[Run it on Codepen]

[Rule project files for importing into Corticon.js Studio]

Decision Management Community Challenge March-2024

Submitted by: Seth Meldon

Decision Modeling Tool: Progress Corticon.js

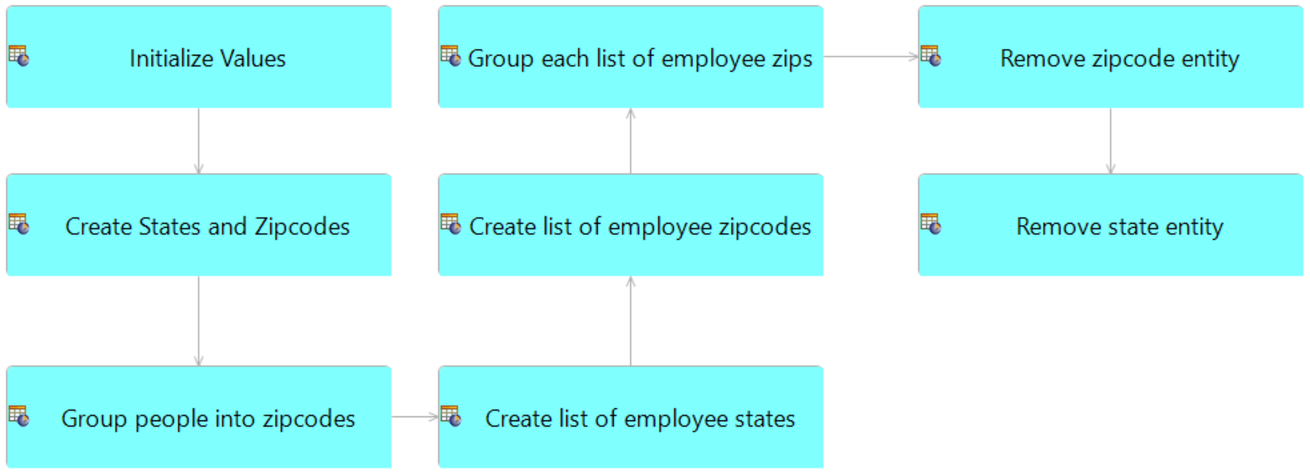
Prompt

Decision models (similarly to databases) frequently deal with analysis of collections of objects. Here is an example. Let's help an HR office create a rules-based service to analyze its employees. Each employee has a unique name, age, gender, marital status, locations (places of residence), number of children, salary, and probably more attributes. This information is coming to the service as a JSON request such as in this file. Your service should find answers to the following questions:

- What is the current total number of employees?
- How many children all employees have? How many children does the average employee have?
- What is an average salary? What is the maximal and minimal salaries?
- How many employees are single?
- In which states do the employee have residences?
- How many people are inside 20% of highest paid employees? Who are these high-paid employees?

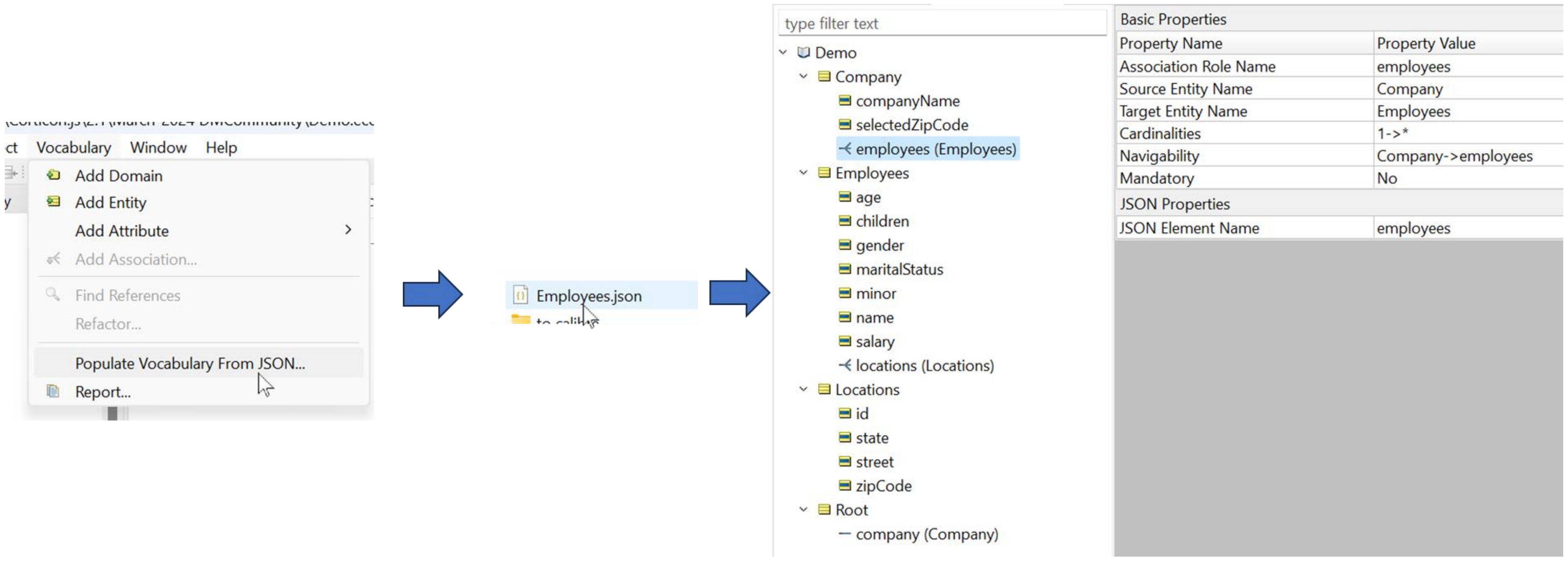
Approach Taken

All rules are defined rulesheets, the blue nodes in the screenshot below, sequenced into a Ruleflow. The ruleflow is what is generated into a self-contained JavaScript decision service file.


Initialize Values
Create States and Zipcodes
Group people into zipcodes
Create list of employee states
Create list of employee zipcodes
Group each list of employee zips
Remove zipcode entity
Remove state entity

1 - Generate Rule Vocabulary from Employees.json

The structure of the JSON in the provided JSON file is downloaded, and generated into the Corticon.js Rule Vocabulary.



Basic Properties

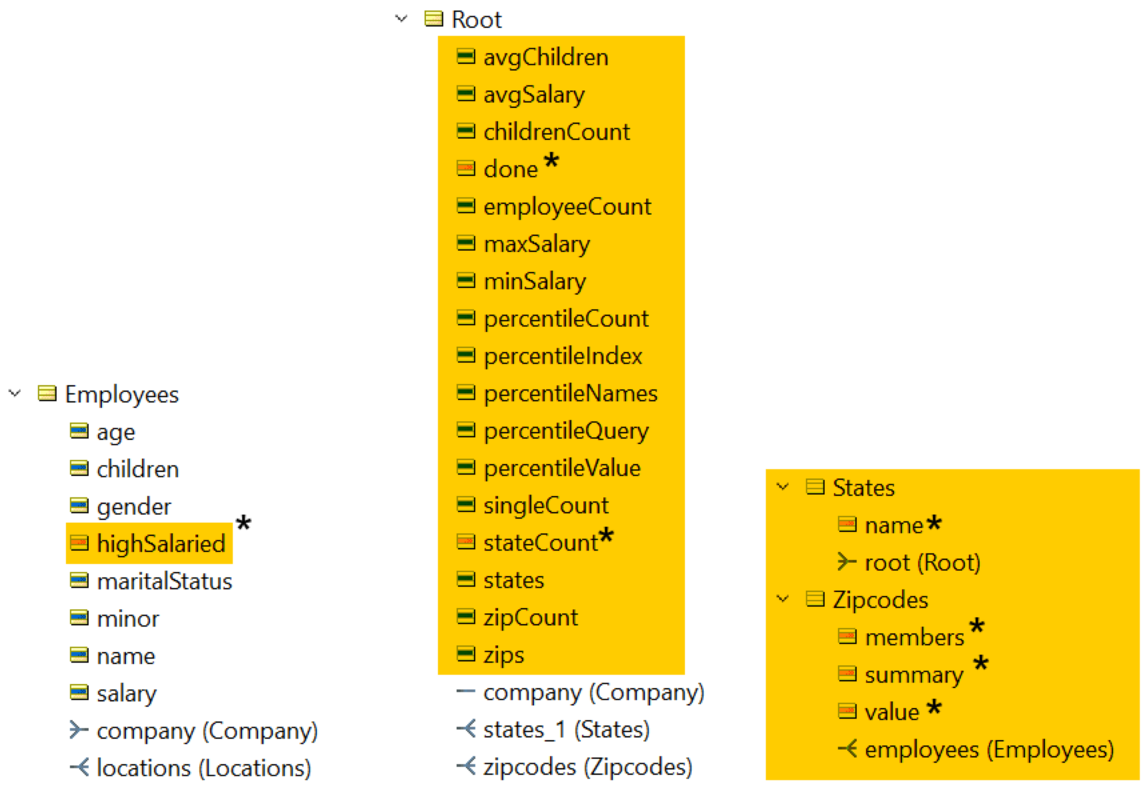
Property Name	Property Value
Association Role Name	employees
Source Entity Name	Company
Target Entity Name	Employees
Cardinalities	1->*
Navigability	Company->employees
Mandatory	No

JSON Properties

JSON Element Name	employees

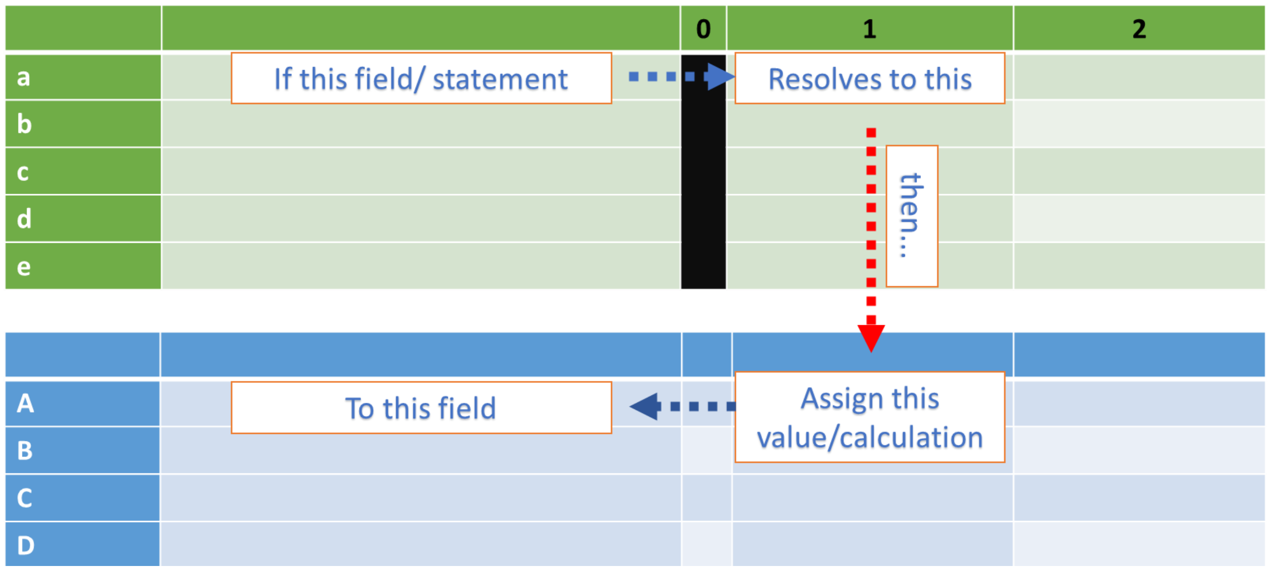
2 - Add any needed additional fields to rule vocabulary

The JSON didn't already have fields needed to store the values that are being solved for (average salary, number of children etc), so these are added to the generated vocabulary. Shown below, the added vocabulary attributes are highlighted. The vocabulary attributes with an asterisk next to them are transient attributes. Transient attributes are used as “intermediate” value holders that do not need to be returned in a response.

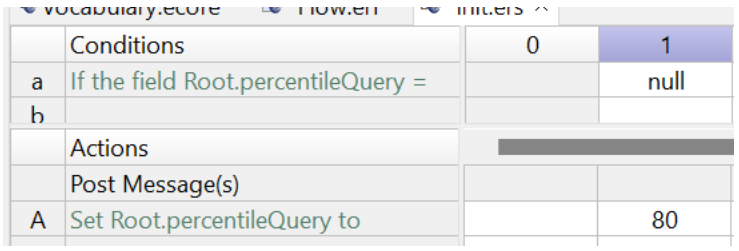


3 - Rulesheets specify rules to change the data and create new data elements

Corticon rules are modeled in decision tables by dragging and dropping elements of the vocabulary onto a table and defining any number of conditions which when met result in any number of actions:



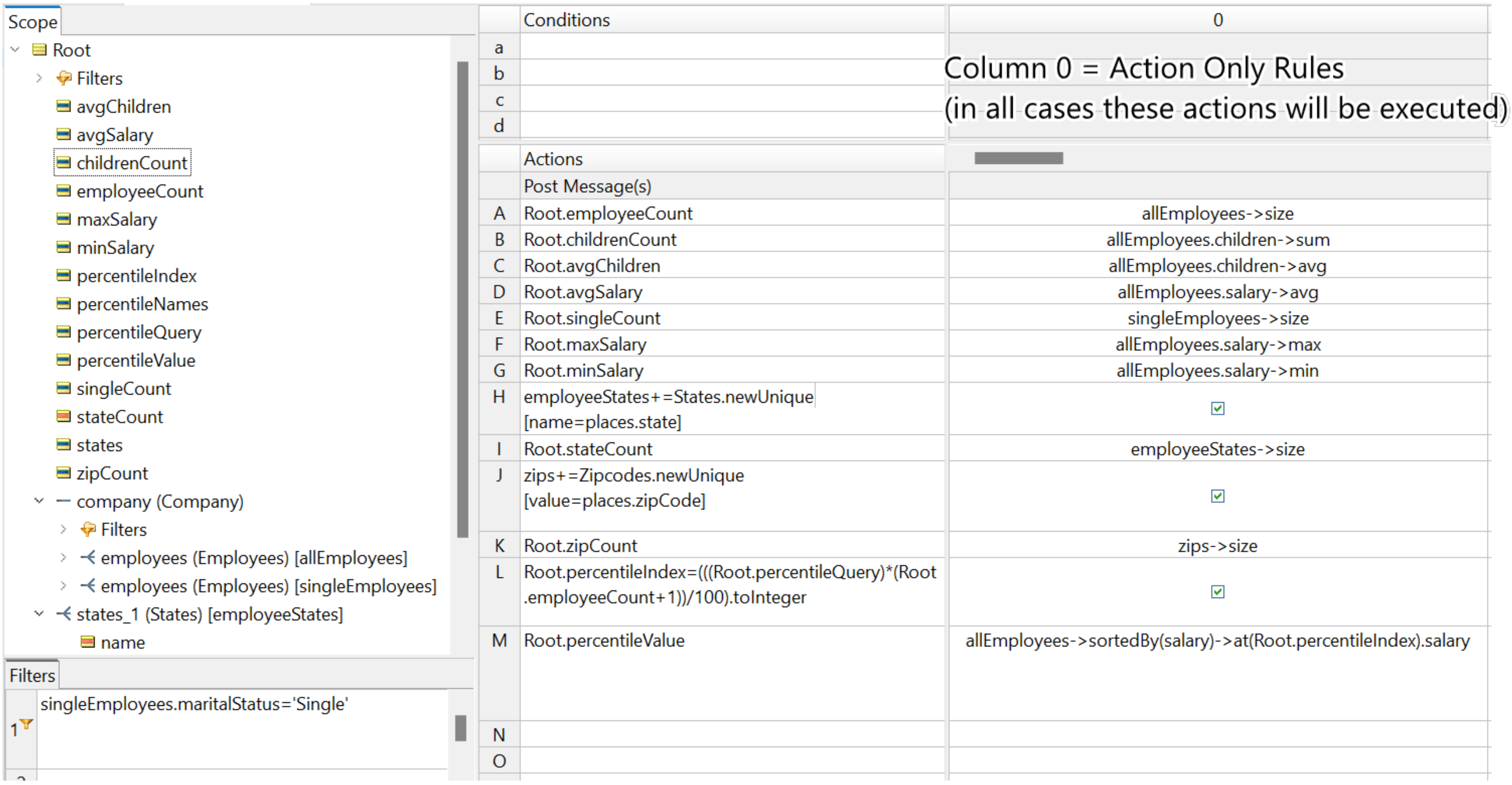
Rulesheet 1 – Specifies default values if not provided in input payload (if desired percentile value is not passed in up front, then assign the percent to be the value specified in the prompt of 80%).



	Conditions	0	1
a	If the field Root.percentileQuery =		null
b			
	Actions		
	Post Message(s)		
A	Set Root.percentileQuery to		80

Rulesheet 2 –

Here's where things start getting interesting. Here, we are starting to define rules that apply to *collections*, not just individual entities.



Scope
- Root
  - Filters
  - avgChildren
  - avgSalary
  - childrenCount
  - employeeCount
  - maxSalary
  - minSalary
  - percentileIndex
  - percentileNames
  - percentileQuery
  - percentileValue
  - singleCount
  - stateCount
  - states
  - zipCount
  - company (Company)
    - Filters
    - employees (Employees) [allEmployees]
    - employees (Employees) [singleEmployees]
  - states_1 (States) [employeeStates]
    - name

Filters

1	singleEmployees.maritalStatus='Single'

	Conditions	0
a		
b		
c		
d		

Column 0 = Action Only Rules
(in all cases these actions will be executed)

	Actions	
	Post Message(s)	
A	Root.employeeCount	allEmployees->size
B	Root.childrenCount	allEmployees.children->sum
C	Root.avgChildren	allEmployees.children->avg
D	Root.avgSalary	allEmployees.salary->avg
E	Root.singleCount	singleEmployees->size
F	Root.maxSalary	allEmployees.salary->max
G	Root.minSalary	allEmployees.salary->min
H	employeeStates+=States.newUnique [name=places.state]	✓
I	Root.stateCount	employeeStates->size
J	zips+=Zipcodes.newUnique [value=places.zipCode]	✓
K	Root.zipCount	zips->size
L	Root.percentileIndex=(((Root.percentileQuery)*(Root .employeeCount+1))/100).toInteger	✓
M	Root.percentileValue	allEmployees->sortedBy(salary)->at(Root.percentileIndex).salary
N		
O		

Filter rows:

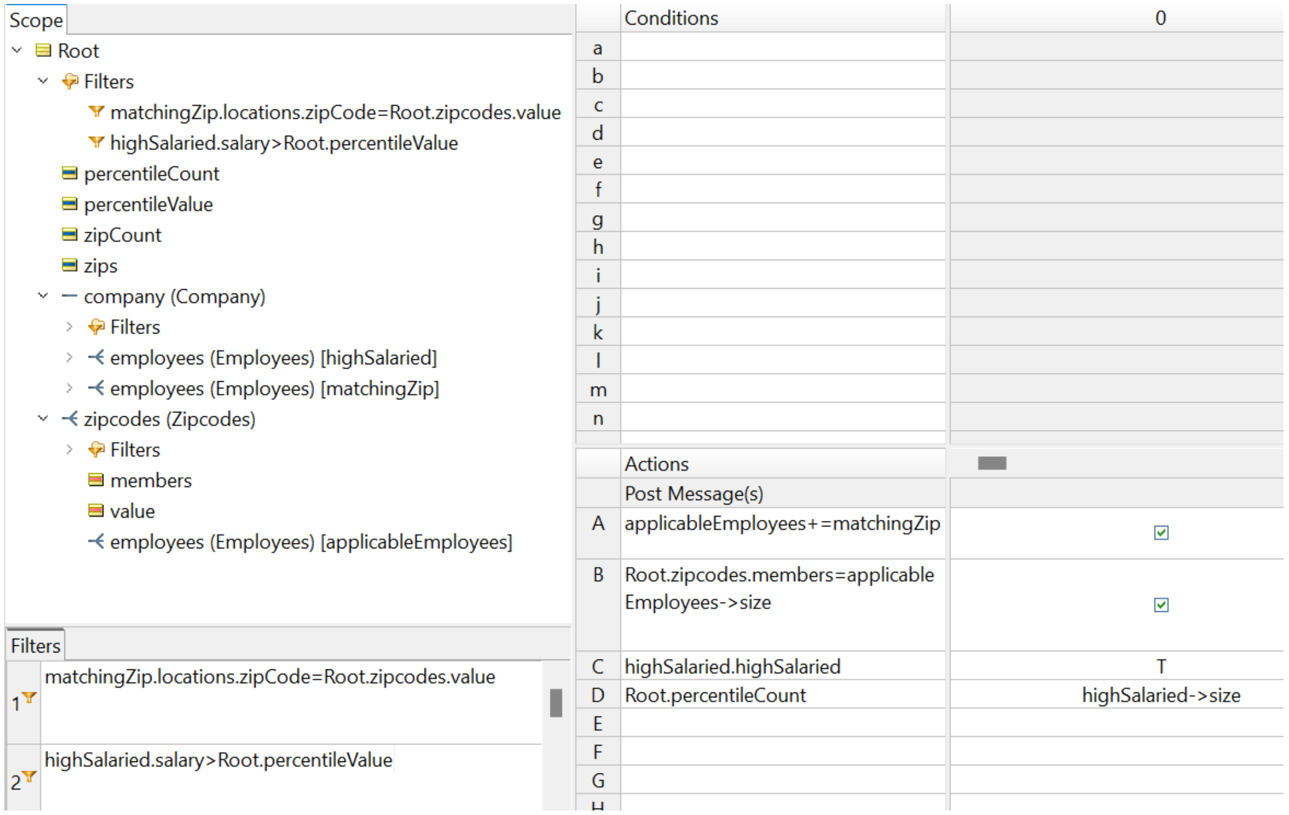
1. Create alias Single for all instances of `Root.company.employees` where `Employee.maritalStatus = 'Single'`

Action Rows:

A) Set the current total number of employees (`Root.employeeCount`) to count the number of individual employee records

B) Set total children for all employees (`Root.childrenCount`) to the sum each employee's children
C) Set the average children per employee (`Root.avgChildren`) to be the average of all employees' children
D) Set the average salary per employee (`Root.avgSalary`) to be the average of all employee salaries
E) Set the count of single employees (`Root.singleCount`) to be the size of the collection of employees in the collection alias Single
F) Set employees' max salary (`Root.maxSalary`) to be the maximum of all values for `Employee.salary`
G) Set employees' min salary (`Root.minSalary`) to be the minimum of all values for `Employee.salary`
H) Create unique entities (no duplicates) of all states where employees live
I) Set the `Root.stateCount` field to the size of all state entities
J) Create unique entities (no duplicates) of all zip codes where employees live
K) Set the `Root.zipCount` field the size of all zip code entities
L) Set `Root.percentileIndex` to the output of (`Root.percentileQuery` /100) * (`Root.employeeCount`+1)
M) Sort employees by salary, and set `Root.percentileValue` to the salary of the employee that is the value of `Root.percentileIndex` in the list

Rulesheet 3:



Scope

```
Root
  Filters
    matchingZip.locations.zipCode=Root.zipcodes.value
    highSalaried.salary>Root.percentileValue
  percentileCount
  percentileValue
  zipCount
  zips
  company (Company)
    Filters
    employees (Employees) [highSalaried]
    employees (Employees) [matchingZip]
  zipcodes (Zipcodes)
    Filters
    members
    value
    employees (Employees) [applicableEmployees]
```

Filters

1	matchingZip.locations.zipCode=Root.zipcodes.value
2	highSalaried.salary>Root.percentileValue

	Conditions	0
a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		

	Actions	
	Post Message(s)	
A	applicableEmployees+=matchingZip	✓
B	Root.zipcodes.members=applicable Employees->size	✓
C	highSalaried.highSalaried	T
D	Root.percentileCount	highSalaried->size
E		
F		
G		
H		

Filter rows:

1. Create alias of matchingZip for all instances of `Root.company.employee` with the same `zipcode` as the newly create zipcode entity's attribute `value`.
2. Create alias of highSalaried for all instances of `Root.company.employee` greater than `Root.percentileValue`

Action Rows:

A) Add all members of the collection with the alias matchingZip to the collection with the alias applicableEmployees

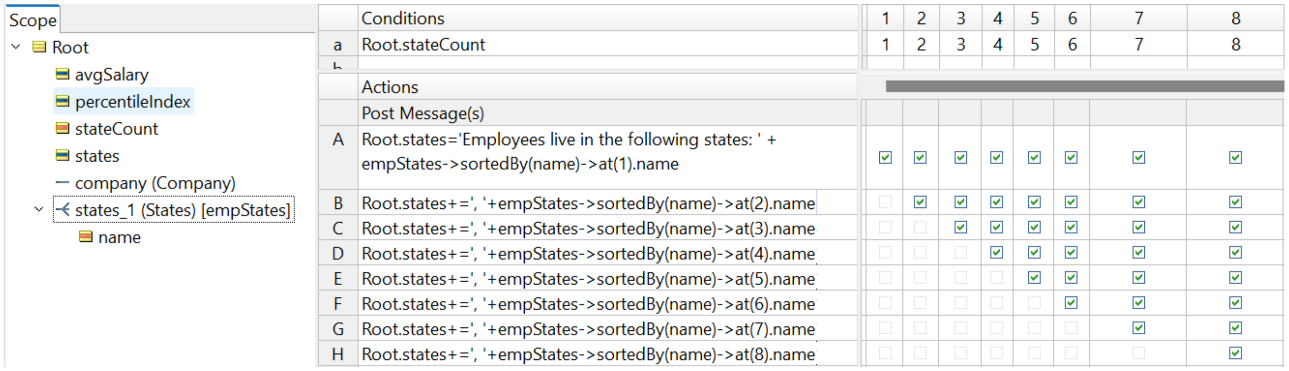
B) Set the value of `Root.zipcodes.members` to be the size of the employees in the applicableEmployees collection alias

C) Set the boolean field `Employees.highSalaried` to true for all instances of the Employees entity that meet the criteria for the highSalaried alias

D) Set `Root.percentileCount` to the size of all employees for which `Employees.highSalaried = true`

Rulesheet 4:

Here, we're building out a sentence dynamically to create a comma separated list of each state in which employees live. The higher the number of states, the more content that gets added tot the sentence.

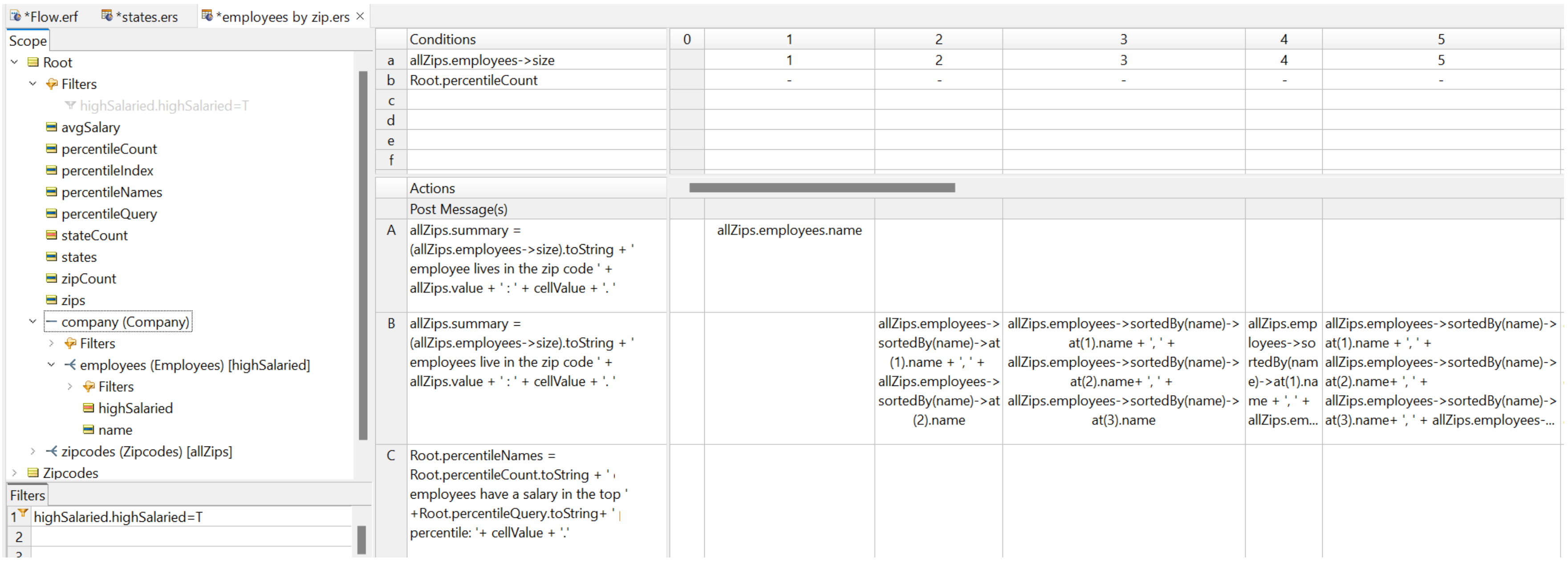


```
Scope
Root
    avgSalary
    percentileIndex
    stateCount
    states
    company (Company)
    states_1 (States) [empStates]
        name
```

	Conditions	1	2	3	4	5	6	7	8
a	Root.stateCount	1	2	3	4	5	6	7	8
	Actions								
	Post Message(s)								
A	Root.states='Employees live in the following states: ' + empStates->sortedBy(name)->at(1).name	✓	✓	✓	✓	✓	✓	✓	✓
B	Root.states+=', '+empStates->sortedBy(name)->at(2).name		✓	✓	✓	✓	✓	✓	✓
C	Root.states+=', '+empStates->sortedBy(name)->at(3).name			✓	✓	✓	✓	✓	✓
D	Root.states+=', '+empStates->sortedBy(name)->at(4).name				✓	✓	✓	✓	✓
E	Root.states+=', '+empStates->sortedBy(name)->at(5).name					✓	✓	✓	✓
F	Root.states+=', '+empStates->sortedBy(name)->at(6).name						✓	✓	✓
G	Root.states+=', '+empStates->sortedBy(name)->at(7).name							✓	✓
H	Root.states+=', '+empStates->sortedBy(name)->at(8).name								✓

Rulesheet 5:

Here, we're again dynamically building out lists based upon the number of members of collections. The collection of employees for whom `highSalaried=T` is classified again into an alias highSalaried. Note that in the top left pane that the field under 'Filters' is grayed out—this is because after we created this alias, we disabled it from acting like a filter. We call this a 'limited filter'—basically it will won't eliminate any data from evaluation by Corticon in this rulesheet, but simply allows us to refer to this subset of Employees without filtering out the Employees that are not part of this collection.


*Flow.erf   *states.ers   *employees by zip.ers

Scope
```
Root
  Filters
    highSalaried.highSalaried=T
  avgSalary
  percentileCount
  percentileIndex
  percentileNames
  percentileQuery
  stateCount
  states
  zipCount
  zips
  company (Company)
    Filters
    employees (Employees) [highSalaried]
      Filters
      highSalaried
      name
  zipcodes (Zipcodes) [allZips]
Zipcodes
```

Filters

1	highSalaried.highSalaried=T
2	

	Conditions	0	1	2	3	4	5
a	allZips.employees->size		1	2	3	4	5
b	Root.percentileCount		-	-	-	-	-
c							
d							
e							
f							
	Actions						
	Post Message(s)						
A	allZips.summary = (allZips.employees->size).toString + ' employee lives in the zip code ' + allZips.value + ' : ' + cellValue + '. '		allZips.employees.name				
B	allZips.summary = (allZips.employees->size).toString + ' employees live in the zip code ' + allZips.value + ' : ' + cellValue + '. '			allZips.employees->sortedBy(name)->at(1).name + ', ' + allZips.employees->sortedBy(name)->at(2).name	allZips.employees->sortedBy(name)->at(1).name + ', ' + allZips.employees->sortedBy(name)->at(2).name+ ', ' + allZips.employees->sortedBy(name)->at(3).name	allZips.employees->sortedBy(name)->at(1).name + ', ' + allZips.em...	allZips.employees->sortedBy(name)->at(1).name + ', ' + allZips.employees->sortedBy(name)->at(2).name+ ', ' + allZips.employees->sortedBy(name)->at(3).name+ ', ' + allZips.employees-...
C	Root.percentileNames = Root.percentileCount.toString + ' employees have a salary in the top ' +Root.percentileQuery.toString+ ' percentile: '+ cellValue + '.'						

Filter rows:

1. Create alias of highSalaried for all instances of `Root.company.employee` where the boolean attribute `highSalaried=T`

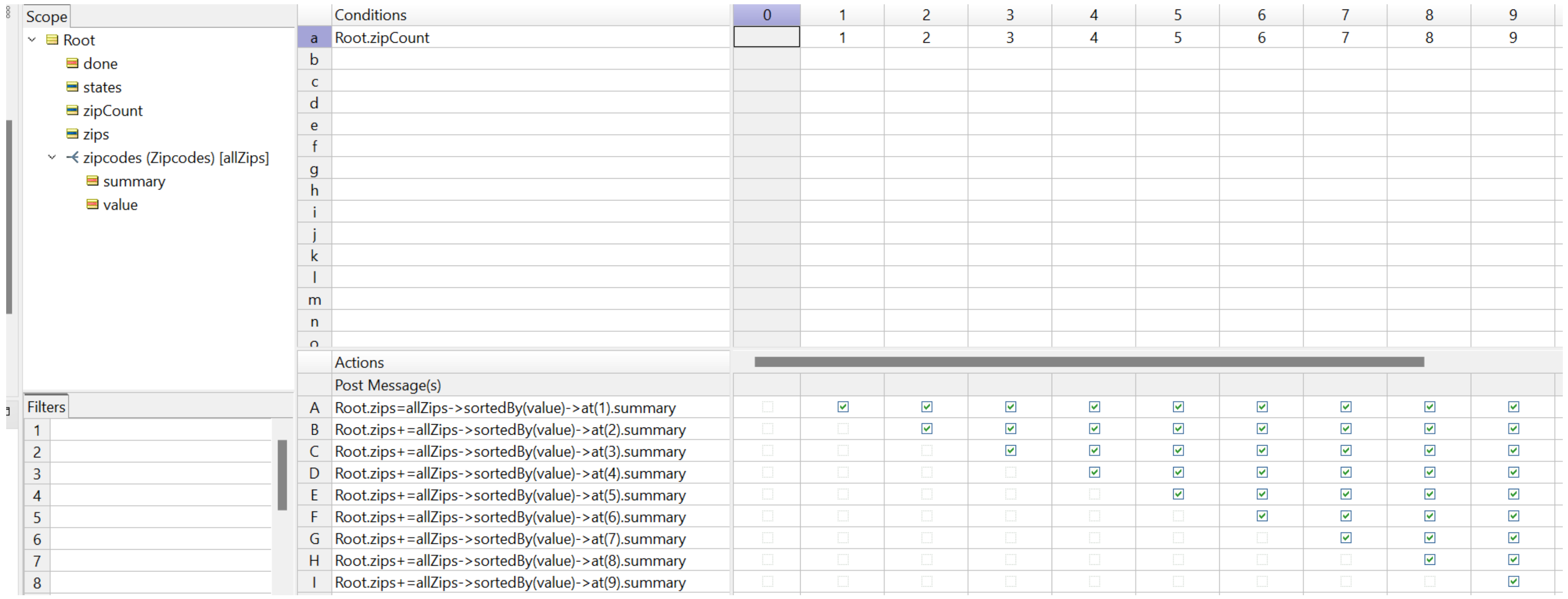
Action Rows:

A) If the number of employees associated with a given zip code entity is 1, assign that zipcode's `Zipcodes.summary` field to be 'One employee lives in the zip code [`Zipcodes.value`] : [*name of the one employee associated with this zipcode*]'

B) If the number of employees associated with a given zip code entity is 2, assign that zipcode's `Zipcodes.summary` field to be 'Two employee live in the zip code [`Zipcodes.value`] : [*name of the first employee associated with this zipcode*], [*name of the second employee associated with this zipcode*]'… **This pattern continues for up to 9 employees per zip code.**

C) If the number of employees in the collection highSalaried entity is [1,2,3,4,5], set `Root.percentileNames` field to be '[1,2,3,4,5] employees have a salary in the top `Root.percentileQuery` percentile: [Name of employee 1 of highSalaried collection, … name of employee 5 in `highSalaried` collection]'

Rulesheet 6:



Scope
- Root
  - done
  - states
  - zipCount
  - zips
  - zipcodes (Zipcodes) [allZips]
    - summary
    - value

	Filters
1	
2	
3	
4	
5	
6	
7	
8	

	Conditions	0	1	2	3	4	5	6	7	8	9
a	Root.zipCount		1	2	3	4	5	6	7	8	9
b											
c											
d											
e											
f											
g											
h											
i											
j											
k											
l											
m											
n											

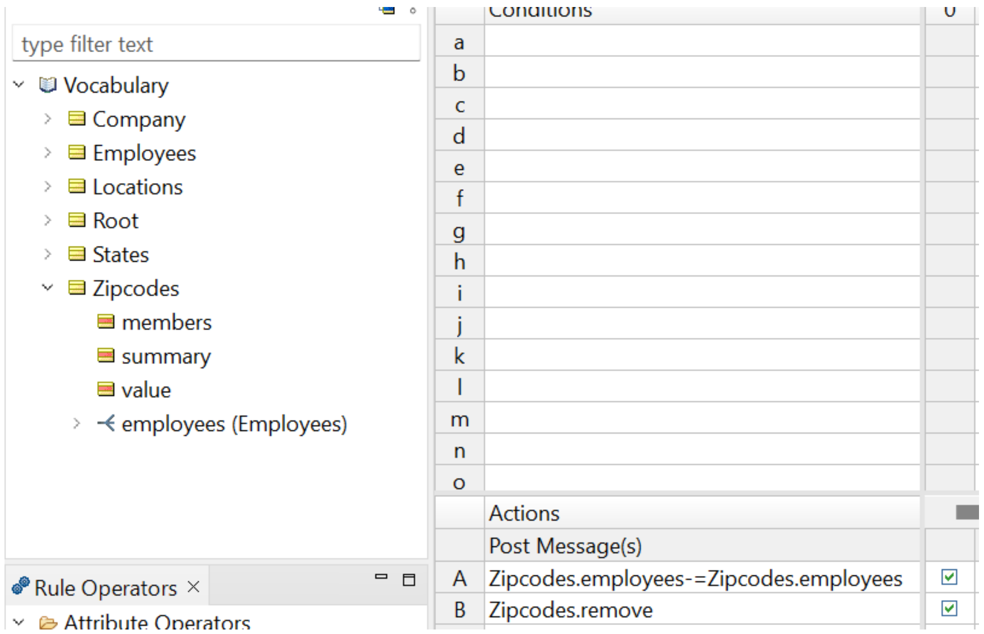
	Actions	0	1	2	3	4	5	6	7	8	9
	Post Message(s)										
A	Root.zips=allZips->sortedBy(value)->at(1).summary		✓	✓	✓	✓	✓	✓	✓	✓	✓
B	Root.zips+=allZips->sortedBy(value)->at(2).summary			✓	✓	✓	✓	✓	✓	✓	✓
C	Root.zips+=allZips->sortedBy(value)->at(3).summary				✓	✓	✓	✓	✓	✓	✓
D	Root.zips+=allZips->sortedBy(value)->at(4).summary					✓	✓	✓	✓	✓	✓
E	Root.zips+=allZips->sortedBy(value)->at(5).summary						✓	✓	✓	✓	✓
F	Root.zips+=allZips->sortedBy(value)->at(6).summary							✓	✓	✓	✓
G	Root.zips+=allZips->sortedBy(value)->at(7).summary								✓	✓	✓
H	Root.zips+=allZips->sortedBy(value)->at(8).summary									✓	✓
I	Root.zips+=allZips->sortedBy(value)->at(9).summary										✓

Action Rows:

A) If the value of `Root.zipCount` is 1, then set the string attribute `Root.zips` to be the `summary` field for the 1 zipcode

B) If the value of `Root.zipCount` is 2, then set the string attribute `Root.zips` to be the `summary` field for the first zipcode + the `summary` field of the second zipcode… **This pattern continues for up to 20 zipCodes.**

Rulesheet 7:



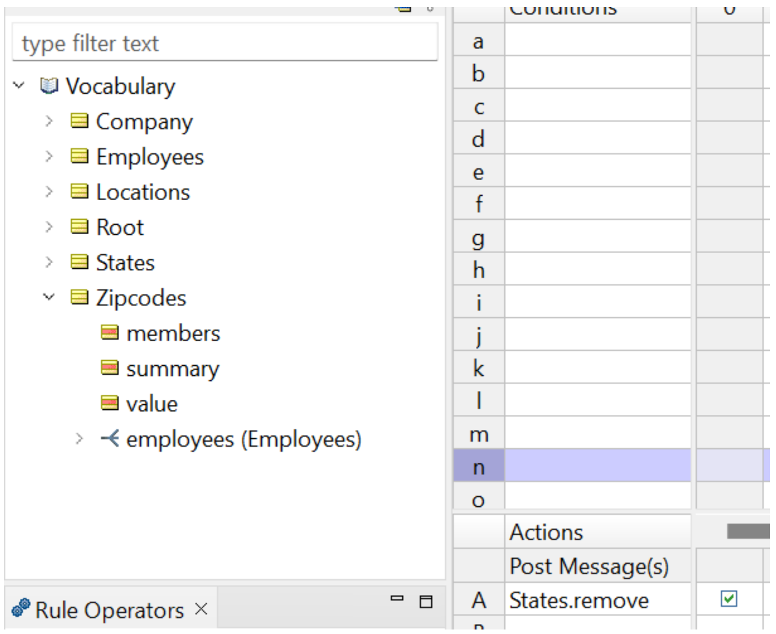
	Actions	
A	Zipcodes.employees-=Zipcodes.employees	✓
B	Zipcodes.remove	✓

Action Rows:

A) Disassociate the collection employees from being children entities to the Zipcodes entity

B) Eliminate the Zipcodes entity

Rulesheet 8:


type filter text

- Vocabulary
  - Company
  - Employees
  - Locations
  - Root
  - States
  - Zipcodes
    - members
    - summary
    - value
    - employees (Employees)

Rule Operators

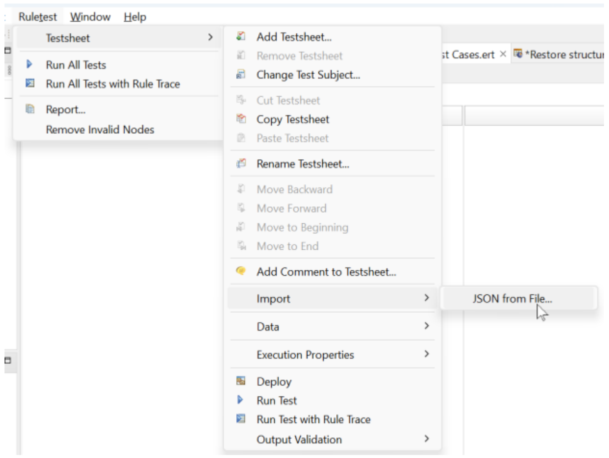
	Conditions	0
a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
	Actions	
	Post Message(s)	
A	States.remove	✓

Action Rows:

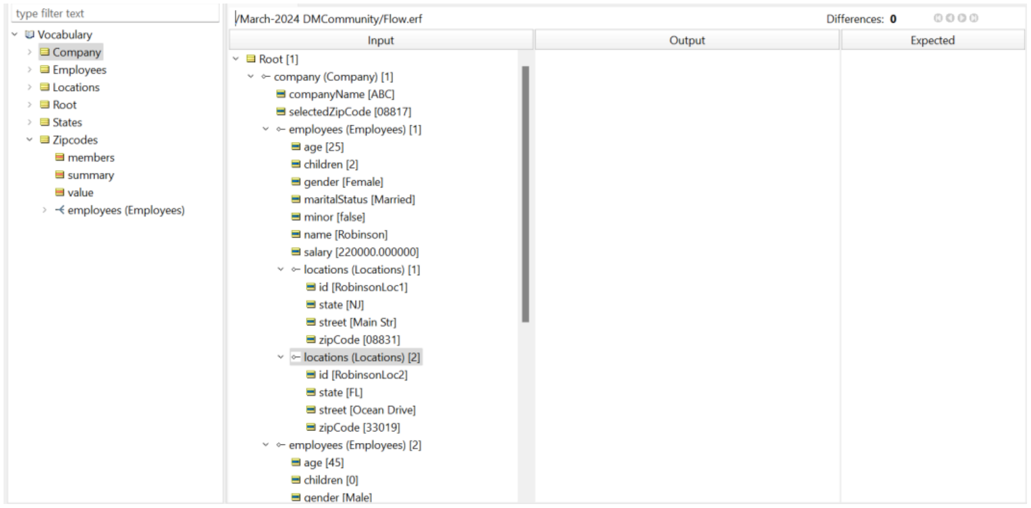
A) Eliminate the States entity

Verify outputs in Ruletest

1. We can import the same JSON document from the prompt into a Corticon Ruletest in order to verify the result and audit the sequence/nature of the changes all of the rules made.

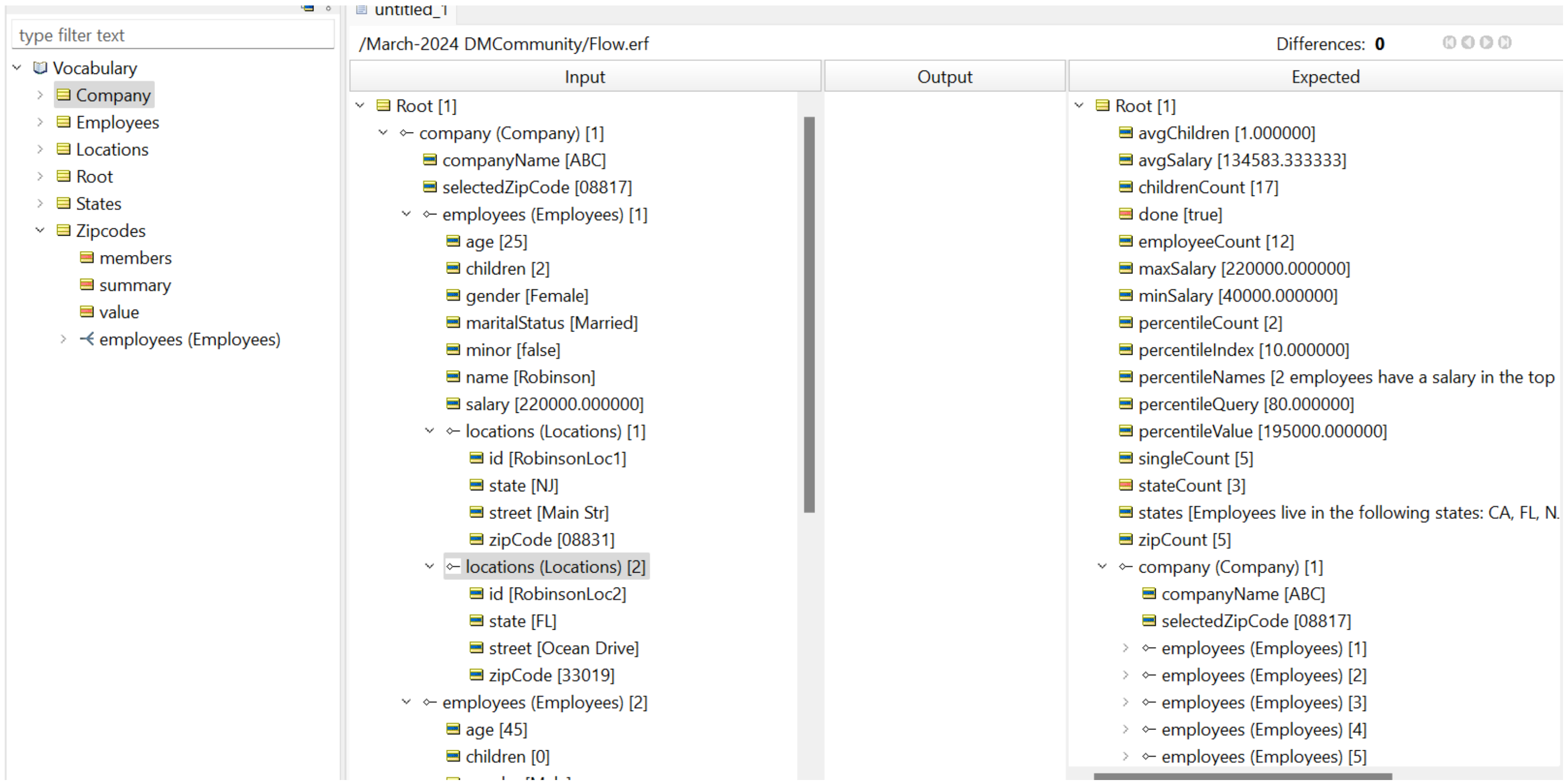


2. This data will be fed into the input payload, which will be run againstv the Ruleflow containing all 8 rulesheets and their rulesheets.



```
Root [1]
  company (Company) [1]
    companyName [ABC]
    selectedZipCode [08817]
    employees (Employees) [1]
      age [25]
      children [2]
      gender [Female]
      maritalStatus [Married]
      minor [false]
      name [Robinson]
      salary [220000.000000]
      locations (Locations) [1]
        id [RobinsonLoc1]
        state [NJ]
        street [Main Str]
        zipCode [08831]
      locations (Locations) [2]
        id [RobinsonLoc2]
        state [FL]
        street [Ocean Drive]
        zipCode [33019]
    employees (Employees) [2]
      age [45]
      children [0]
      gender [Male]
```

3. Optionally, we can define expected outputs. When we run the test, we can toggle through each difference between expected and actual outputs.



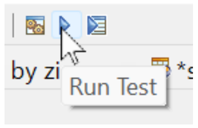
untitled_1

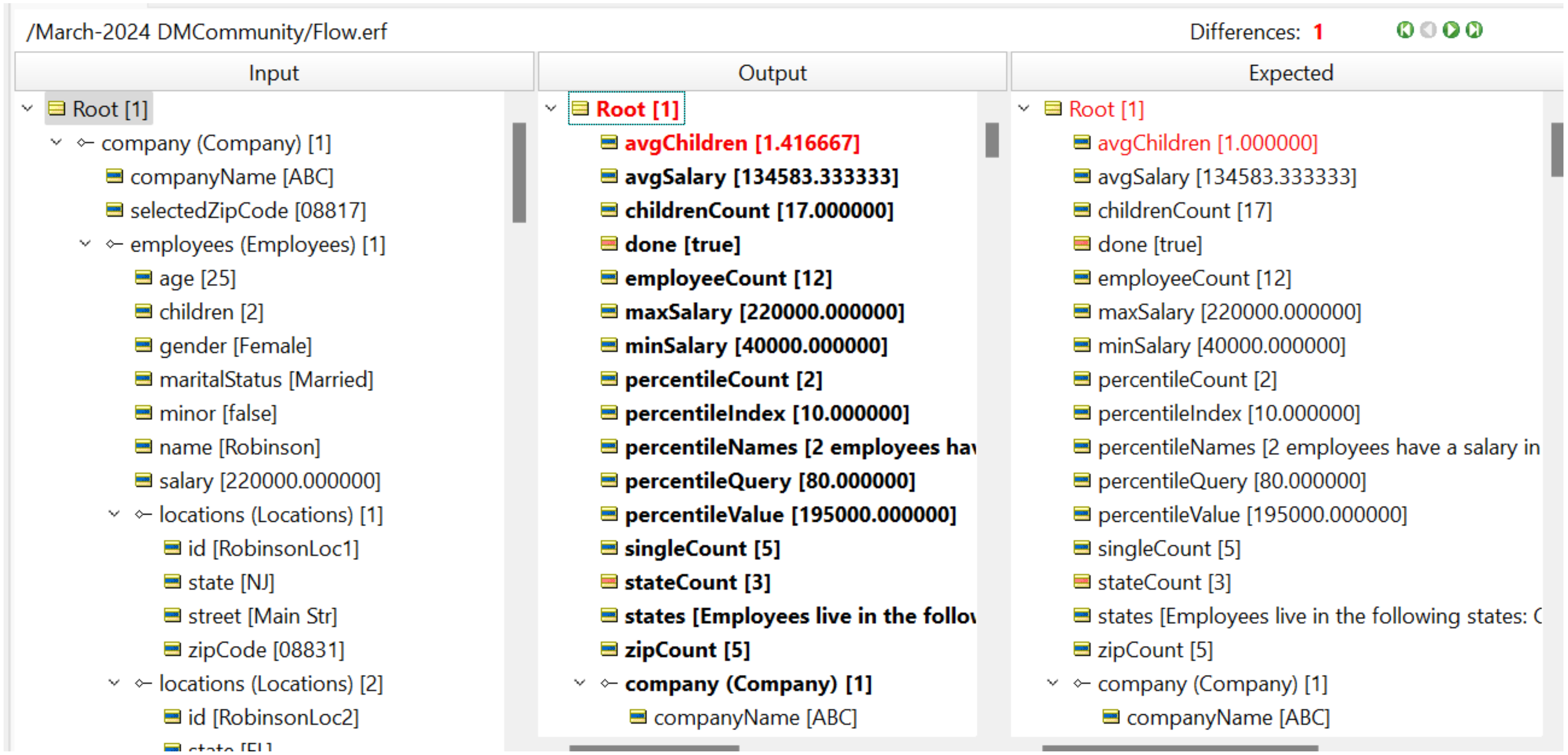
/March-2024 DMCommunity/Flow.erf Differences: **0**

Vocabulary
- Company
- Employees
- Locations
- Root
- States
- Zipcodes
  - members
  - summary
  - value
  - employees (Employees)

Input	Output	Expected
Root [1]		Root [1]
company (Company) [1]		avgChildren [1.000000]
companyName [ABC]		avgSalary [134583.333333]
selectedZipCode [08817]		childrenCount [17]
employees (Employees) [1]		done [true]
age [25]		employeeCount [12]
children [2]		maxSalary [220000.000000]
gender [Female]		minSalary [40000.000000]
maritalStatus [Married]		percentileCount [2]
minor [false]		percentileIndex [10.000000]
name [Robinson]		percentileNames [2 employees have a salary in the top
salary [220000.000000]		percentileQuery [80.000000]
locations (Locations) [1]		percentileValue [195000.000000]
id [RobinsonLoc1]		singleCount [5]
state [NJ]		stateCount [3]
street [Main Str]		states [Employees live in the following states: CA, FL, N
zipCode [08831]		zipCount [5]
locations (Locations) [2]		company (Company) [1]
id [RobinsonLoc2]		companyName [ABC]
state [FL]		selectedZipCode [08817]
street [Ocean Drive]		employees (Employees) [1]
zipCode [33019]		employees (Employees) [2]
employees (Employees) [2]		employees (Employees) [3]
age [45]		employees (Employees) [4]
children [0]		employees (Employees) [5]

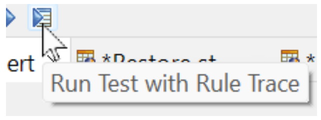
4. When we run the test, all of the rules in the ruleflow are generated into a JavaScript bundle and tested locally within the Corticon Studio ruletest. This JavaScript bundle generation step mirrors the actual deployment of a ruleflow to a decision service, so there is no distinction in behavior between results in test cases and live services.

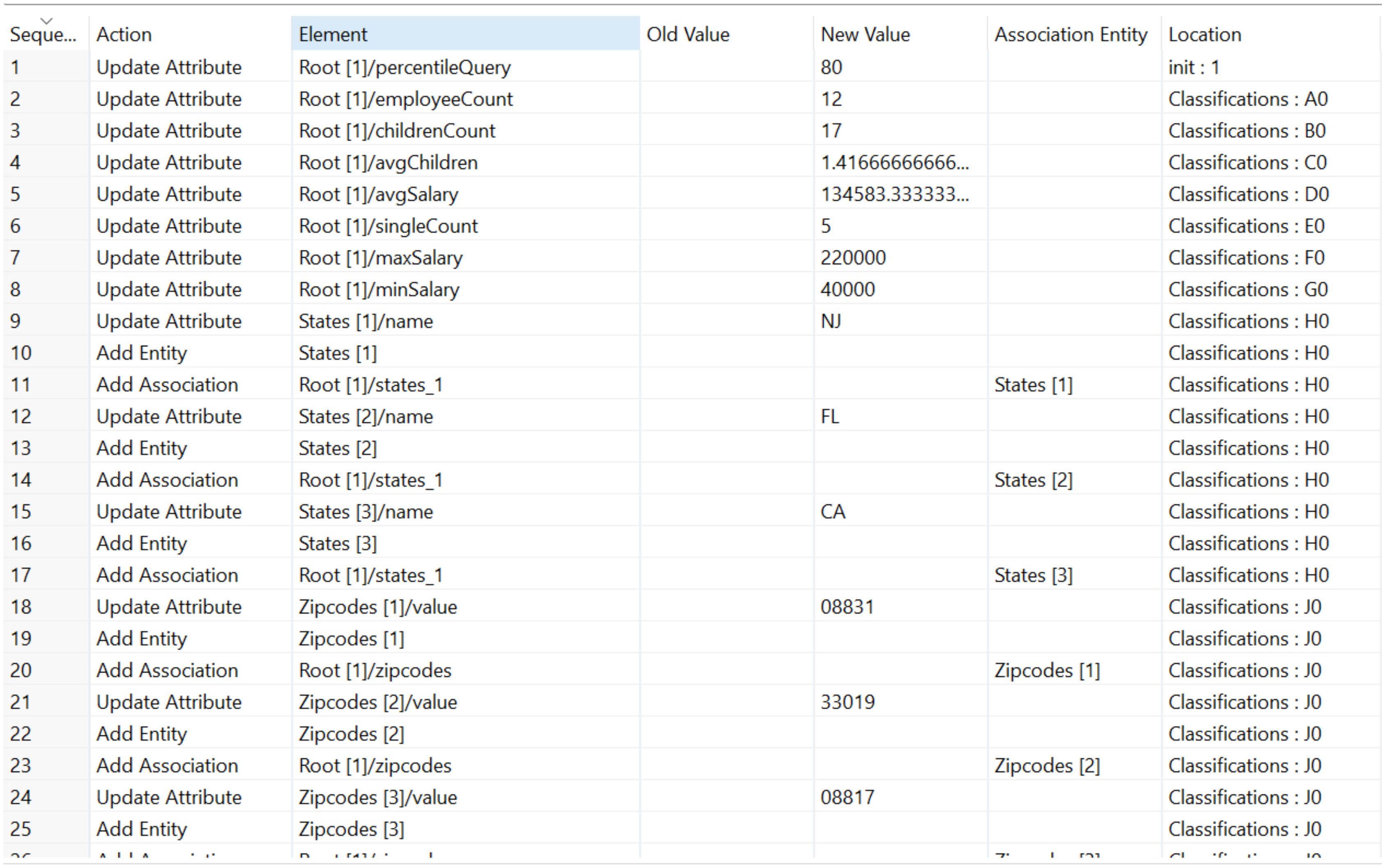



/March-2024 DMCommunity/Flow.erf   Differences: 1

Input	Output	Expected
Root [1]	Root [1]	Root [1]
company (Company) [1]	avgChildren [1.416667]	avgChildren [1.000000]
companyName [ABC]	avgSalary [134583.333333]	avgSalary [134583.333333]
selectedZipCode [08817]	childrenCount [17.000000]	childrenCount [17]
employees (Employees) [1]	done [true]	done [true]
age [25]	employeeCount [12]	employeeCount [12]
children [2]	maxSalary [220000.000000]	maxSalary [220000.000000]
gender [Female]	minSalary [40000.000000]	minSalary [40000.000000]
maritalStatus [Married]	percentileCount [2]	percentileCount [2]
minor [false]	percentileIndex [10.000000]	percentileIndex [10.000000]
name [Robinson]	percentileNames [2 employees hav	percentileNames [2 employees have a salary in
salary [220000.000000]	percentileQuery [80.000000]	percentileQuery [80.000000]
locations (Locations) [1]	percentileValue [195000.000000]	percentileValue [195000.000000]
id [RobinsonLoc1]	singleCount [5]	singleCount [5]
state [NJ]	stateCount [3]	stateCount [3]
street [Main Str]	states [Employees live in the follov	states [Employees live in the following states: (
zipCode [08831]	zipCount [5]	zipCount [5]
locations (Locations) [2]	company (Company) [1]	company (Company) [1]
id [RobinsonLoc2]	companyName [ABC]	companyName [ABC]

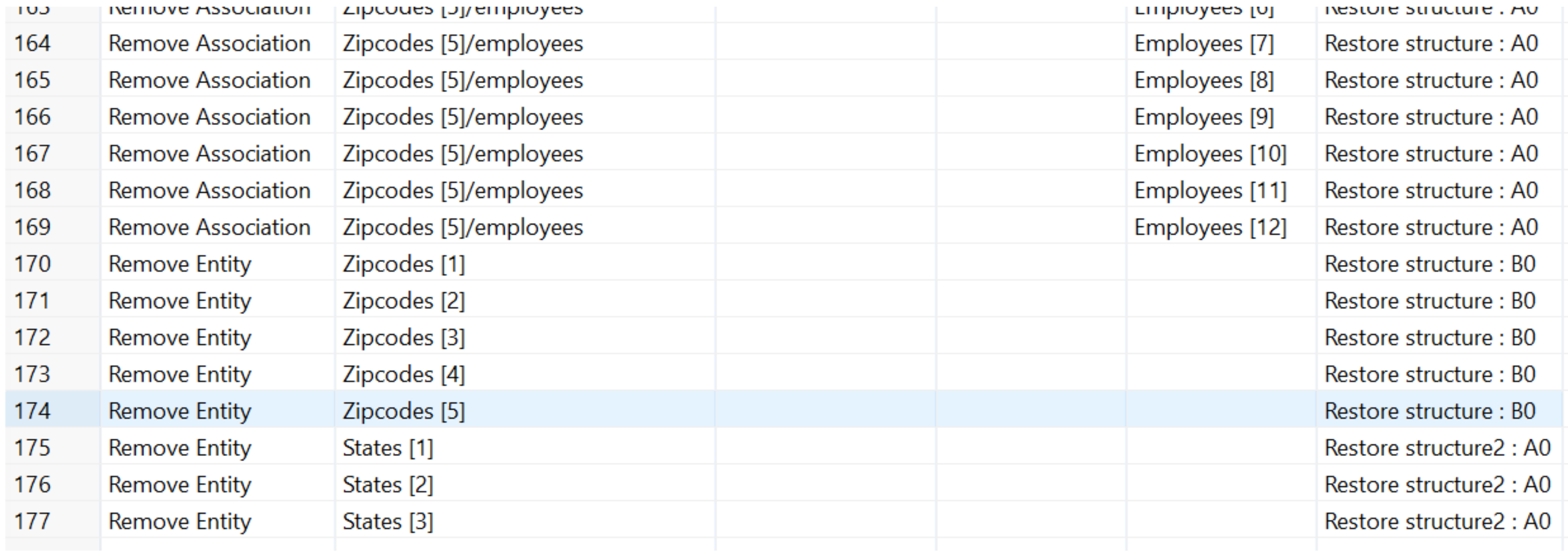
5. If we run the test once more, this time with 'Rule Trace', we'll see the entire sequence and nature of the changes to the input payload. The last column points us to the name of the rulesheet and rule number which produced each change.





Seque...	Action	Element	Old Value	New Value	Association Entity	Location
1	Update Attribute	Root [1]/percentileQuery		80		init : 1
2	Update Attribute	Root [1]/employeeCount		12		Classifications : A0
3	Update Attribute	Root [1]/childrenCount		17		Classifications : B0
4	Update Attribute	Root [1]/avgChildren		1.41666666666...		Classifications : C0
5	Update Attribute	Root [1]/avgSalary		134583.333333...		Classifications : D0
6	Update Attribute	Root [1]/singleCount		5		Classifications : E0
7	Update Attribute	Root [1]/maxSalary		220000		Classifications : F0
8	Update Attribute	Root [1]/minSalary		40000		Classifications : G0
9	Update Attribute	States [1]/name		NJ		Classifications : H0
10	Add Entity	States [1]				Classifications : H0
11	Add Association	Root [1]/states_1			States [1]	Classifications : H0
12	Update Attribute	States [2]/name		FL		Classifications : H0
13	Add Entity	States [2]				Classifications : H0
14	Add Association	Root [1]/states_1			States [2]	Classifications : H0
15	Update Attribute	States [3]/name		CA		Classifications : H0
16	Add Entity	States [3]				Classifications : H0
17	Add Association	Root [1]/states_1			States [3]	Classifications : H0
18	Update Attribute	Zipcodes [1]/value		08831		Classifications : J0
19	Add Entity	Zipcodes [1]				Classifications : J0
20	Add Association	Root [1]/zipcodes			Zipcodes [1]	Classifications : J0
21	Update Attribute	Zipcodes [2]/value		33019		Classifications : J0
22	Add Entity	Zipcodes [2]				Classifications : J0
23	Add Association	Root [1]/zipcodes			Zipcodes [2]	Classifications : J0
24	Update Attribute	Zipcodes [3]/value		08817		Classifications : J0
25	Add Entity	Zipcodes [3]				Classifications : J0

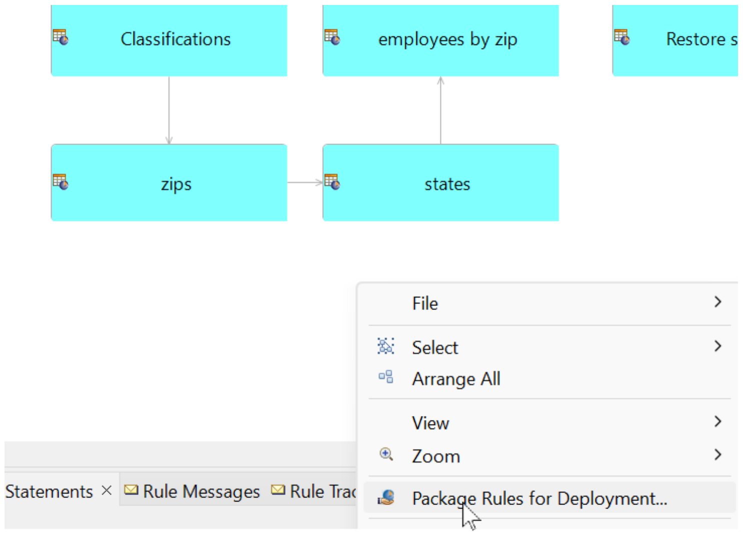
....

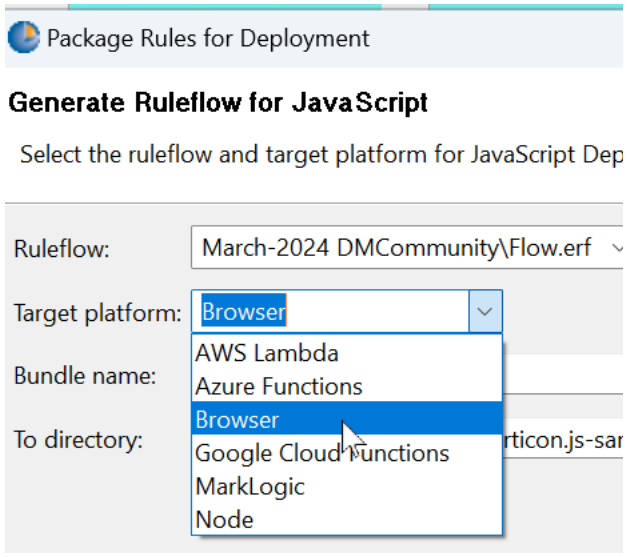


163	Remove Association	Zipcodes [5]/employees			Employees [6]	Restore structure : A0
164	Remove Association	Zipcodes [5]/employees			Employees [7]	Restore structure : A0
165	Remove Association	Zipcodes [5]/employees			Employees [8]	Restore structure : A0
166	Remove Association	Zipcodes [5]/employees			Employees [9]	Restore structure : A0
167	Remove Association	Zipcodes [5]/employees			Employees [10]	Restore structure : A0
168	Remove Association	Zipcodes [5]/employees			Employees [11]	Restore structure : A0
169	Remove Association	Zipcodes [5]/employees			Employees [12]	Restore structure : A0
170	Remove Entity	Zipcodes [1]				Restore structure : B0
171	Remove Entity	Zipcodes [2]				Restore structure : B0
172	Remove Entity	Zipcodes [3]				Restore structure : B0
173	Remove Entity	Zipcodes [4]				Restore structure : B0
174	Remove Entity	Zipcodes [5]				Restore structure : B0
175	Remove Entity	States [1]				Restore structure2 : A0
176	Remove Entity	States [2]				Restore structure2 : A0
177	Remove Entity	States [3]				Restore structure2 : A0

Package Rules for Deployment

All that remains is to deploy our ruleflow into a runnable Corticon.js JavaScript Decision Service Bundle.





Generate Ruleflow for JavaScript

Select the ruleflow and target platform for JavaScript Dep

Ruleflow: March-2024 DMCommunity\Flow.erf

Target platform: Browser

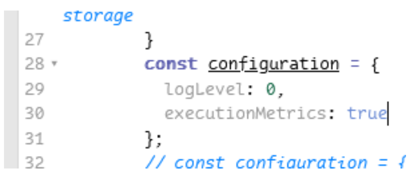
Bundle name:

To directory: rticon.js-sar

[Run this decision service in your browser]

Rule Trace Data at Runtime

A final note – the runnable sample can be easily tweaked to return the rule trace data (same rule trace data as we saw in the ruletest) in the response payload. In the HTML in the top left of the Codepen sandbox, simply add the `executionMetrics` configuration setting as shown, then re-run the decision service:



```
 storage
        }
        const configuration = {
          logLevel: 0,
          executionMetrics: true
        };
        // const configuration = {
```