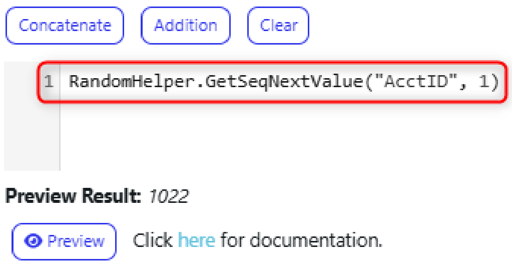


RandomHelper.GetAllocatedTestData(allocatedTestID)	List<Dictionary<String,String>>
RandomHelper.GetAllocatedTestDataByName(allocationPoolID,suiteName,testName)	List<Dictionary<String,String>>
RandomHelper.GetListColumnData(strParams,columnName)	List<String>
RandomHelper.GetListData(strParams)	List<Dictionary<String,String>>
RandomHelper.GetListFirstColumnData(strParams)	List<String>
RandomHelper.GetRandomListValue(strParams,columnName)	String
RandomHelper.GetSeqNextValue(sequenceName,applicationComponentID,howMany)	Int64
RandomHelper.GetSeqNextValues(sequenceName,applicationComponentID,howMany)	List<Int64>
RandomHelper.GetSequentialListValue(strParams,colName)	String

2. The 'Select Sequence' dialogue box will show. This is where you can choose the name of the sequence. In the 'Advanced' section you can choose how many of the values to return from the sequence.



Clicking **'OK'** will finish the wizard and populate the rule with the desired sequence function. It will now appear in data painter, and you can click on the preview button to see an example result.



```
RandomHelper.GetSeqNextValue("AcctID", 1)
```

Preview Result: *1022*

Exercise 7

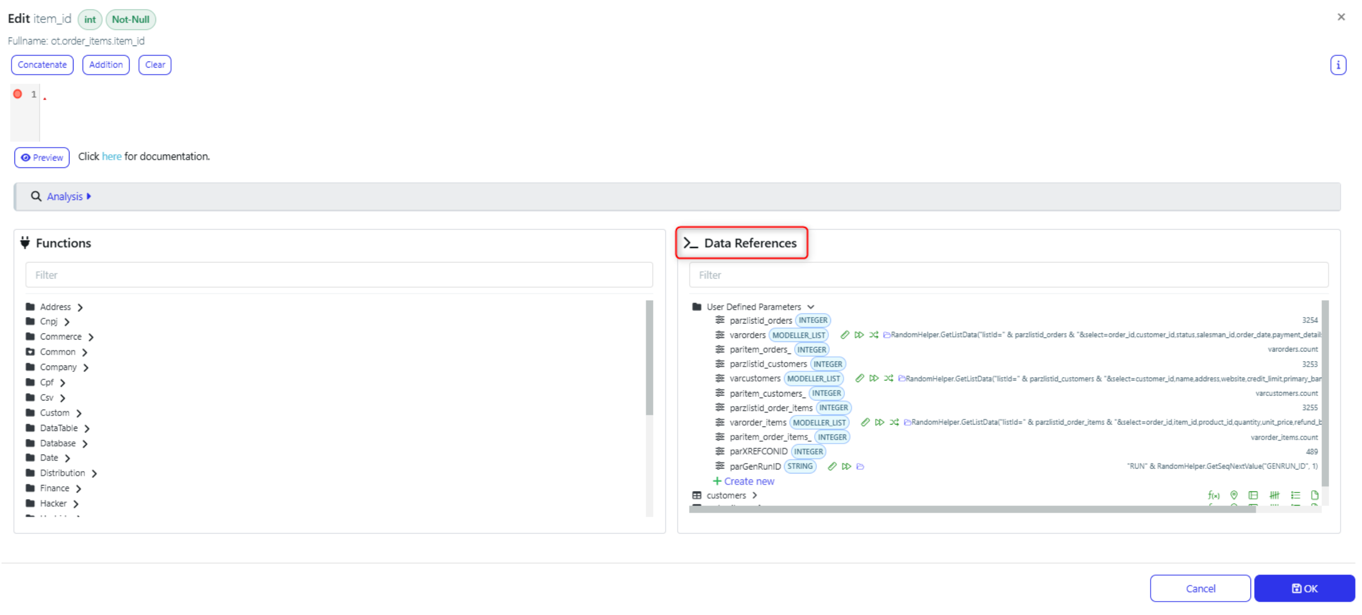
1. Select one of the columns in the rule set to use a sequence as a data generation rule

Step 4 – Data references

When creating rule sets, it can be useful to either use the user-defined variables (explained in Section 2 - Step 3) or reference a value created in the same or a related table in the generation rule set.

This allows us to keep referential integrity across parent-child relationships (which are automatically filled out if the scan has been done).

You can find them to the right of the functions screen in the data painter:

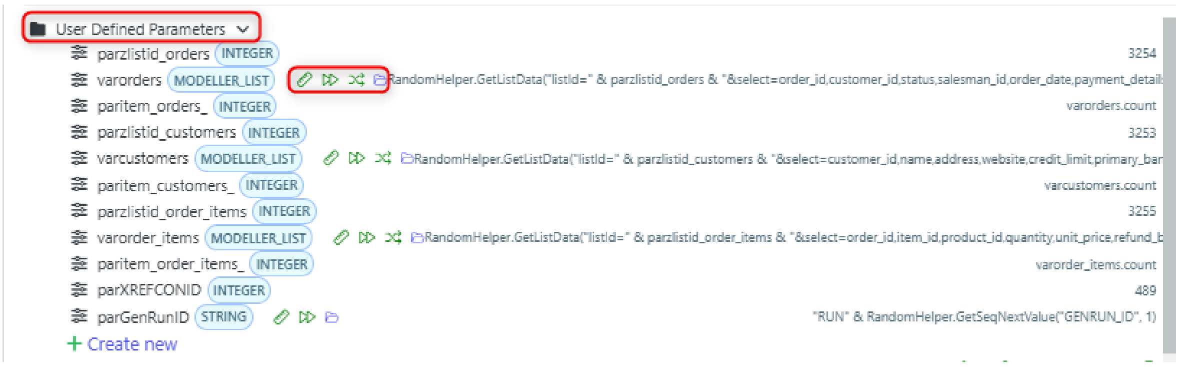


```
Edit item_id  int  Not-Null
Fullname: ot.order_items.item_id
Concatenate  Addition  Clear
1 .
Preview  Click here for documentation.
Analysis

Functions                                Data References
Filter                                   Filter
Address >                                User Defined Parameters
Cnpj >                                     parzlistid_orders  INTEGER                                3254
Commerce >                                 varorders  MODELLER_LIST  RandomHelper.GetListData("listId=" & parzlistid_orders & "&select=order_id,customer_id,status,salesman_id,order_date,payment_detail
Common >                                   paritem_orders_  INTEGER                           varorders.count
Company >                                  parzlistid_customers  INTEGER                             3253
Cpf >                                      varcustomers  MODELLER_LIST  RandomHelper.GetListData("listId=" & parzlistid_customers & "&select=customer_id,name,address,website,credit_limit,primary_bar
Csv >                                      paritem_customers_  INTEGER                     varcustomers.count
Custom >                                   parzlistid_order_items  INTEGER                           3255
DataTable >                                varorder_items  MODELLER_LIST  RandomHelper.GetListData("listId=" & parzlistid_order_items & "&select=order_id,item_id,product_id,quantity,unit_price,refund_b
Database >                                 paritem_order_items_  INTEGER                 varorder_items.count
Date >                                     parXREFCONID  INTEGER                                      489
Distribution >                             parGenRunID  STRING            "RUN" & RandomHelper.GetSeqNextValue("GENRUN_ID", 1)
Finance >                                  + Create new
Hacker >                                 customers >

                                                                                       Cancel  OK
```

Step 4a – User-defined parameters

You can use the values from the user-defined variables that either you or the tool have created. If these are a value that easily fits inside a column – such as a string or an integer – then you can click on the user-defined variable that you wish to select.

In some cases, there may be a list where it makes sense to select one value to insert into the column. In this case, icons are shown next to the parameter so you can select either a random or sequential value for the column. The folder sign also allows you to view the variable to help you visualise the function you are creating.



```
User Defined Parameters
  parzlistid_orders  INTEGER                                3254
  varorders  MODELLER_LIST  RandomHelper.GetListData("listId=" & parzlistid_orders & "&select=order_id,customer_id,status,salesman_id,order_date,payment_detail
  paritem_orders_  INTEGER                           varorders.count
  parzlistid_customers  INTEGER                             3253
  varcustomers  MODELLER_LIST  RandomHelper.GetListData("listId=" & parzlistid_customers & "&select=customer_id,name,address,website,credit_limit,primary_bar
  paritem_customers_  INTEGER                     varcustomers.count
  parzlistid_order_items  INTEGER                           3255
  varorder_items  MODELLER_LIST  RandomHelper.GetListData("listId=" & parzlistid_order_items & "&select=order_id,item_id,product_id,quantity,unit_price,refund_b
  paritem_order_items_  INTEGER                 varorder_items.count
  parXREFCONID  INTEGER                                      489
  parGenRunID  STRING            "RUN" & RandomHelper.GetSeqNextValue("GENRUN_ID", 1)
  + Create new
```

Exercise 8

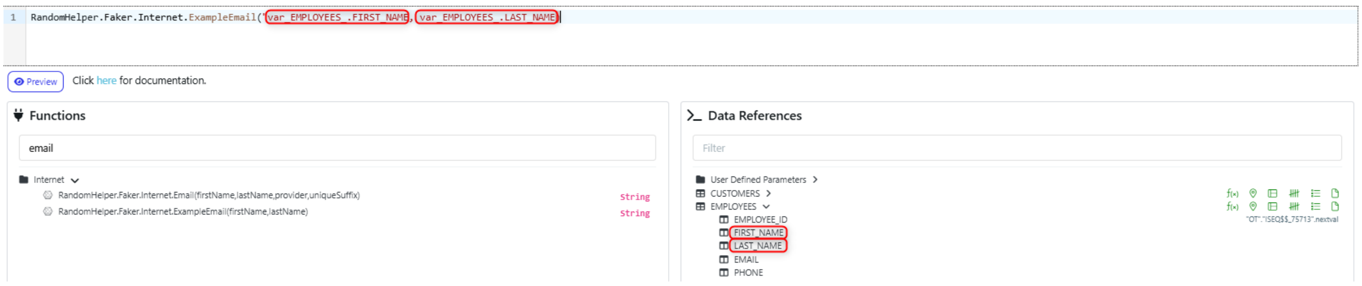
1. Use a user-defined parameter to populate one of your data generation columns in a rule set

Step 4b – Linked table references

When creating synthetic data rules, it is often required to use data that has been generated for another column, either in the same table or a different table. For instance if an email field is being generated using the values from a 'first name' and a 'last name' column in a table, to create an email that follows the pattern 'firstname.lastname@company.com'.

Additionally, when creating tables with a parent-child relationship, for example a header table called 'orders' and a detail table called 'order_items', linked by a foreign key on 'order_id', it is important to maintain referential integrity by ensuring consistent 'order_id' values when generating data for both tables.

Linked table references can help in both scenarios. By navigating to the 'Data References' section in data painter, you can select an object associated with columns in the same table as you are generating into, as well as a table that has been generated beforehand. For the email example, this means that a linked table reference can be used to the first name and last name columns to create an email address in the Email column.



```
RandomHelper.Faker.Internet.ExampleEmail(`var_EMPLOYEES_.FIRST_NAME`,`var_EMPLOYEES_.LAST_NAME`)
```

Click here for documentation.

Functions

email

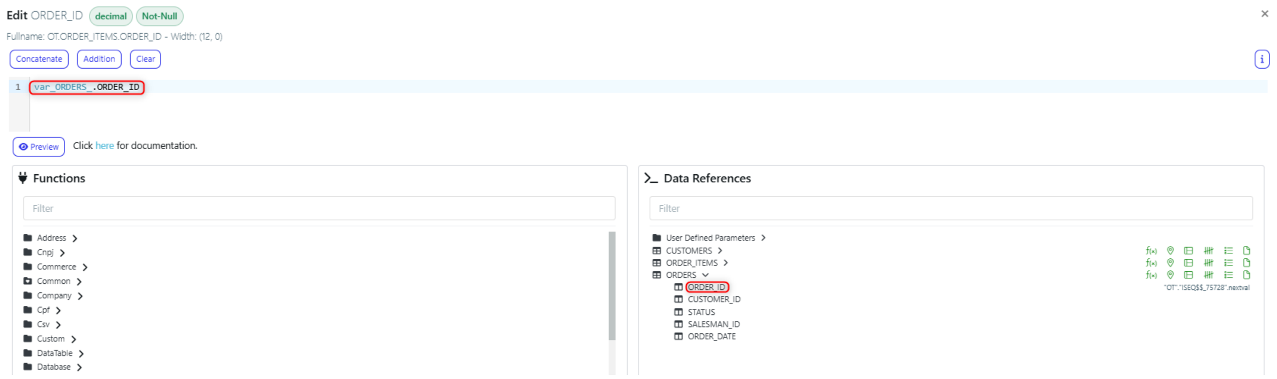
Internet
- RandomHelper.Faker.Internet.Email(firstName,lastName,provider,uniqueSuffix) — string
- RandomHelper.Faker.Internet.ExampleEmail(firstName,lastName) — string

Data References

User Defined Parameters
CUSTOMERS
EMPLOYEES
- EMPLOYEE_ID
- FIRST_NAME
- LAST_NAME
- EMAIL
- PHONE

"OT"."ISEQ$$_75713".nextval

In the scenario where a value from a table generated beforehand is required, the linked table references can be used again. The only difference here is that you need to select the column from a different table. For example, to generate data into the 'order items' table, the 'order id' from the 'orders' table can be used to make sure the data generated is referentially integral.



Edit ORDER_ID decimal Not-Null

Fullname: OT.ORDER_ITEMS.ORDER_ID - Width: (12, 0)

```
var_ORDERS_.ORDER_ID
```

Click here for documentation.

Functions

Address, Cnpj, Commerce, Common, Company, Cpf, Csv, Custom, DataTable, Database

Data References

User Defined Parameters
CUSTOMERS
ORDER_ITEMS
ORDERS
- ORDER_ID
- CUSTOMER_ID
- STATUS
- SALESMAN_ID
- ORDER_DATE

"OT"."ISEQ$$_75728".nextval

It is important to note that if the data profiling and discovery activity has been completed, the Curiosity Platform will automatically insert the necessary linked table reference to keep data referentially integral for foreign key relationships. This will also happen for soft keys if these have also been identified.

Exercise 9

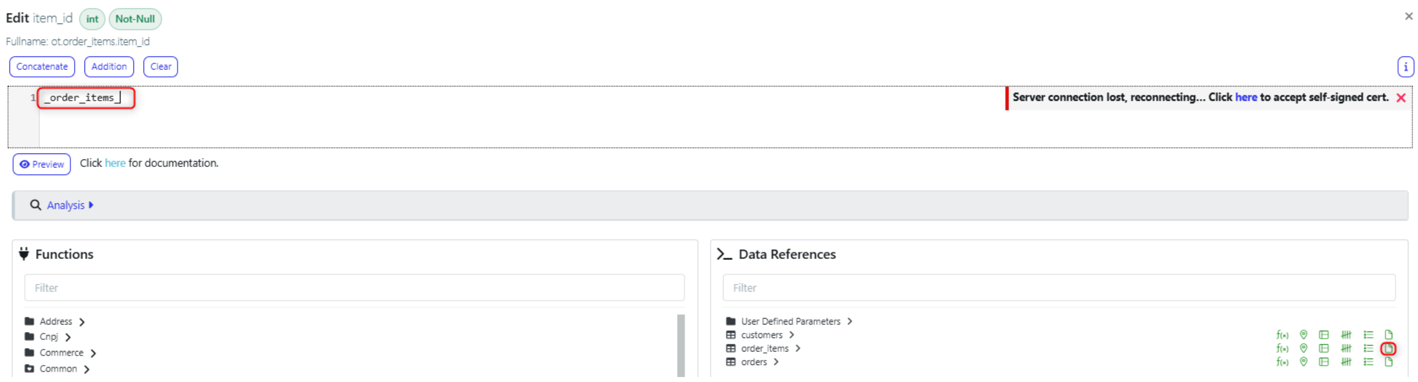
1. Use a linked table reference to populate one of the columns in the rule set

Step 5 – Data reference variables

There are several variables that are commonly used in synthetic data generation rule sets that are made available to the user for use in the creation of synthetic data generation rules. These are the most used global or local variables found in the VIP flow.

1. **Object variable**
   - The object represents the current row of data being generated
   - Data is stored in an object named var_tableName
   - To access a specific column's value within this row, use the syntax: var_tableName.columnName
     - Example: var_Employees.FirstName
2. **Local iteration variable**
   - Refers to the count of rows generated within data painter, not the original data source
   - Syntax: parCount_tableName
3. **Number of rows variable**
   - Represents the total number of rows you intend to generate
   - Syntax: parItem
   - This variable can be used in custom logic to determine the data set size dynamically
4. **List variable**
   - Refers to the list containing all object variables generated.
   - Syntax: lst_tableName
5. **Current Index**
   - Indicates the current row number being generated within the list.
   - Syntax: _tableName_

When clicking on the relevant Data Reference Variable the relevant object is inserted into the data painter screen. In the example below, we have used the reference variable 'current_index' which will give a current indicator for the 'order_items' table, and will therefore act as 'Item ID'.


Edit item_id int Not-Null
Fullname: ot.order_items.item_id

```
_order_items_
```

Server connection lost, reconnecting... Click here to accept self-signed cert.

Click here for documentation.

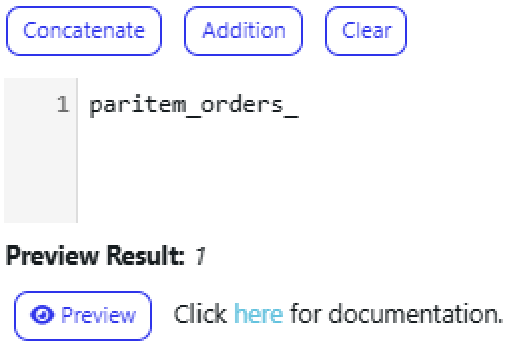
Exercise 10

1. Select a column in the rule set and use one of the data reference variables to either populate, or be part of a function that populates it

Data painter – Additional information

1. Preview

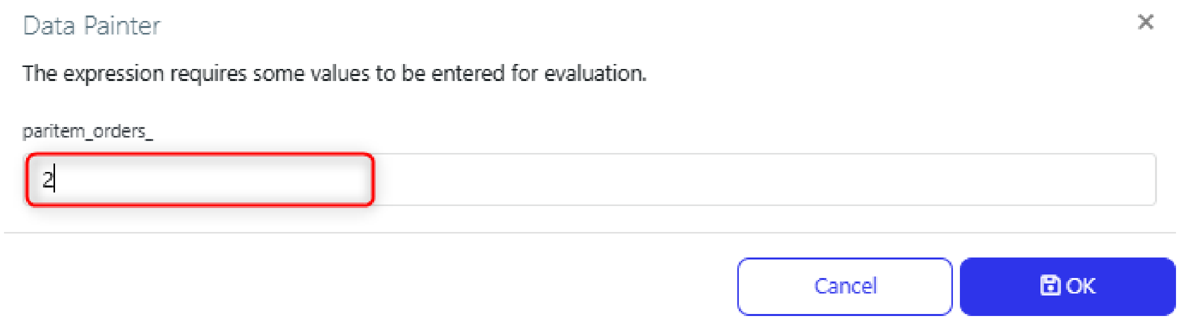
To work out whether the function you have typed out returns the data you want, the **'Preview'** button can be used.



```
paritem_orders_
```

Preview Result: *1*

If the value is not a function, but instead a parameter or variable that needs information to be inputted to be resolved, the preview requires a value to be inputted.



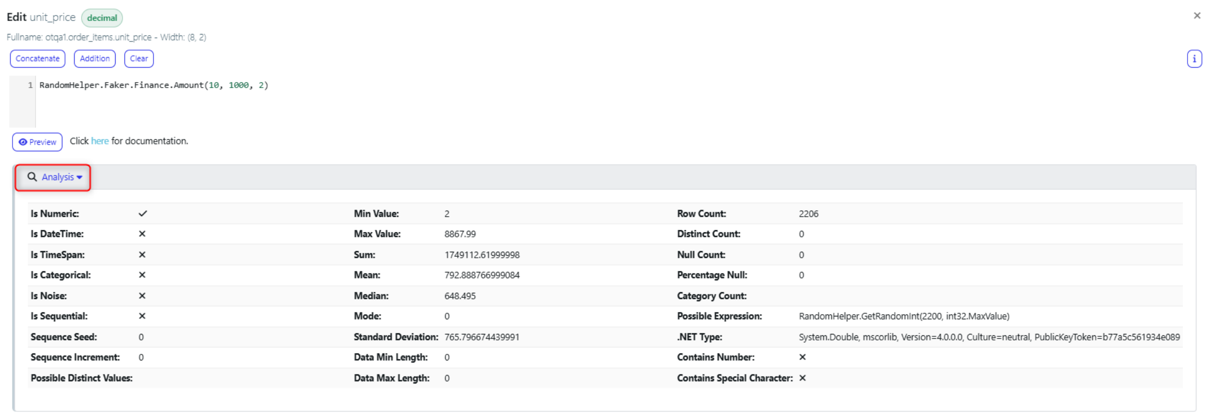
The expression requires some values to be entered for evaluation.

paritem_orders_: 2

2. Analysis

It is often helpful to have the meta data about the column you are generating into available in data painter. This information is available in the analysis section, and contains all the information captured in the deep scanning conducted in Module 1 – Data profiling and discovery.

Note: this section only appears if you have run a profiling job.



Fullname: otqa1.order_items.unit_price - Width: (8, 2)

```
RandomHelper.Faker.Finance.Amount(10, 1000, 2)
```

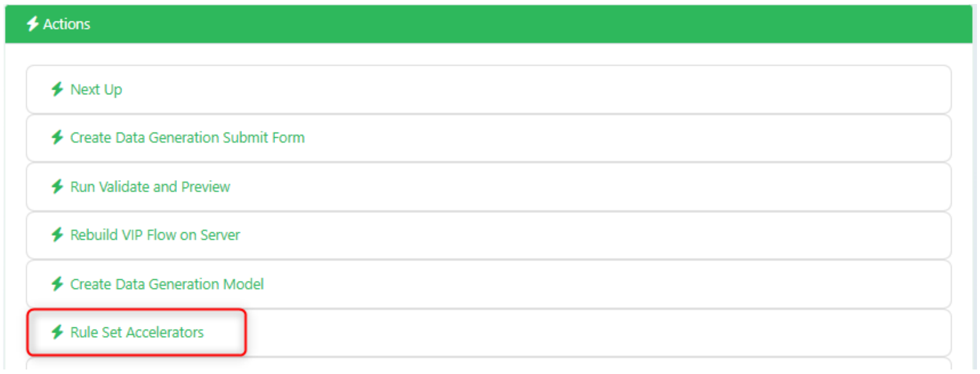
Is Numeric:	✓	**Min Value:**	2	**Row Count:**	2206
Is DateTime:	✕	**Max Value:**	8867.99	**Distinct Count:**	0
Is TimeSpan:	✕	**Sum:**	1749112.61999998	**Null Count:**	0
Is Categorical:	✕	**Mean:**	792.888766999084	**Percentage Null:**	0
Is Noise:	✕	**Median:**	648.495	**Category Count:**	
Is Sequential:	✕	**Mode:**	0	**Possible Expression:**	RandomHelper.GetRandomInt(2200, int32.MaxValue)
Sequence Seed:	0	**Standard Deviation:**	765.796674439991	**.NET Type:**	System.Double, mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
Sequence Increment:	0	**Data Min Length:**	0	**Contains Number:**	✕
Possible Distinct Values:		**Data Max Length:**	0	**Contains Special Character:**	✕

Method 3: Rule set accelerators

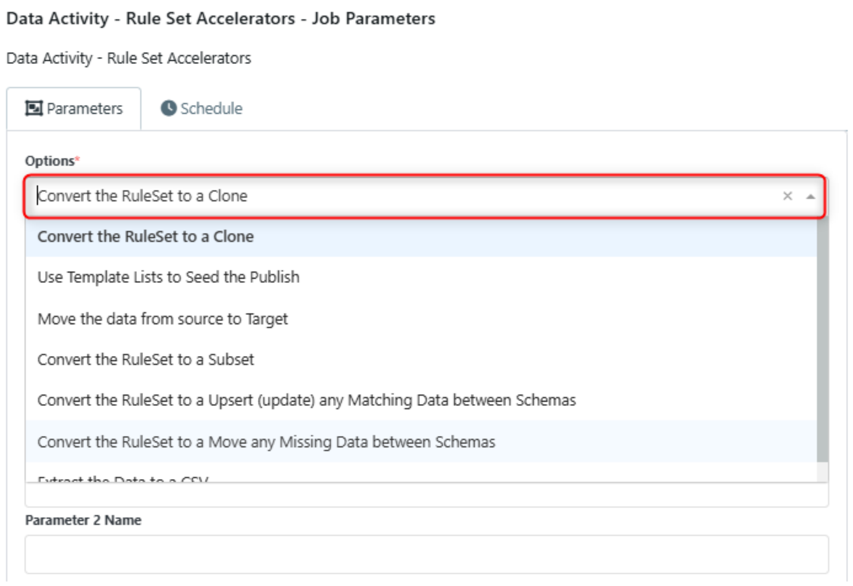
The Curiosity Platform has an extensive list of rule set accelerators that automatically configure the rule set to conduct certain actions. These include configuring the rule set to clone certain data from one source and create additional copies of it, only changing the desired fields.

Accessing the rule set accelerators

1. Scroll down to **'Actions'** and click on the '**Rule Set Accelerators'** button


Actions
Next Up
Create Data Generation Submit Form
Run Validate and Preview
Rebuild VIP Flow on Server
Create Data Generation Model
Rule Set Accelerators

2. This reveals a form that allows the user to choose from a selection of rule set accelerators:
   - **Convert the Rule Set to a Clone –** Use the ruleset to update the ruleset to run a clone
   - **Use Template Lists to Seed the Publish –** Create a template for data generation
   - **Move the data form source to Target –** Allows the rule set to be used to move data from one database to another
   - **Convert the Rule Set to a Subset –** Converts the rule set to create a subset of the data
   - **Convert the Rule Set to Upsert (update) any Matching Data between Schemas –** Upgrades the rule set to take data from one version of the table and update any differences in another
   - **Convert the Rule Set to Move any Missing Data between Schemas –** Upgrades the rule set to identify differences in the same tables between different environments and move any data that is missing
   - **Extract the Data to a CSV –** Changes the rule set to export the data into a CSV


Data Activity - Rule Set Accelerators - Job Parameters

Data Activity - Rule Set Accelerators

Options*

Convert the RuleSet to a Clone

Convert the RuleSet to a Clone
Use Template Lists to Seed the Publish
Move the data from source to Target
Convert the RuleSet to a Subset
Convert the RuleSet to a Upsert (update) any Matching Data between Schemas
Convert the RuleSet to a Move any Missing Data between Schemas
Parameter 2 Name

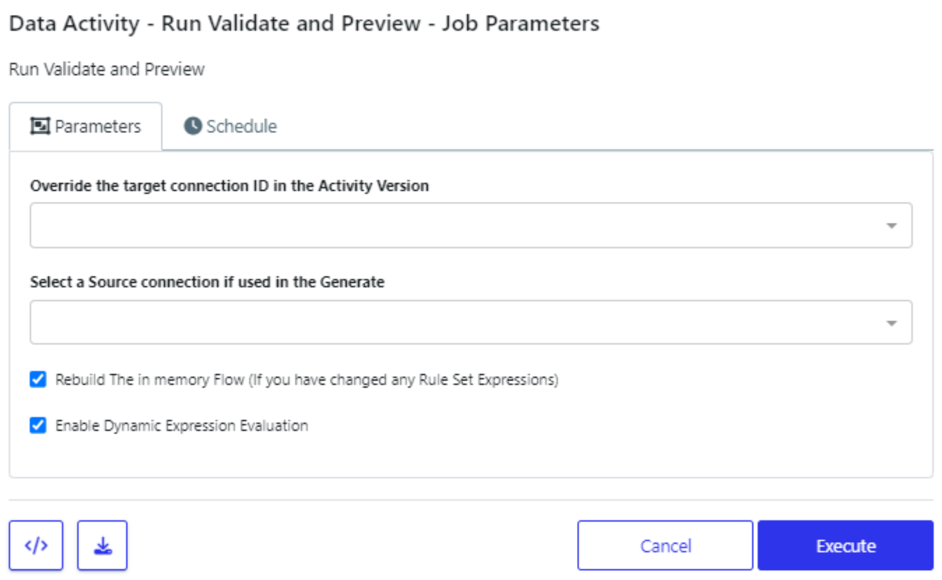
Section 4 – Execute the synthetic data generation activity

Now that you have configured the rule set, it is now time to confirm it is generating the data to meet your requirements. Once you are happy with the results, you can then build the VIP Flow.

Once the VIP flow is built, you can create a form to kick off the data generation routine whenever needed. The activity can also be included in a test data pipeline or can be kicked off by other applications. For example, the form can be kicked off by an API call, embedded in a tool like Confluence as an iFrame, or you can download a JavaScript that allows the job to be executed.

Step 1 – Validate and preview

This is to confirm that the rules create data as expected. You can leave the values on the dialogue as default and click ‘**Execute’.**


Data Activity - Run Validate and Preview - Job Parameters

Run Validate and Preview

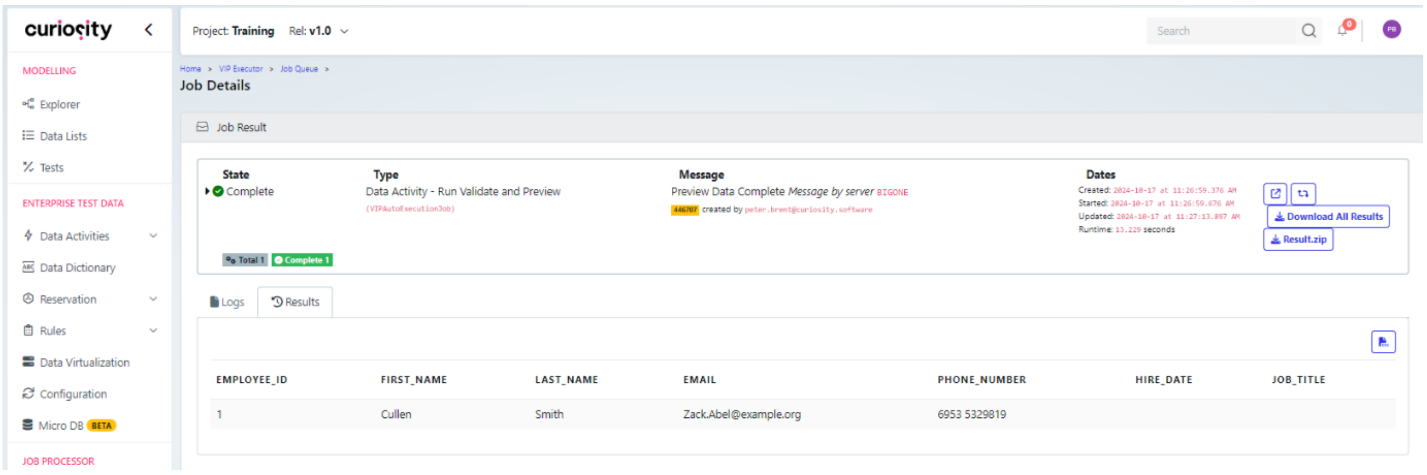
Override the target connection ID in the Activity Version

Select a Source connection if used in the Generate

☑ Rebuild The in memory Flow (If you have changed any Rule Set Expressions)

☑ Enable Dynamic Expression Evaluation

This will open a new browser tab showing the running job. When it’s complete, the sample data will be visible in the ‘Results’ tab.



Project: Training Rel: v1.0

Job Details

Job Result

State	Type	Message	Dates
Complete	Data Activity - Run Validate and Preview (VIPAutoExecutionJob)	Preview Data Complete *Message by server* BIGONE 446707 created by peter.brent@curiosity.software	Created: 2024-10-17 at 11:26:59.376 AM Started: 2024-10-17 at 11:26:59.676 AM Updated: 2024-10-17 at 11:27:13.897 AM Runtime: 13.229 seconds

Total 1 Complete 1

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_TITLE
1	Cullen	Smith	Zack.Abel@example.org	6953 5329819		

You can also download the ‘result.zip’ to see the data in a file format such as CSV.

Exercise 11

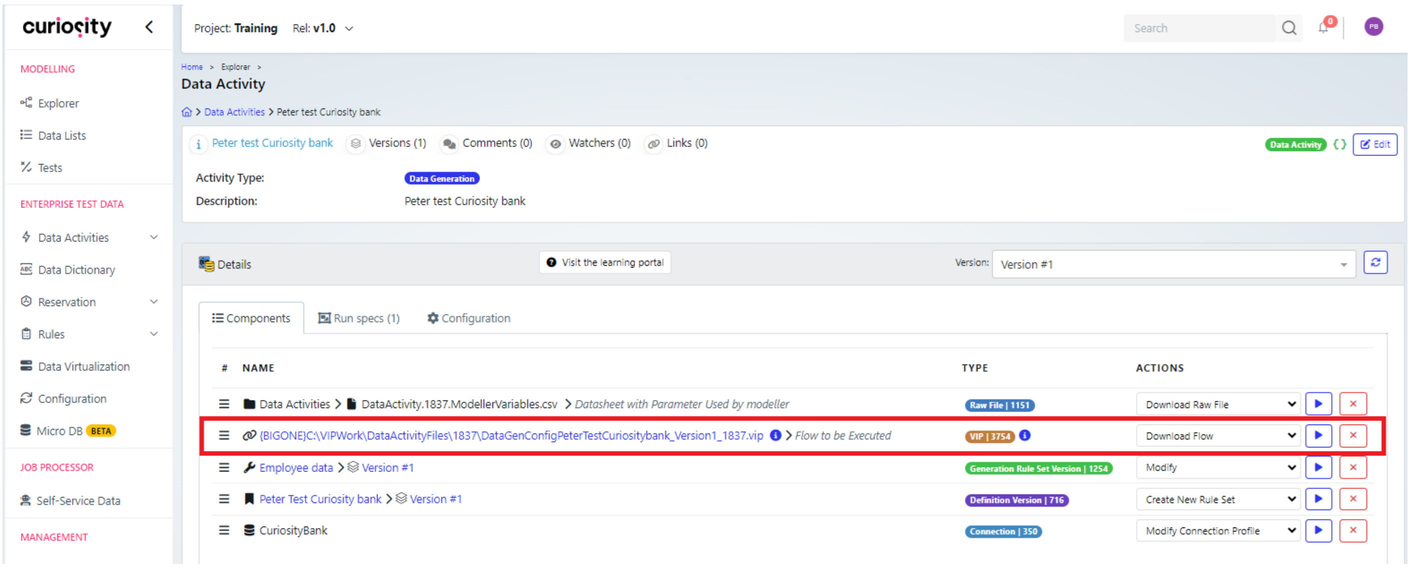
1. Run the preview and validate action on your ruleset to check the results being produced by the rule set

Step 2 – Rebuild the VIP flow on the server

This action compiles the rule set into a VIP flow so that it can be executed on demand to produce synthetic data. If there are any compilation issues, these will be captured and produced in a report that can be downloaded.

If there is a compilation error, the results section of the job details will tell you how many have occurred. You can then download the report and address any compilation issues.

Once it successfully completes, the flow will be added as a **'component'**.



Project: Training Rel: v1.0

Home > Explorer >
Data Activity
Data Activities > Peter test Curiosity bank

Peter test Curiosity bank | Versions (1) | Comments (0) | Watchers (0) | Links (0)

Activity Type: Data Generation
Description: Peter test Curiosity bank

Details | Visit the learning portal | Version: Version #1

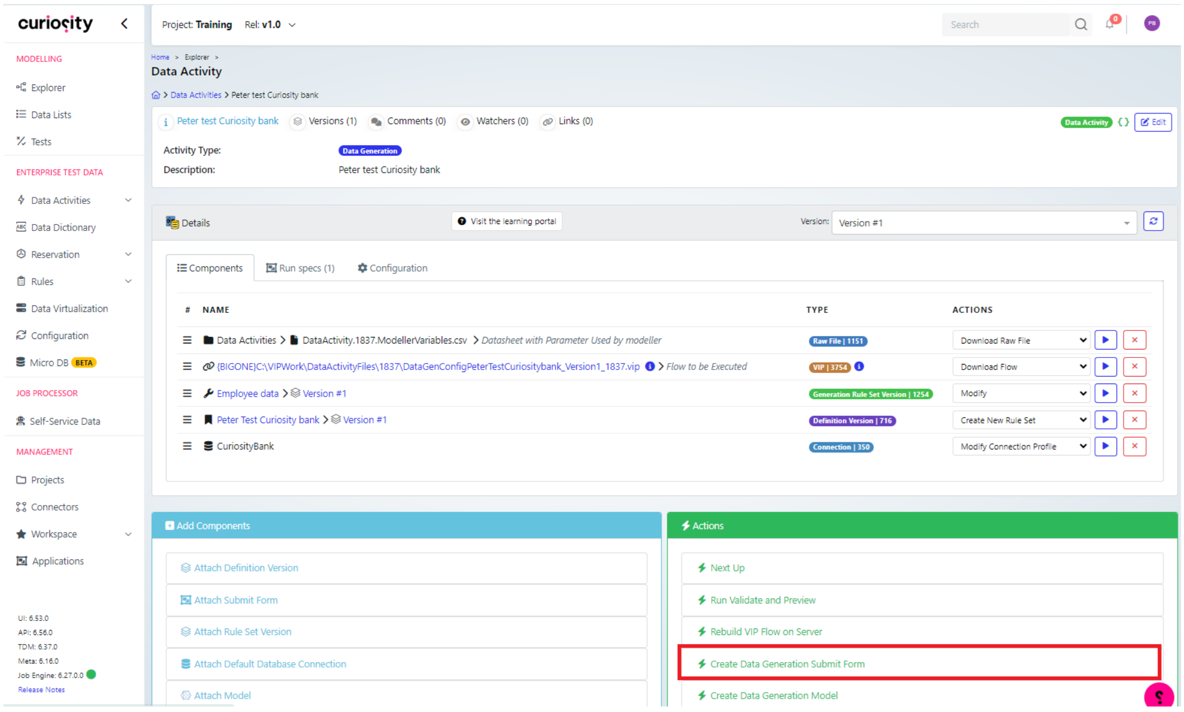
Components | Run specs (1) | Configuration

#	NAME	TYPE	ACTIONS
☰	Data Activities > DataActivity.1837.ModellerVariables.csv > *Datasheet with Parameter Used by modeller*	Raw File | 1151	Download Raw File
☰	(BIGONE)C:\VIPWork\DataActivityFiles\1837\DataGenConfigPeterTestCuriositybank_Version1_1837.vip > *Flow to be Executed*	VIP | 3754	Download Flow
☰	Employee data > Version #1	Generation Rule Set Version | 1254	Modify
☰	Peter Test Curiosity bank > Version #1	Definition Version | 716	Create New Rule Set
☰	CuriosityBank	Connection | 350	Modify Connection Profile

Step 3 – Create a data generation submit form

To generate data, you need to create a submit form. This will allow anyone with appropriate permissions in your organisation to create the data.

On the data generation activity page, click the '**Create Data Generation Submit Form**' action.



Add Components
- Attach Definition Version
- Attach Submit Form
- Attach Rule Set Version
- Attach Default Database Connection
- Attach Model

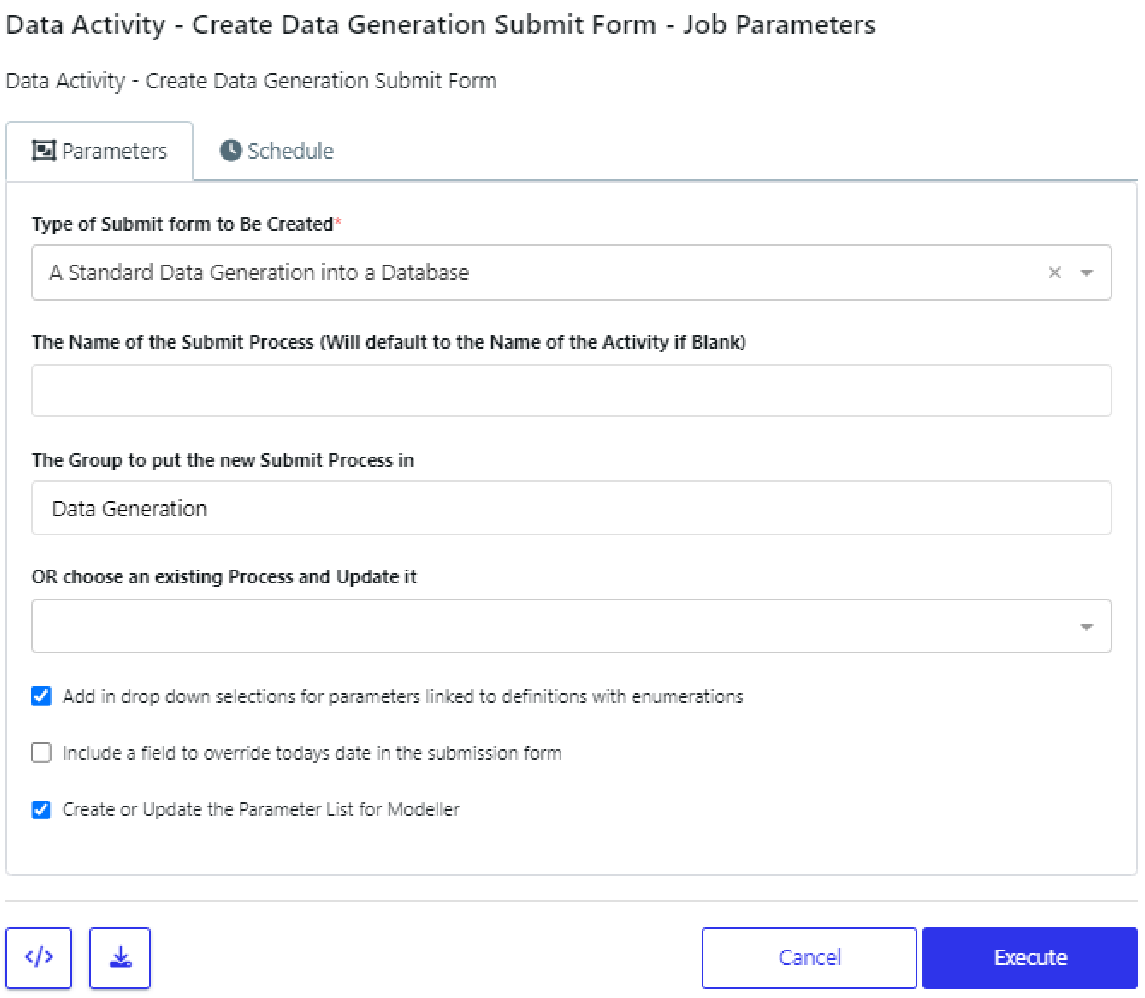
Actions
- Next Up
- Run Validate and Preview
- Rebuild VIP Flow on Server
- Create Data Generation Submit Form
- Create Data Generation Model

UI: 6.53.0
API: 6.56.0
TDM: 6.37.0
Meta: 6.16.0
Job Engine: 6.27.0.0

This will display the 'Create Data Generation Submit Form' dialogue box:

- This is a job scheduling dialogue box, so you can schedule the job to run at a different time, if need, on the 'Schedule' tab
- You can generate a CSV file instead by changing the 'Type of Submit form to Be Created' in the dropdown
- You can also use this to update an existing submit form, by selecting the form in the 'Choose an existing process and Update it' field

When you click **'Execute'**, a new browser tab will open, so you can see the job to create the form being processed.


Data Activity - Create Data Generation Submit Form - Job Parameters

Data Activity - Create Data Generation Submit Form

Type of Submit form to Be Created*: A Standard Data Generation into a Database

The Name of the Submit Process (Will default to the Name of the Activity if Blank):

The Group to put the new Submit Process in: Data Generation

OR choose an existing Process and Update it:

[✓] Add in drop down selections for parameters linked to definitions with enumerations

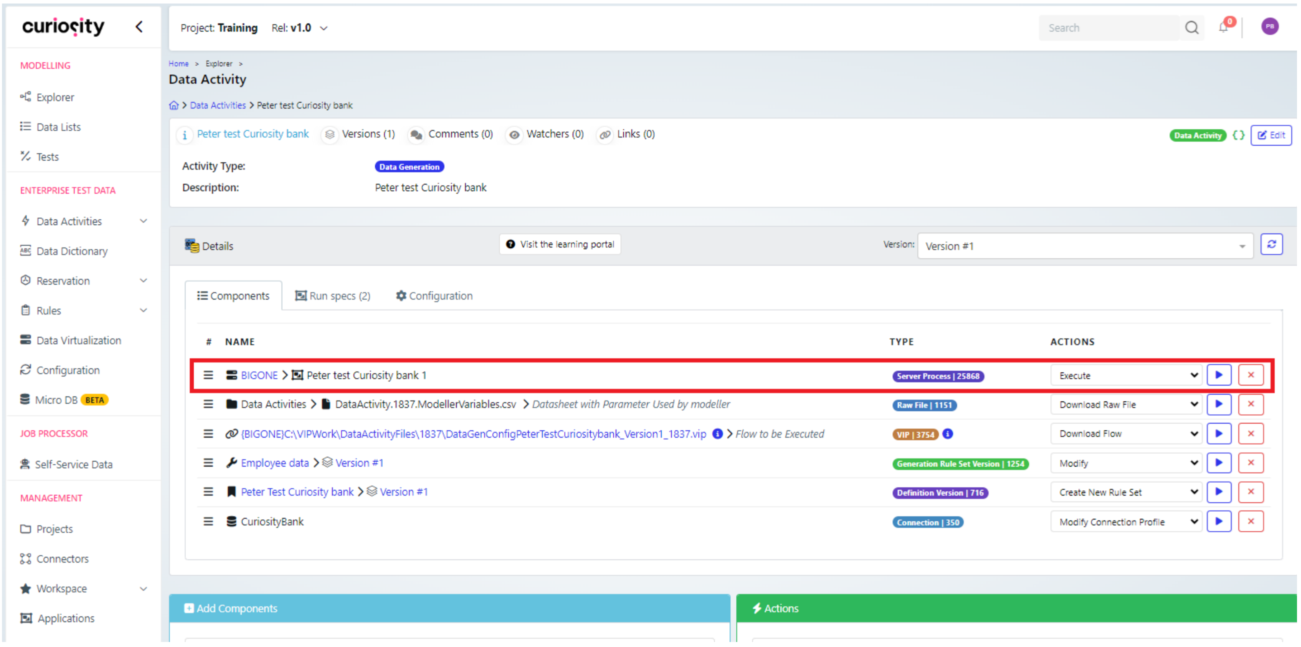
[ ] Include a field to override todays date in the submission form

[✓] Create or Update the Parameter List for Modeller

Once the job successfully completes and you refresh the browser tab, you should see the submit form appear as a component in the data activity.

Step 4 – Run the data generation submit form

To run the form, make sure the ‘Action’ is set to ‘Execute’ for the form, then click the blue arrow:


Activity Type: Data Generation
Description: Peter test Curiosity bank

Version: Version #1

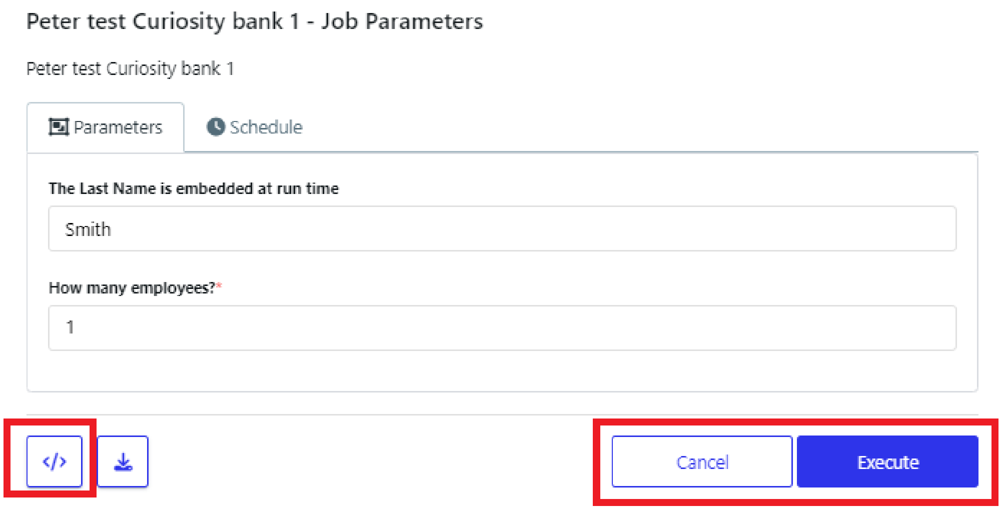
#	NAME	TYPE	ACTIONS
	BIGONE > Peter test Curiosity bank 1	Server Process | 25868	Execute
	Data Activities > DataActivity.1837.ModellerVariables.csv > *Datasheet with Parameter Used by modeller*	Raw File | 1151	Download Raw File
	{BIGONE}C:\VIPWork\DataActivityFiles\1837\DataGenConfigPeterTestCuriositybank_Version1_1837.vip > *Flow to be Executed*	VIP | 3754	Download Flow
	Employee data > Version #1	Generation Rule Set Version | 1254	Modify
	Peter Test Curiosity bank > Version #1	Definition Version | 716	Create New Rule Set
	CuriosityBank	Connection | 350	Modify Connection Profile

When you run the form it will create a job, which in turn will create data in your chosen database.

You can change the number of records created and any parameters that you exposed on the activity. In this example, we added ‘Last Name’.

The </> button exposes the form as an API call, iFrame or JavaScript to be executed how you would like.

Click **‘Execute’** to run the job.


Peter test Curiosity bank 1 - Job Parameters

Peter test Curiosity bank 1

The Last Name is embedded at run time: Smith

How many employees?*: 1

Exercise 12

1. Create the VIP flow and the submit form, and run it to create data
2. Check your results to see the data and check it has been created as you would expect

Check the solution videos for all Exercises in this course >