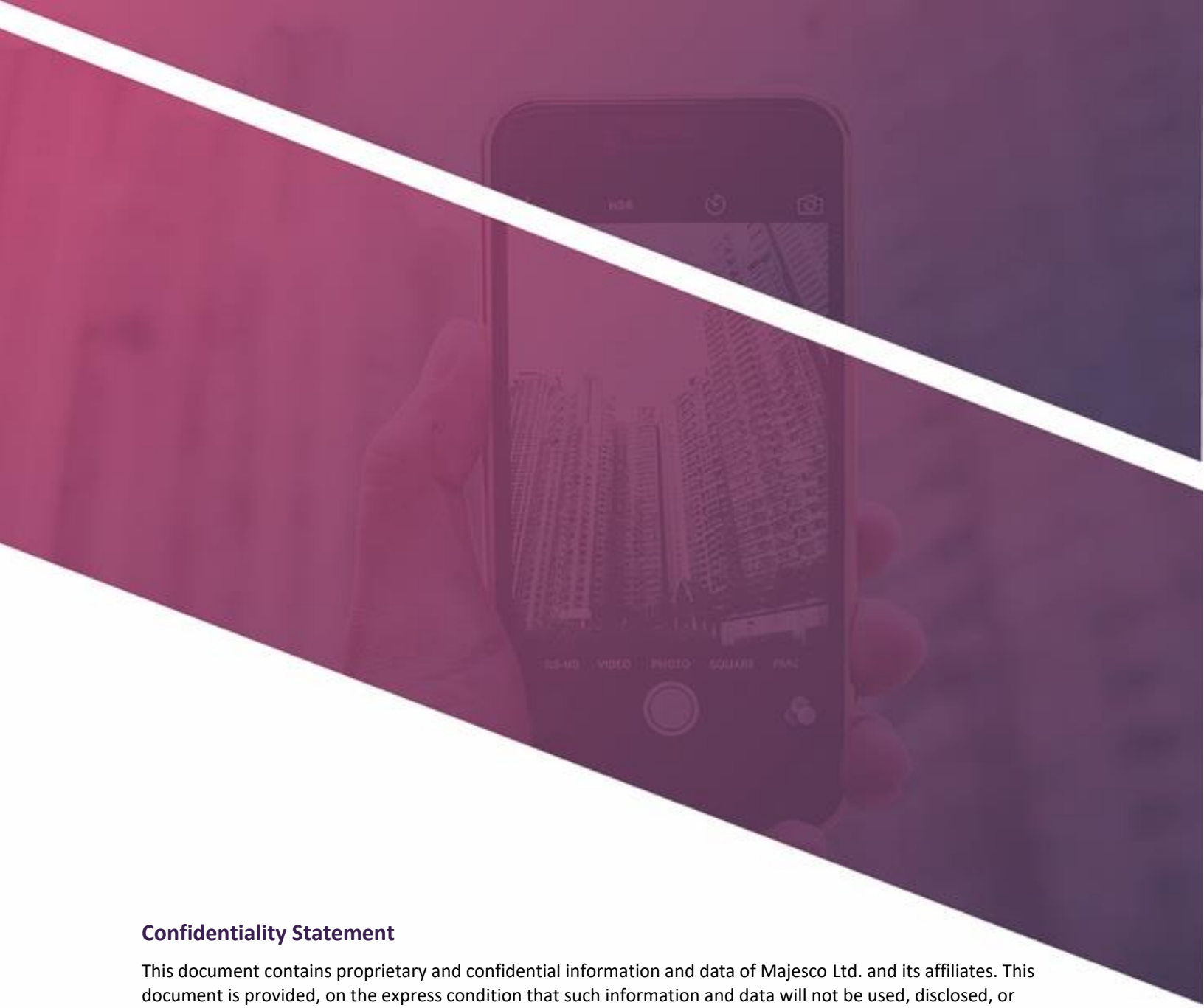
MAJESCO

The Future of Insurance Starts Here

Snowflake - Domain Specific Language



Confidentiality Statement

This document contains proprietary and confidential information and data of Majesco Ltd. and its affiliates. This document is provided, on the express condition that such information and data will not be used, disclosed, or reproduced in any form, in whole or in part, for any purpose (other than solely for evaluation purposes by authorized representatives with a need to know), without the express written approval of Majesco. The right to use, disclose, and reproduce such information and data shall be governed by the service agreement between the client and Majesco.

Warnings and Disclaimer

The information provided is on an as is basis. Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The authors and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book. Screenshots and data used within are for illustration purpose and may vary from the actual application.

Table of Contents

1 Introduction

Snowflake is an adaptive Domain Specific Language (DSL) and not a programming language. It is designed to write business processing logic for a particular business domain such as Insurance. Snowflake adapts very easily to the new or evolving domain object model. All programming language features are not be available as part of DSL language implementation.

1.1 Objectives

This language has been designed with the following objectives in mind:

- It should be simple enough for a business analyst or domain expert to use with minimal training
- It should be aware of the domain or Object Models
- It should be dynamic and not require phases such as compilation, build process etc.
- It should be fast and run as fast as compiled general-purpose language such as Java
- It should be extensible to allow future extension
- It should be modular and any future extensions should not make existing rules invalid

1.2 Focus Areas

Snowflake can be used to create rules or processing logic in following areas of the system:

- LOV (List of values)
- Validation rules
- Error messages
- Object Model rules such as validations, error messages
- UI rules such as conditional display of field/sections
- Navigation rules for conditional routing through screens in a flow
- Pre/Post processing logic between steps in a flow

1.3 Snowflake Editor

The Designer provides an intuitive editor wherever rules can be entered - for example, DSL can be written on:

- In an App - On a page > any widget > Settings icon > anywhere the '+' icon is available > Advanced logic in when configuration
- In an App - On a page > any widget > Settings icon > anywhere the '+' icon is available > is expression in then condition

- In an App - On a page > any widget > Settings icon > anywhere the '<>' icon is available > Expression editor
- In an App - On a page > In inner row widget > Click here to enter logic' link > Expression editor
- In an App - On a flow route > Settings icon > Event/Expression editor
- In an App - In a report > Expression editor
- In a Content - On an object > any object field > settings icon > Any properties having '+' > Advanced logic in when configuration
- In a Content - On an object > any object field > settings icon > Any properties having '+' > is expression in then condition
- In a Content - On an object model > any route between two objects or object models > settings icon > Any properties having '+' > Advanced logic in when configuration
- In a Content - On an object model > any route between two objects or object models > settings icon > Any properties having '+' > is expression in then condition
- In a Content - On an object model > any route between objects & object model and vice a versa> settings icon > Any properties having '+' > Advanced logic in when configuration
- In a Content - On an object model > any route between objects & object model and vice a versa > settings icon > Any properties having '+' > is expression in then condition

1.4 Features

Editor gives productivity features such as -

- Color coding (implemented)
- syntax highlighting (not yet implemented)
- auto completion (not yet implemented)
- error tips (not yet implemented)
- warning tips (not yet implemented)

For more insights on a Domain Specific Language please refer the following link
https://en.wikipedia.org/wiki/Domain-specific_language

2 Language Syntax

This reference guide will walk you through the language specification usages and examples.

- Snowflake is a case sensitive language. This applies to all the keywords, object models, event references as well.
- Snowflake is NOT space sensitive language. This applies to space between two separate words, lines between two words etc.

Correct examples:

```
i] def
   x = 5
ii] def x            =    5
```

Incorrect examples:

```
i] def x as new  MasterModel:      \*anything connected with
symbols is one entity*/
                    Root
ii] def x as n                                \* space between
characters of a single word is not allowed */
                    ew MasterModel:Root
```

Just like Java or any other language building blocks of Snowflake language is it's => keywords, operators, special characters and numbers.

3 Master Repository- All Snowflake Keywords, operators, special characters, numbers

Here is a super set list of all words (keywords, functions, operators), symbols (arithmetic operators, logical operators & other special characters) & numbers used in Snowflake. This repository mentions whether it is 'Implemented' in Platform application or not. It also mentions the combinations that word can have with other keywords & symbols.

Please take extra care while updating anything under this header, since this is the master repository for all.

3.1 Snowflake All Keywords

Sr No	keywords	Implemented in Platform ?	Type	All possible combinations with other keywords	Comment
1	from	Yes	MOQL	1] values from	
2	select	Yes	MOQL	1] from __ select	
3	where	Yes	MOQL	1] select __ where	
4	group by	Yes	MOQL	1] select __ group by	
5	order by	Yes	MOQL	1] select __order by	
6	ASC	Yes	MOQL	1] select ___order by ASC	
7	DESC	Yes	MOQL	1] select ___order by DESC	
8	lookup	Yes	MOQL	1] return lookup	
9	return	Yes	function	1] return concat	
10	security	Yes	function	Requires private URL at run time for execution	
11	context	Yes	function	Requires private URL at run time for execution	
12	def	Yes	statements	1] def x as 5 2] def x as new MasterModel:Root 3] def x = 5 others?	
13	if	Yes	statements	1] if__then__end 2] if__then__else__then__end	

Sr No	keywords	Implemented in Platform ?	Type	All possible combinations with other keywords	Comment
14	then	Yes	statements	1] if__then__end 2] if__then__else__then__end	
15	end	Yes	statements	1] if__then__end 2] if__then__else__then__end 3] for__do__end 4] foreach__do__end	
16	else	Yes	statements	1] if__then__else__then__end	
17	foreach	Yes	statements	1] foreach__do__end	
18	do	Yes	statements	1] foreach__do__end	
19	for	Yes	statements	1] for__do__end	
20	increment	Yes	function		
21	execute	Yes	statements		
22	rule	Yes	statements		
23	skip	Yes	statements		
24	execution	Yes	statements		
25	resume	Yes	statements		
26	break	Yes	statements		
27	continue	Yes	statements		
28	save	No	function		descoped
29	raise	No	function		descoped
30	mod	Yes	function		
31	or	Yes	Operators		

Sr No	keywords	Implemented in Platform ?	Type	All possible combinations with other keywords	Comment
32	like	Yes	Operators		
33	not like	Yes	Operators		
34	less than	Yes	Operators		
35	greater than	Yes	Operators		
36	less than equals to	Yes	Operators		
37	greater than equals to	Yes	Operators		
38	not equals to	Yes	Operators		
39	equals to	Yes	Operators		
40	TRUE	Yes	Operators	keywords in Snowflake are - 'true' & 'false' not TRUE & FALSE	
41	FALSE	Yes	Operators	keywords in Snowflake are - 'true' & 'false' not TRUE & FALSE	
42	new	Yes	statements	1] def x as new MasterModel:Root 2] new MasterModel:Insured	
43	del	Yes	statements	1] del association	
44	association	Yes	statements	1] del association	
45	Parent	Yes	keyword	1] This.Parent	
46	This	Yes	keyword	1] This.Parent	
47	Global	Yes	keyword		
48	Event	No	keyword		descoped
49	Root	Yes	keyword	1] new	

Sr No	keywords	Implemented in Platform ?	Type	All possible combinations with other keywords	Comment
50	values	Yes	keyword	1] number of values others?	
51	without descendants	Yes	keyword		
52	descendants	Yes	keyword	1] without descendants	
53	using	Yes	keyword		
54	null	Yes	keyword	1] as null 2] equals to null	
55	of	Yes	keyword	1] min of 2] max of 3] avg of 4] count of 5] sum of 6] length of? Other combinations?	
56	with	Yes	keyword	1] with optional match 2] with exact match	
57	starting at	Yes	keyword		
58	to	Yes	keyword	Other combinations?	
59	size	No	keyword		descoped
60	matching	Yes	function		
61	separated	Yes	function	1] separated by others?	
62	by	Yes	keyword	1] separated by others?	

Sr No	keywords	Implemented in Platform ?	Type	All possible combinations with other keywords	Comment
63	not	Yes	keyword	1] not like 2] not contains 3] not equals to 4] not greater than 5] not less than 6] not less than equals to 7] not greater than equals to 8] not null others?	
64	contains	Yes	function	1] not contains other?	
65	starts with	Yes	function	1] not starts with	
66	ends with	Yes	function	2] not ends with	
67	ignore case	Yes	function		
68	matches	Yes	function	1] with optional matches 2] matches with() 3] matches case? Others?	
69	start transaction	Yes	function		
70	complete transaction	Yes	function		
71	concat	Yes	function	1] return concat (" "," ")	
72	length	Yes	function	1] length of?	
73	left	Yes	function	1] left of	left of, left()
74	mid	Yes	function	2] mid of?	
75	right	Yes	function	1] right of	
76	compare	Yes	function	1] compare with others?	

Sr No	keywords	Implemented in Platform ?	Type	All possible combinations with other keywords	Comment
77	trim	Yes	function	1] trim left 2] trim right	
78	trim left	Yes	function		
79	trim right	Yes	function		
80	upper	Yes	function	1] return upper	
81	lower	Yes	function	1] return lower	
82	title	Yes	function	1] return title	
83	sentence	TBD	keyword		
84	left pad	Yes	function	1] return left pad	
85	right pad	Yes	function	1] return right pad	
86	replace	Yes	function	1] return replace	
87	replace all	Yes	function	1] return replace all	
88	replace first	Yes	function	1] return replace first	
89	replace last	Yes	function	1] return replace last	
90	split	Yes	function	1] split () 2] split __,__ others?	
91	value from	Yes	keyword	1] value from x at y	
92	columns	Yes	keyword	1] return columns(x) 2] for y as 1 to columns(x) do	
93	number of values	Yes	function	1] return number of values	
94	rows	Yes	keyword	1] return rows(x) 2] for y as 1 to columns(x) do	

Sr No	keywords	Implemented in Platform ?	Type	All possible combinations with other keywords	Comment
95	number of records	TBC	function	1] return number of records	Need syntax & information
96	at	Yes	keyword	1] value from x at y 2] starting at others?	
97	reference for	Yes	keyword	1] return reference for	
98	date	Yes	keyword	1] return date?	
99	current date	Yes	function	1] return current date	
100	as	Yes	keyword	1] def x as 5 2] for x as 5 3] x as 5 4] if x as 5 5] for x as 5 6] foreach x as 1 others?	
101	with format	Yes	keyword	1] return with format	
102	difference	Yes	function	1] date difference others?	
103	change	Yes	function	1] change date years 2] change date months 3] change date days 4] as change date 5] as current date Others?	
104	between	Yes	function	1] return between	
105	format date	Yes	function	1] return format date	
106	ceiling	Yes	function	1] return ceiling	
107	floor	Yes	function	1] return floor	
108	round	Yes	function	1] return round	

Sr No	keywords	Implemented in Platform ?	Type	All possible combinations with other keywords	Comment
109	up to	Yes	function	1] return up to	
110	digit	Yes	keyword	1] return digit	
111	absolute	Yes	keyword	1] return absolute	
112	convert to text	Yes	function	1] return convert to text	
113	format number	Yes	function	1] return format number	
114	in	Yes	keyword	1] min in 2] max in	
115	min	Yes	function	1] min in	
116	max	Yes	function	1] max in	
117	avg	Yes	function	1] return avg	
118	count	Yes	function	1] return count	
119	sum	Yes	function	1] return sum	
120	convert to number	Yes	function	1] return convert to number	
121	with optional match	Yes	statements	1] return with optional match	JIRA-4097. Also add keywords and cases for other joins
122	Base	Yes	keyword	1] return Base	
123	Custom	Yes	keyword	1] return Custom	
124	in memory	Yes	statements	1] return in memory	
125	as change date	Yes	function	1] return as change date	
126	as current date	Yes	function	1] return as current date	

Sr No	keywords	Implemented in Platform ?	Type	All possible combinations with other keywords	Comment
127	HAS_ROLE	Yes	keyword	1] security.HAS_ROLE	
128	USER_LAST_LOGIN	Yes	keyword	1] security.USER_LAST_LOGIN	
129	USER_NAME	Yes	keyword	1] security.USER_NAME	
130	CURRENT_ROLE	Yes	keyword	1] security.CURRENT_ROLE	
131	CLIENT_DEVICE_TYPE	Yes	keyword	1] security.CLIENT_DEVICE_TYPE	
132	CLIENT_BROWSER	Yes	keyword	1] security.CLIENT_BROWSER	
133	CLIENT_ORIENTATION	Yes	keyword	1] security.CLIENT_ORIENTATION	
134	CLIENT_OS	Yes	keyword	1] security.CLIENT_OS	
135	CLIENT_SCREEN_WIDTH	Yes	keyword	1] security.CLIENT_SCREEN_WIDTH	
136	format time?	TBD	function	1] return format time?	not a keyword yet. BLACKSWAN-5135,5138
137	decrement	TBD	keyword		Not a keyword yet. If functionality is already added, this should be added in the list.
138	exact	TBD	keyword	1] with exact match	Not a keyword yet. If functionality is already added, this should be added in the list.

3.2 Symbols

Sr No	Symbol	Implemented in Platform?	Type	Comments/Combinations
1	.	Yes	Object Relation	ObjectName.FieldName
2	:	Yes	Root	ModelName:Root
3	" "	Yes	String	"String"
4	,	Yes	Separator	def x = 0, y = 1
5	<	Yes	Relational operator	less than
6	<=	Yes	Relational operator	less than equals to
7	>	Yes	Relational operator	greater than
8	>=	Yes	Relational operator	greater than equals to
9	=	Yes	Relational operator	equals to
10	<>	Yes	Relational operator	not equals to
11	+	Yes	Arithmetic operator	add
12	-	Yes	Arithmetic operator	subtract
13	*	Yes	Arithmetic operator	multiply
14	/	Yes	Arithmetic operator	divide
15	* */	Yes	Comment	multi line comment
16	#	Yes	Comment	single line comment
17	()	Yes	Function	concat()
18	[]	Yes	Conditional parameters	[a-z]

3.3 Numbers

Sr No	Numbers
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10

Sr No	Numbers
11	All positive and negative rational numbers

3.4 Snowflake Functions

Sr No	FunctionType	Functions
1	Date Functions	
2		
3		
4		
5	String Functions	
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16	Logical Functions	
17		
18		
19		
20		
21		
22		
23		
24	Arithmetic Functions	
25		

Sr No	FunctionType	Functions
26		
27		
28		
29		
30		
31	Other functions	
32		
33		
34		
35		
36		
37		
38		
39		
40		

4 Operators

There are Arithmetic, Logical and relational type of operators available in Snowflake.

4.1 Arithmetic Operators

Below mentioned arithmetic operators can be used in any DSL code written in expression editor available at page, flow, object, object model level.

Operation	Symbol
Add	+
Subtract	-
Multiply	*
Divide	/
Percentage	%
Mod	mod

4.1.1 Examples for Arithmetic addition operator

Example1

```
def x as 10
return x+10
#Output above snowflake codes is 20.
or
def x as 10
x=x+10
return x
#Output above snowflake codes is 20.
```

Example2

```
def x as 10
x= x+x
return x
#Output of above snowflake code is 20.
```

Example3

```
def x, z as 30
def y as 20
x = y+z
return x
#Output of above snowflake code is 50.
```

Example4 [incorrect syntax error]
Unlike Java operations during initialization are not supported.

```
def y as 30
def x as y + 10
return x
#At designer level error message is displayed "Syntax error at or near [+], line: position: [2: 11]"
```

4.1.2 Example for Arithmetic multiplication operator

Example1

```
def x as 10
return x * 5
#Output above snowflake codes is 50.
or
def x as 10
x= x * 5
return x
#Output above snowflake codes is 50.
```

Example2

```
def x as 10
x= x * x
return x
#Output of above snowflake code is 100.
```

Example3

```
def x, z as 30
def y as 20
x = z * y
return x
#Output of above snowflake code is 600.
```

Example4
Unlike Java operations during initializations are not supported

```
def y as 30
def x as y * 10
return x
#At designer level error message is displayed "Syntax error at or near [ + ] , line : position : [ 2 : 11 ]"
```

4.1.3 Example for Arithmetic subtraction operator

Example1
def x as 10
return x - 5
#Output above snowflake codes is 5.
or
def x as 10
x= x - 5
return x
#Output above snowflake codes is 5.

Example2
def x as 10
x= x - x
return x
#Output of above snowflake code is 0.

Example3
def x, z as 30
def y as 20
x = z - y
return x
#Output of above snowflake code is 10.

Example4
Unlike Java operations during initialization are not supported.
def y as 30
def x as y - 10
return x
#At designer level error message is displayed "Syntax error at or near [+], line: position: [2:11]"

4.1.4 Example for Arithmetic Division operator

Example1
def x as 10
return x / 5
#Output above snowflake codes is 2.
or
def x as 10
x= x / 5
return x
#Output above snowflake codes is 2.

Example2
def x as 10
x= x / x
return x
#Output of above snowflake code is 1.

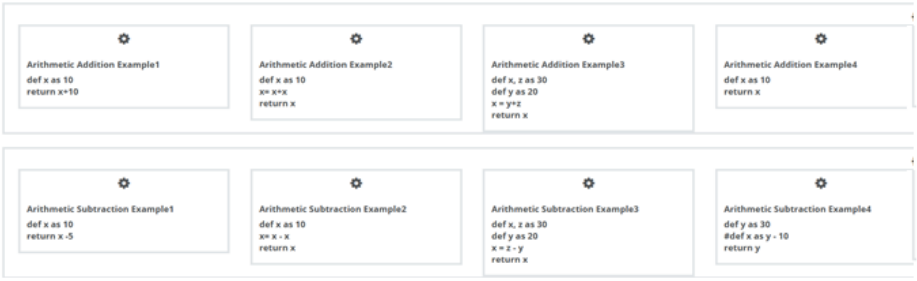
Example3
def x, z as 30
def y as 10
x = z / y
return x
#Output of above snowflake code is 3.

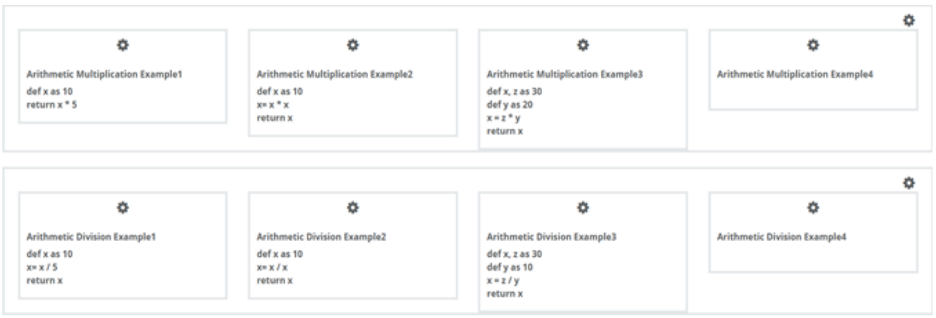
Example4
Unlike Java operations during initialization are not supported
def y as 30
def x as y / 10
return x
#At designer level error message is displayed "Syntax error at or near [+] , line : position : [2 : 11]"

4.1.5 Example for Arithmetic Percentage operator

def x as 15, y as 30
def z as 0
z = x % y
return z
#Output of above snowflake code is 50%.

4.1.6 Designer level illustration for few of above codes:





4.1.7 Verify at runtime

4.2 Relational Operators

Relational operators allow the user to compare values for equality. For relational operators, the words and/or symbols can be used as mentioned below in any DSL code written in expression editor available at page, flow, object, object model level.

Words	Symbol
equals to	=
not equals to	<>
greater than	>
greater than equals to	>=
less than	<
less than equals to	<=

4.2.1 Example#1 Use of =

```
def x as 10
def y
if x = 10
then y = 20
end
return y
```

Output of above DSL code should be: 20

4.2.2 Example#3 Use of >

```
def x as 10
def y
if x > 9
then y =11
end
return y
```

Output of above DSL code should be: 11

4.2.3 Example#5 Use of >=

```
def x as 10
def y
if x >= 10
then y = 11
end
return y
```

Output of above DSL code should be: 11

4.2.4 Example#2 Use of <>

```
def x as 10
def y
if x <> 11
then y =11
end
return y
```

Output of above DSL code should be: 11

4.2.5 Example#4 Use of <

```
def x as 10
def y
if x < 11
then y =11
end
return y
```

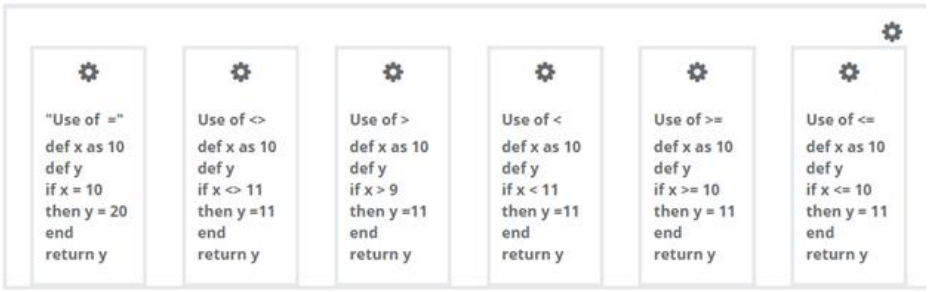
Output of above DSL code should be: 11

4.2.6 Example#6 Use of <=

```
def x as 10
def y
if x <= 10
then y = 11
end
return y
```

Output of above DSL code should be: 11

4.2.7 Designer level illustration of above snowflake codes



4.2.8 Runtime outputs

4.3 Logical Operators

For logical operators must be expressed as word and not in symbol. The words can be used as mentioned below in any DSL code written in expression editor available at page, flow, object, object model level.

Logical Operatorsr
or
and

4.3.1 Example#1 Use of and operator in if else if block

```
def x, y as 10
if x = null and y < 10 then
x = 1
else
```

```
if y >= 10 and y < 15 then
x = 2
end
end
return x
# Result of above execution block will be 2
```

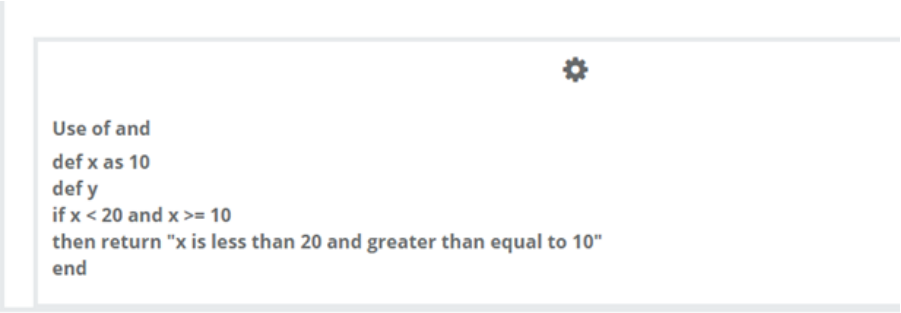
4.3.2 Example#2 Use of and

```
def x as 10 def y if x < 20 and x >= 10
then return "x is less than 20 and greater than equal to 10"
end
```

4.3.3 Example#3 Use of or operator

```
def x as 10 def y if x < 10 or x >= 10
then return "x is less than 10 or greater than equal to 10"
end
```

4.3.4 Designer level illustration of above snowflake codes



4.3.5 Runtime outputs

Use of and

x is less than 20 and greater than equal to 10

Use of or

x is less than 10 or greater than equal to 10

5 Variables

Variables - Variables are containers for storing data values.

5.1 Local Variables

Declaring and initializing local variables

5.1.1 #declaring and initializing x as a whole number

```
def x as 10
return x
#Output for above snowflake code is '10'.
```

5.1.2 #declaring and initializing x as a decimal

```
def x as 100.99
return x * 2
#Output for above snowflake code is '201.98'.
```

5.1.3 #declaring and initializing x as a single Character

```
def x as "D"
return x
#Output for above snowflake code is 'D'.
Like Java we cannot use characters by defining them in single quotes.
But with above code we can achieve same behavior.

#Here variable x is a local variable and its data type is not fixed as per the right hand side
value of assignment changes the data type of var x changes
def x as 5
def x as 10.5
def x as "Hello"
def x as true
return x
#Output for above snowflake code is 'true'.
```

5.2 Designer level illustration

5.3 Run time illustration

5.4 Variable Assignment

Business logic in rules can assign value to local variable by referencing object data, or update the object field values by referencing field name as assignee.

In below example, at runtime through Page widget textbox, one can provide desired color as an input and with the help of 'if - else' block based on the input value, at runtime user can assign value to an another object's field.

Assume 'Vehicle' object already exists with two fields, named as 'Color' and 'RiskToInsure' and both are having 'Short Description' as a data type.

Here x is a local variable. Further with statement 'This.Color' we are referring to current value in field 'Color' which is being assigned to variable x.

With if block we are comparing users input with 'desired static value', which is "Red" in this example, further based on if blocks decision, we are assigning "High" as a value to an another object field which is "RiskToInsure" in this case, this will result in updating the RiskToInsure field in database.

```
def x as This.Color
if x = "Red" then
This.RiskToInsure = "High"
end
```

```
return This.RiskToInsure
```

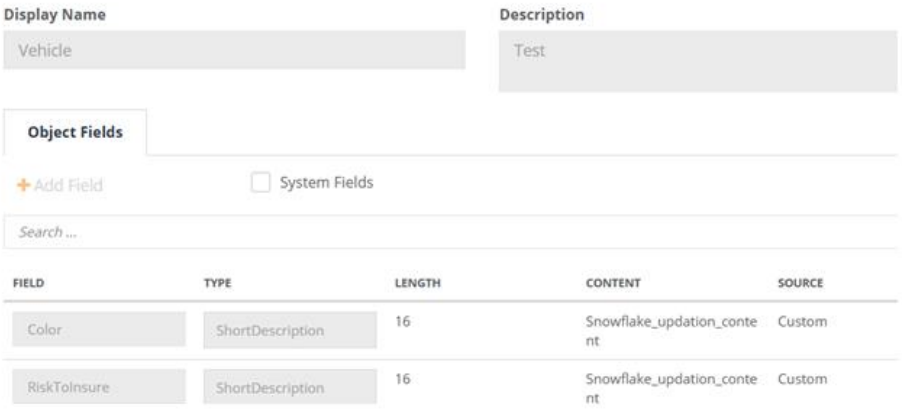
Above mentioned same requirement can be achieved with below mentioned snowflake code

```
if This.Color = "Red" then
This.RiskToInsure = "High"
end
return This.RiskToInsure
```

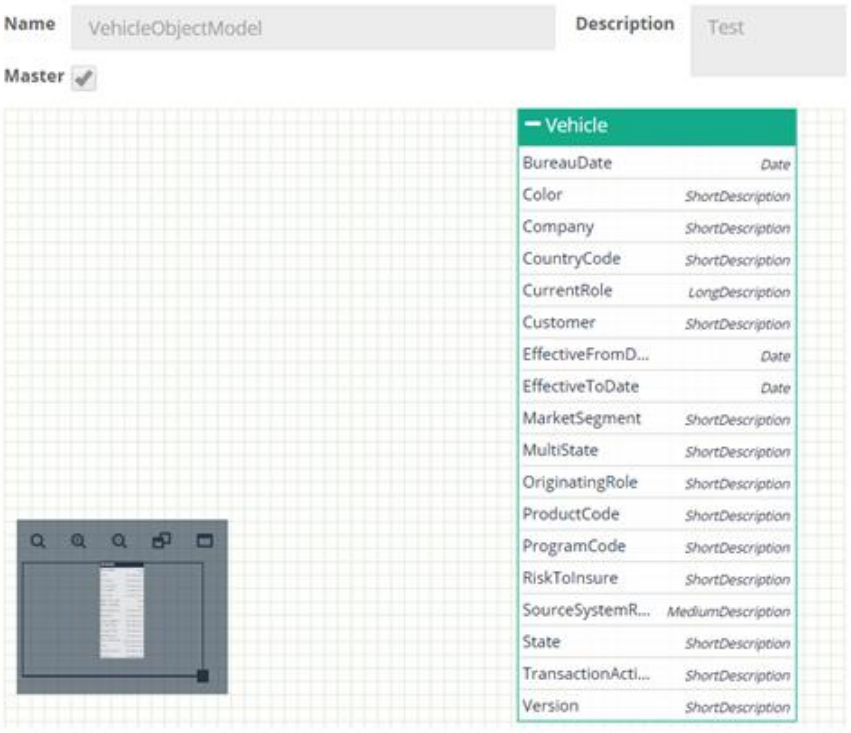
To illustrate above code follow below mentioned steps.

Create objects as below

Vehicle object



Vehicle object model

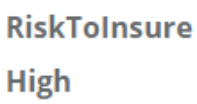


Designer level Page

Verify at runtime

Enter color

Verify value assigned to RiskToInsure field



6 Use of Global Variables

When we want to access values of system variables which are set in previous pages of one particular flow at that time we use Global keyword along with '.' operator.

Syntax: return Global.STATE_CODE

To use it in an application, Consider a hierarchy of an object model, where we have Policy as a root object, Applicant and OccupancyDetails are child siblings objects, having fields as mentioned below.

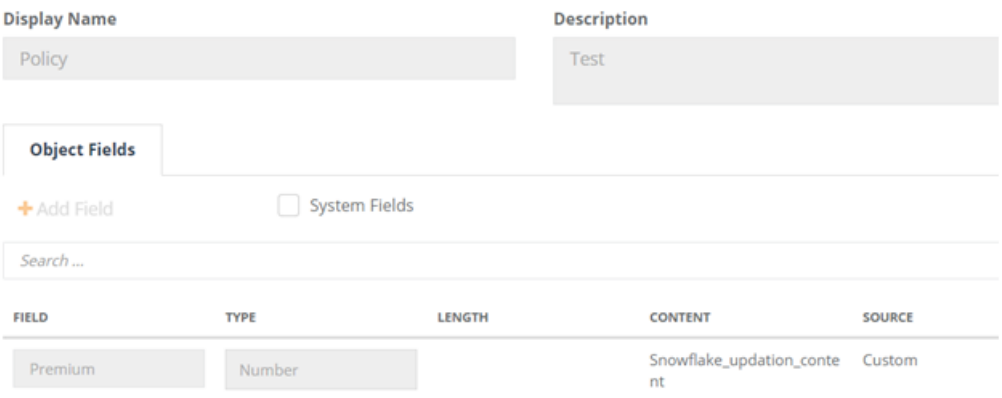
We are entering value in 'State' field and based on that we are executing rule on 'StateDetails' field.

Policy Object - State[System Variable]
- Premium
- Applicant [Child object of Root]
- FirstName
- LastName
- OccupancyDetails [Child object of Root]
-Line1
-Line2
-Zip
StateDetails [We will set 'placeholder' value of this field on OccupancyDetailsPage, based on Value captured in 'State' field on PolicyPage]

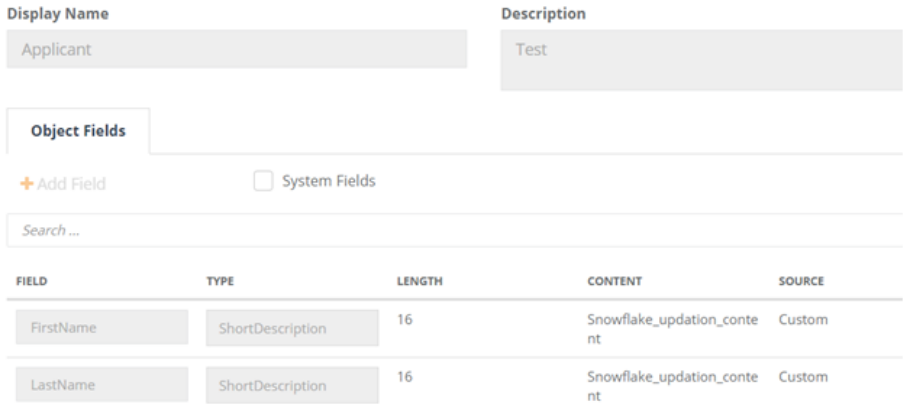
Follow below steps to evaluate above mentioned scenario.

Create object as shown below

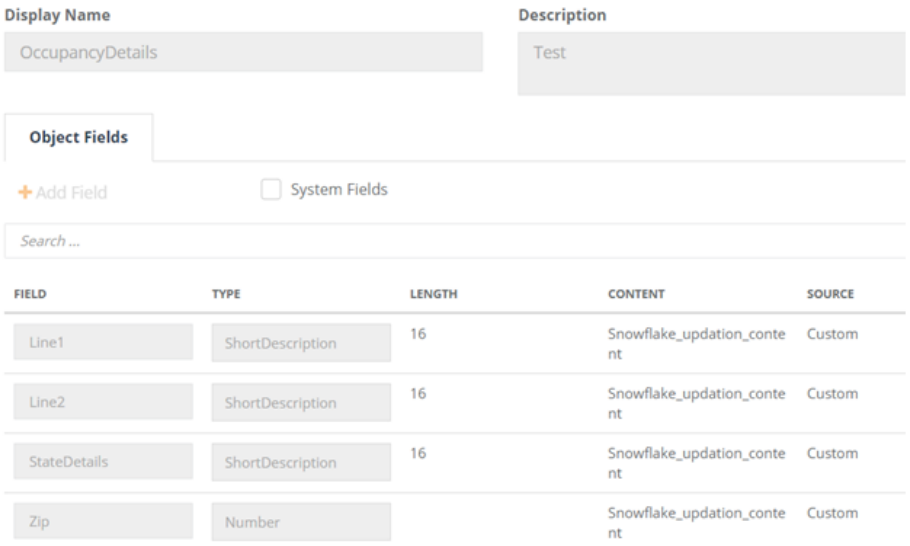
Policy object



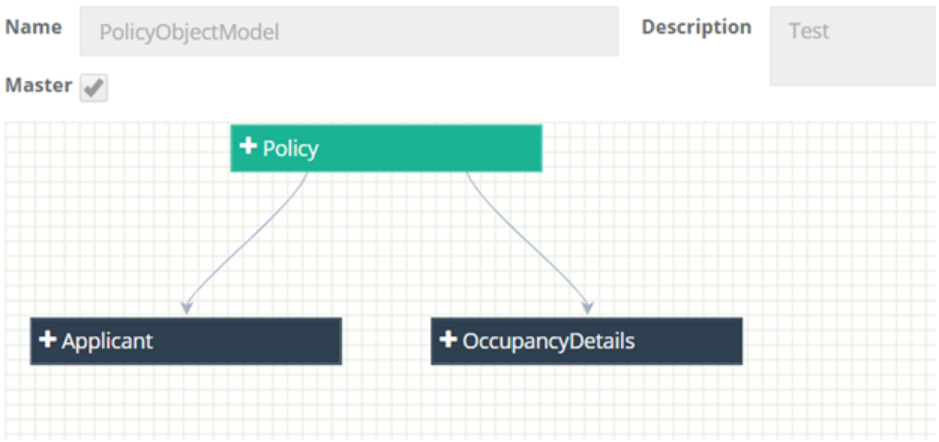
Applicant object



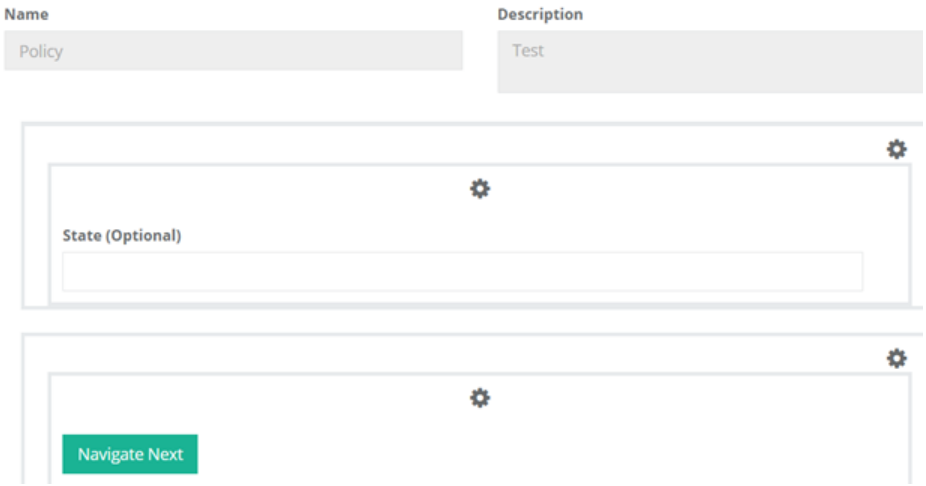
OccupancyDetails object



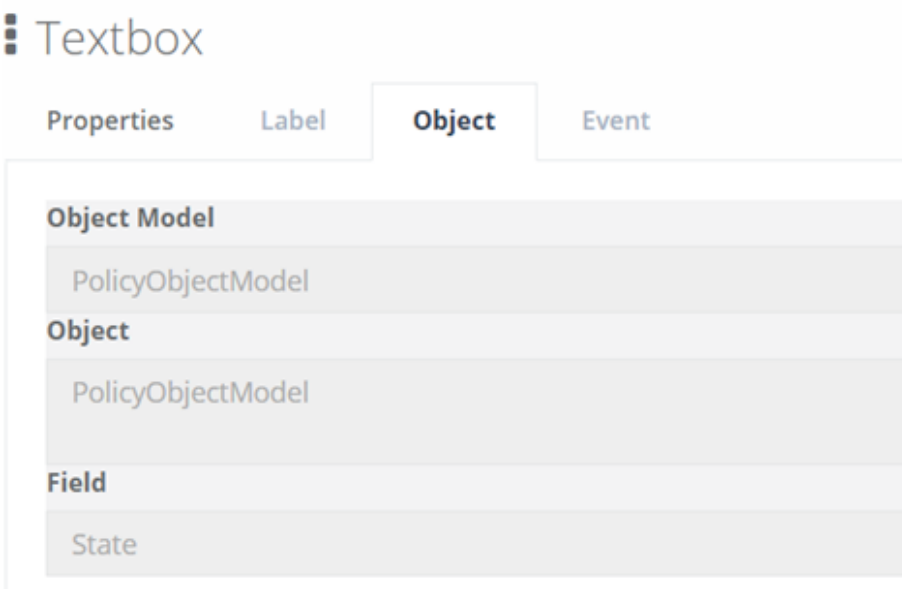
PolicyObjectModel



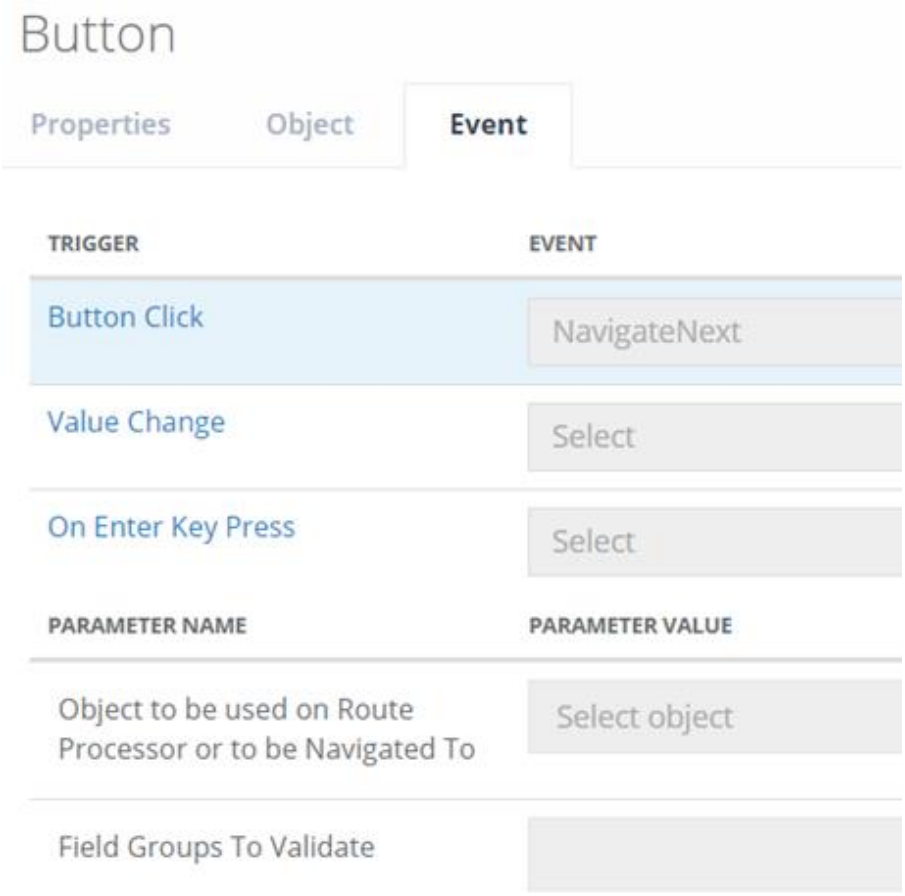
Policy page



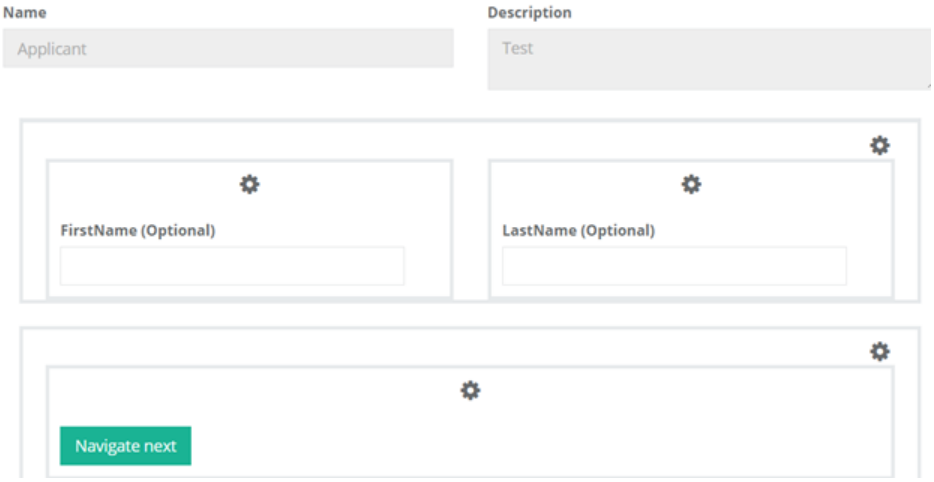
State field



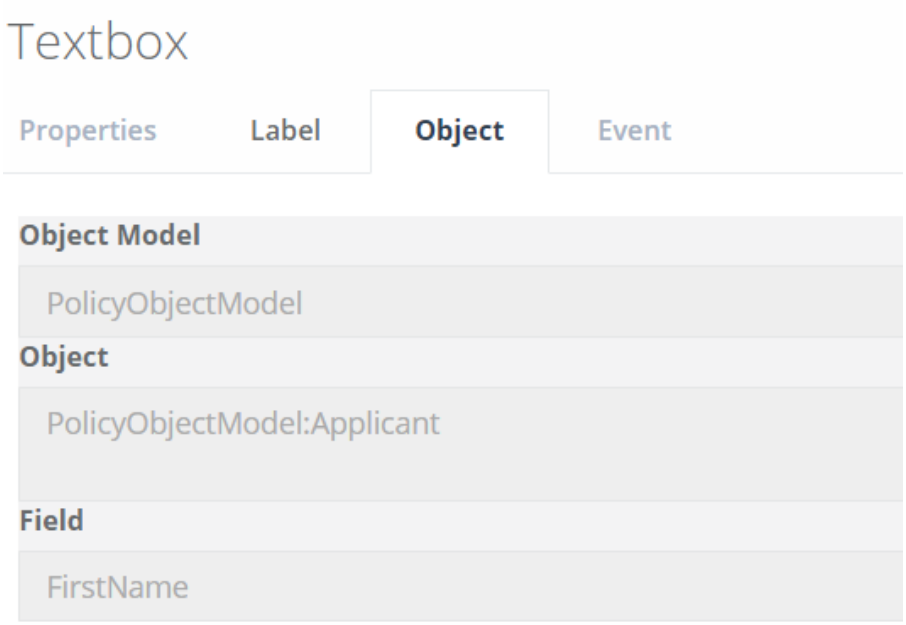
Navigate next button



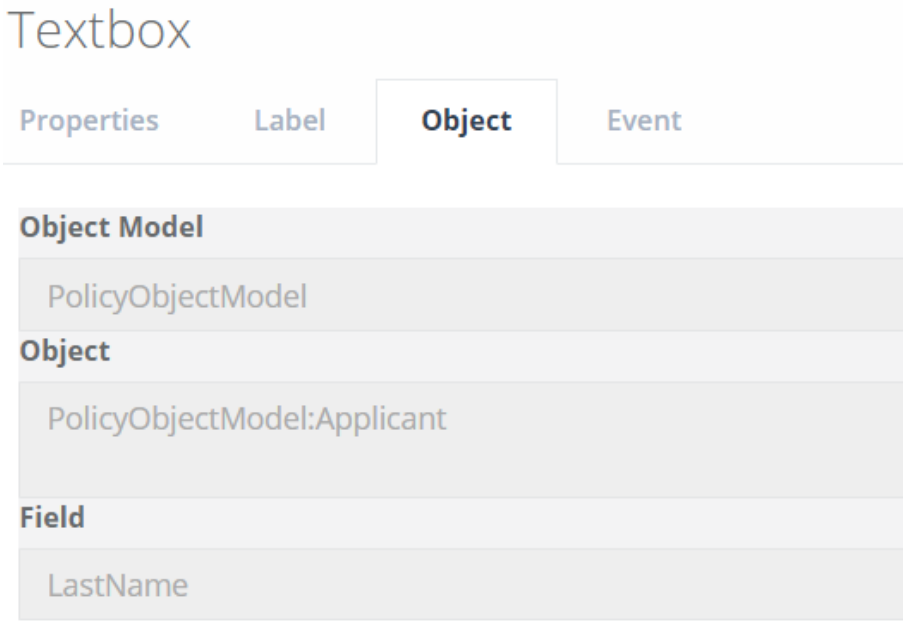
Applicant Page



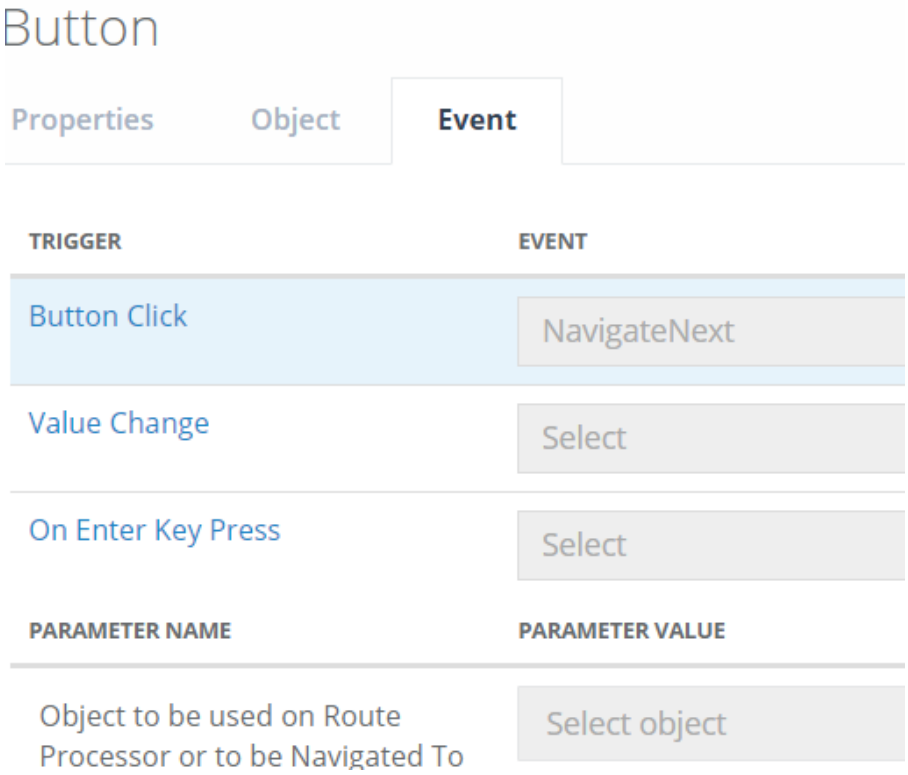
FirstName field



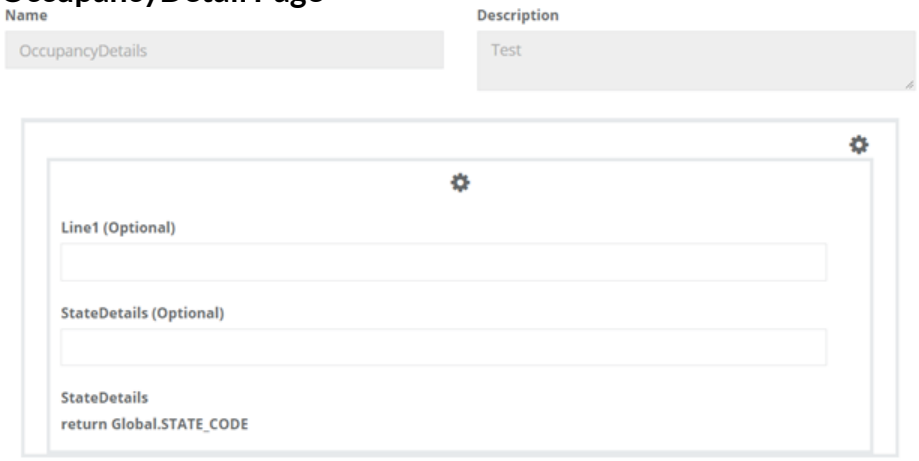
LastName field



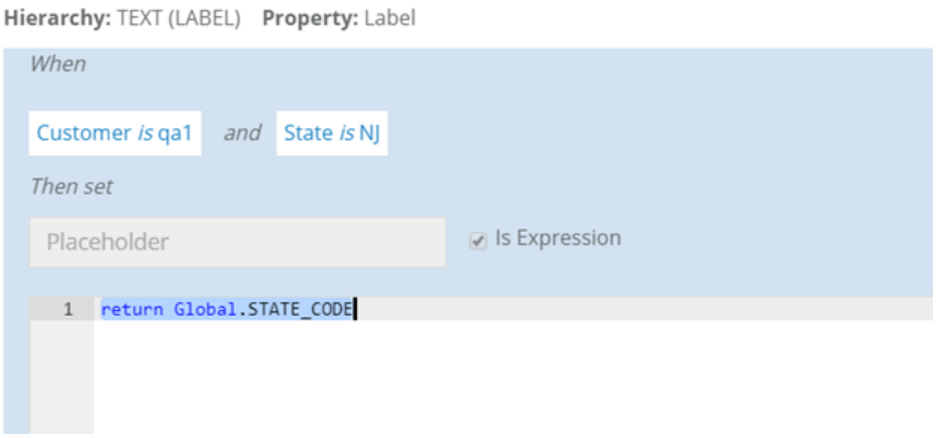
Navigate Next button



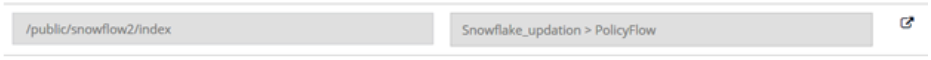
OccupancyDetail Page



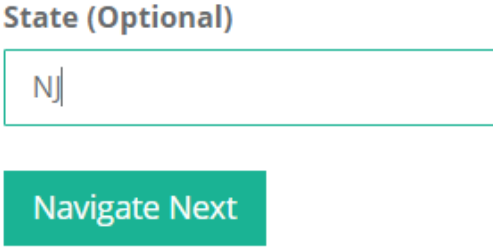
State details field Rule



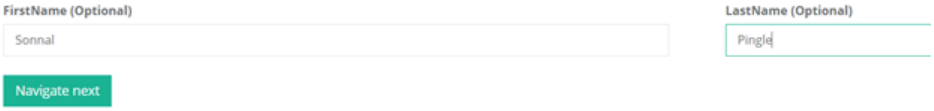
Verify at runtime



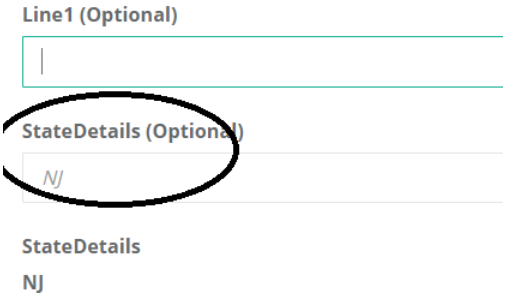
On Policy page, enter state.



On Applicant page, Enter first name and last name and click on next.



On Occupancy Details page – rule executed successfully



7 Special Keywords

Special keywords are used along with relational operators for comparison

Keyword	Description
null	null keyword specifies empty value for a variable

Null keyword can be use in relational operation to compare whether variables/model variable value is empty. Different uses are- initializing null value, comparing null value of variable, comparing null value of object reference variable, finding a record having null value in order to insert value in it. If the page on which DSL is to be written is current page then, make sure that Navigate Next button of previous page has FQOP mapped as Vehicle object.

i]

```
def x = null
if x = null then
x = 1
end
return x
```

Result of above execution block will be 1

ii]

Consider a hierarchy of MasterModel where we have PersonalAuto [with Premium field] as a root object, which is pointing to Vehicle object [with Make and Premium Field] and Vehicle object is pointing to Coverage object [having fields CoverageCode, Limit, Premium, and Description]. MasterModel - PersonalAuto --> Vehicle-->Coverage

```
def x as new MasterModel:Policy.Insured.PersonalAuto.Driver
if x.DriverName = null then
x.DriverName = "Digital1st Driver"
end
return x
```

Result of above execution block will be - if DriverName not already provided then it will be set to - Digital1st Driver.

iii]

return This.Vehicle[DriverName=”Harry Potter”].Premium

iv]

```
def x = lookup RefCoverage to return RefCoverage.Code as code ,
RefCoverage.Description as DESC
if This.Vehicle = null
then
                          # verifying if entire object itself is null
def veh as new PersonalAutoModel:PersonalAuto.Vehicle
This.Vehicle = veh
end
```

#Result of above block will be - if current value for vehicle object is null then value will be assigned to it.

iv]

```
def x = lookup RefCoverage to return RefCoverage.Code as code ,
RefCoverage.Description as DESC
if This.Vehicle = null
then
                          # verifying if entire object itself is null
return This.Vehicle
end
```

#Result of above exception block will be -Null exception will be thrown at run time.

8 Special Characters

8.1 Comments

8.1.1 # Single line comment example

```
 #def x as 5
 def x as 10
if x = 5
then x=x+5
end
return x
```

Output of above snowflake code is 10.

8.1.2 #multiple lines comment example

```
def x as 0, y as 5

if x = null and y >= 5 then
x = 1
else
/* if y >= 10 and y < 15 then
x = 2
end */
if x= 0 and y >= 5 then
x = 2
end
end
return x
```

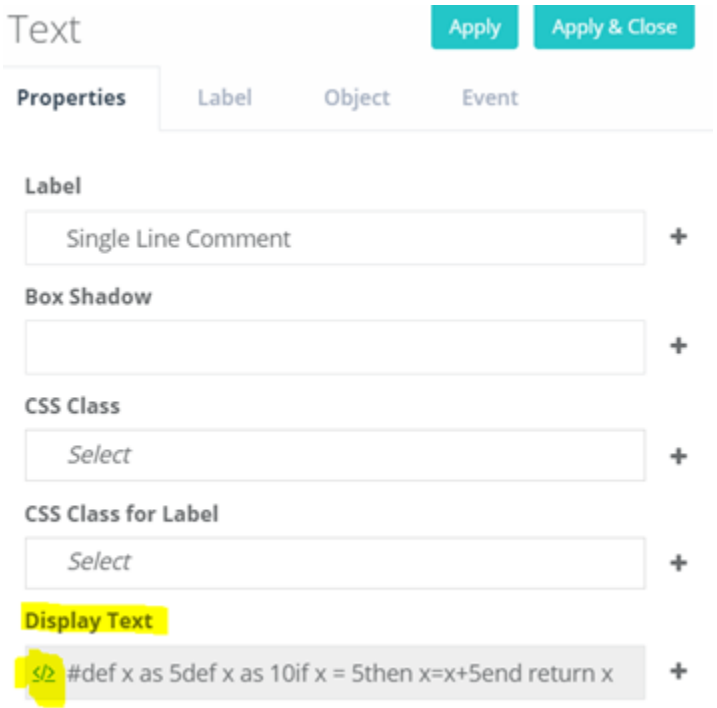
Output of above snowflake code is 2.

8.2 Designer level illustration

```
Single Line Comment
#def x as 5
def x as 10
if x = 5
then x=x+5
end
return x
```

```
Multiple Lines Comment
def x as 0, y as 5
if x = null and y >= 5 then
x = 1
else
/* if y >= 10 and y < 15 then
x = 2
end */
if x= 0 and y >= 5 then
x = 2
end
end
return x
```

8.2.1 Written on Expression box available on 'Display Text' property of text widget



Expression

```
1  #def x as 5
2  def x as 10
3  if x = 5
4  then x=x+5
5  end
6  return x
```

Please perform object mapping and save page before entering logic

8.3 Runtime illustration

Single Line Comment

10

Multiple Lines Comment

2

9 Flow Control Statement

Flow control statements allows code to be executed based on a given condition and snowflake's control statements can be used in below mentioned ways on different editors.

- In an App - On a page - any widget > Settings icon > anywhere the '+' icon is available > Advanced logic in when configuration
- In an App - On a page - any widget > Settings icon > anywhere the '+' icon is available > is expression in then condition
- In an App - On a page - any widget > Settings icon > anywhere the '<>' icon is available > Expression editor
- In an App - On a page > In inner row widget > Click here to enter logic' link > Expression editor
- In an App - On a flow route > Settings icon > Event/Expression editor
- In an App - In a report > Expression editor Conditional statement

9.1 Conditional Statements

9.1.1 if-then-else

The if-then-else statement, including nesting, can be used to evaluate conditions and invoke a set of instructions based on the results of the condition. Example:

```
def a, b, x, z as 7
if x greater than 5 then
b = 3
else
b = 0
end

if z = 6 then
a = 4
end
return a * b
```

Result of above execution block will be null

9.2 Looping statements

Looping statements allow the user to write expression that can iterate over collection of objects.

9.2.1 foreach...end

A foreach loop permits code to operate upon a collection of items. Typically, a parent object will iterate over a collection of child object instances.

Example: Using the Coverage Premium scenario, code can be written to loop through each of the CoverageCode records for a given Vehicle using the following syntax:

```
def x
x = lookup RefCoverage to return RefCoverage.Code as code ,
RefCoverage.Description as desc
foreach x as y do
def code as y.code
def cov as new MasterModel:Vehicle.Coverage
cov.CoverageCode = y.code
This.Vehicle.Coverage = cov
end

def totalPremium as 0
foreach This.Vehicle.Coverage as cov do
def limit
limit = lookup RefLimit to return RefLimit.Value as value having
RefLimit.CoverageCode = cov.CoverageCode
cov.Limit = limit.value
cov.Premium = limit.value / 100
totalPremium = totalPremium + cov.Premium
This.Premium = totalPremium
end
```

In above example:

1.
Consider a hierarchy of MasterModel where we have PersonalAuto [with Premium field] as a root object, which is pointing to Vehicle object [with Make and Premium Field] and Vehicle object is pointing to Coverage object [having fields CoverageCode, Limit, Premium, and Description]
MasterModel - PersonalAuto --> Vehicle-->Coverage

2.
In same content, we have two reference tables RefCoverage[Code and Description] and RefLimit[CoverageCode and Value]

3.
At app level, Pages will include two pages
3.1.Vehicle Page which will capture Vehicle Premium and Vehicle Make values.
3.2.Coverage Page which will display sum of Premiums for all coverage codes.
Based on formula : cov.Premium = limit.value / 100

4.

We have to write above code snippet while navigating from Vehicle page to Coverage page.

5.
Below code snippet is required to create the coverage instances based on the coverages defined in RefCoverage reference table, if we dont write this we will get totalPremium as 0. Here x references to RefCoverage table values and with the help of foreach loop we iterate through all the instances of x and for respective instance we store the value present in RefCoverage table's code,
in CoverageCode field. Eventually assigning this instance of coverage (cov) to current instance of Coverage object.

```
def x
x = lookup RefCoverage to return RefCoverage.Code as code,
RefCoverage.Description as desc
foreach x as y do
def code as y.code
def cov as new MasterModel:Vehicle.Coverage
cov.CoverageCode = y.code
This.Vehicle.Coverage = cov
End
```

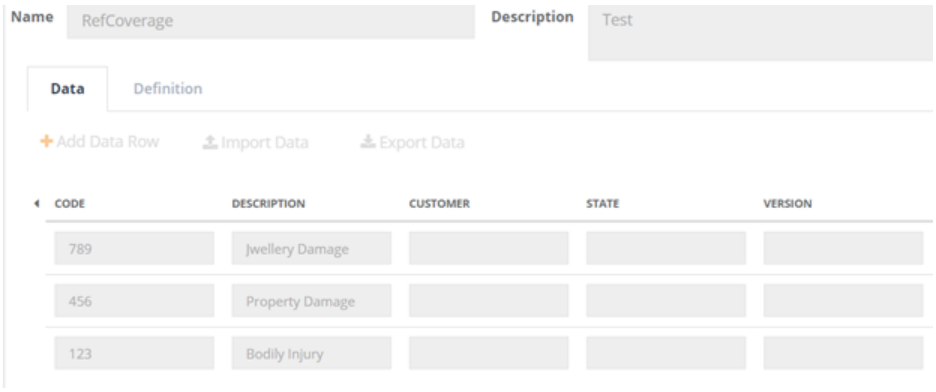
6.
Through below code we are comparing the codes present in RefLimit table with RefCoverage table's indirectly via cov.CoverageCode statement, once comparison is matched we are calculating cov.Premium, eventually taking sum of all coverage codes.

```
def totalPremium as 0
foreach This.Vehicle.Coverage as cov do
def limit
limit = lookup RefLimit to return RefLimit.Value as value having
RefLimit.CoverageCode = cov.CoverageCode
cov.Limit = limit.value
cov.Premium = limit.value / 100
totalPremium = totalPremium + cov.Premium
This.Premium = totalPremium
end
```

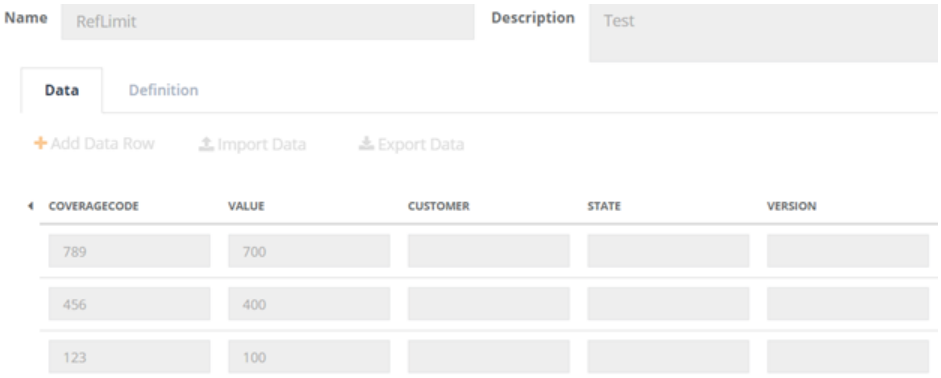
9.2.1.1 To illustrate above scenario follow below mentioned steps:

AppName: Snowflake_updation
FlowName: PolicyFlow

Create Reference tables as below:
RefCoverage reference table

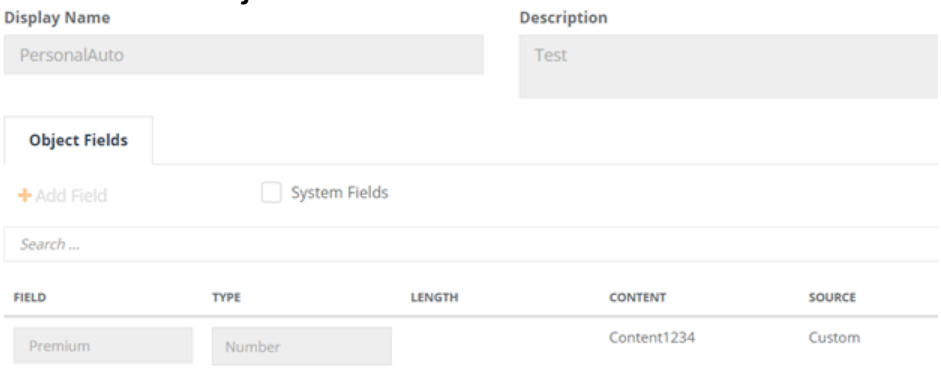


RefLimit reference table

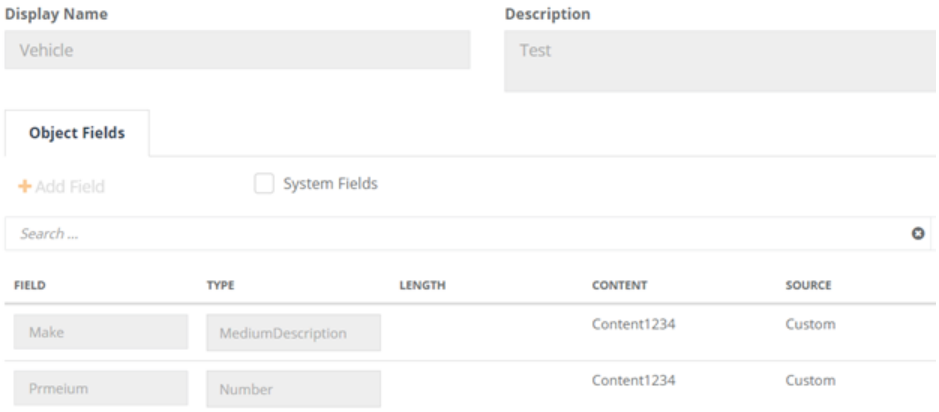


Create objects as below:

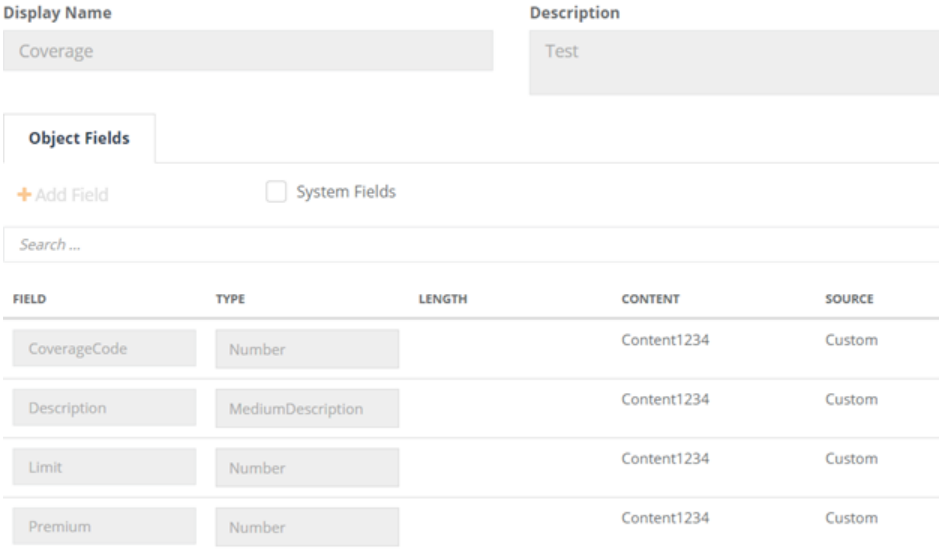
PersonalAuto object



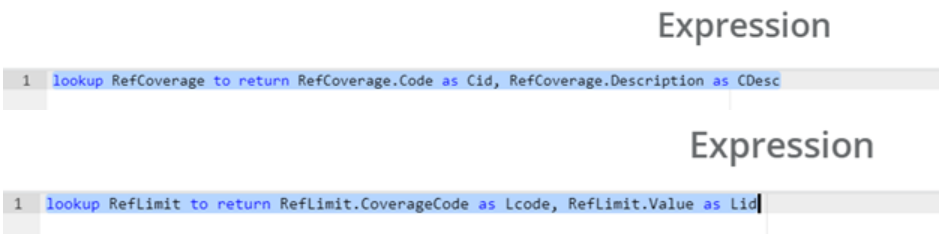
Vehicle object



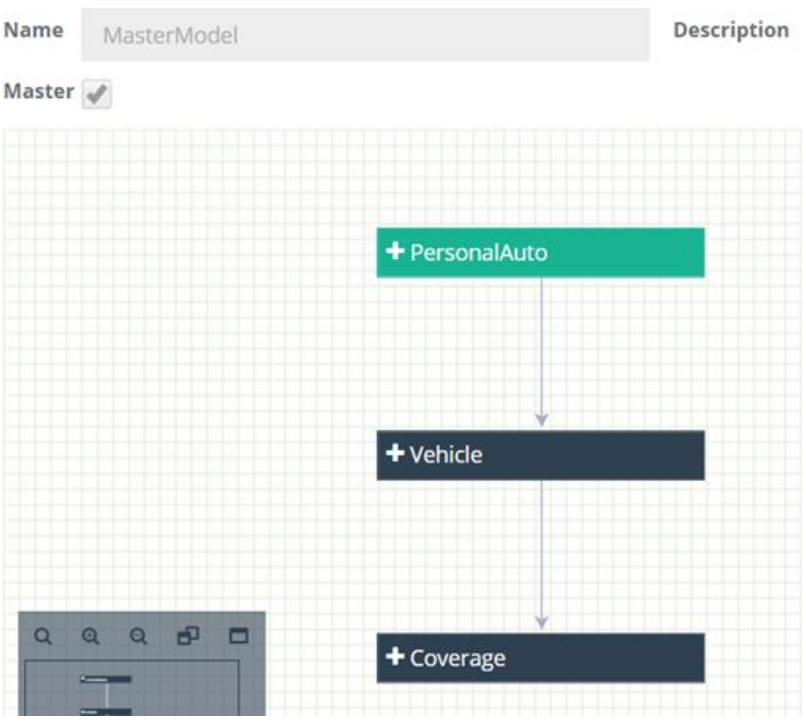
Coverage object



Write below mentioned expressions on CoverageCode and Limit fields to fetch data through RefCoverage and RefLimit reference tables respectively.

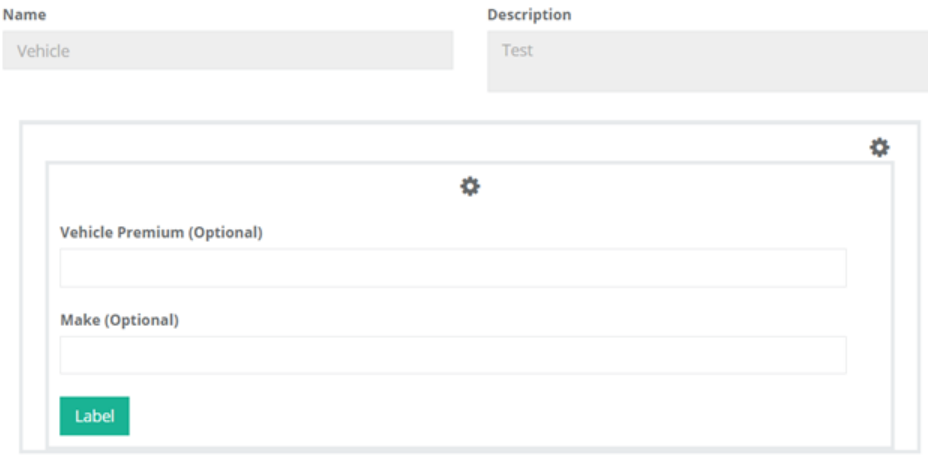


Create object model as below:

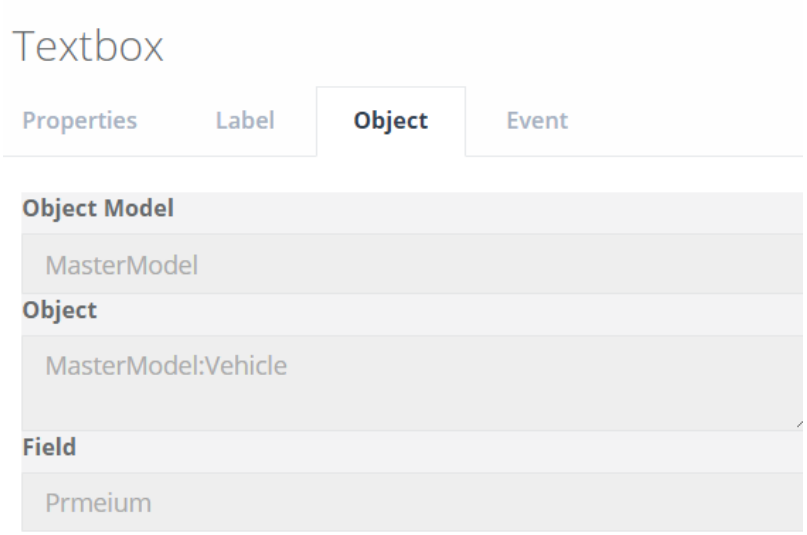


Create Pages as below:

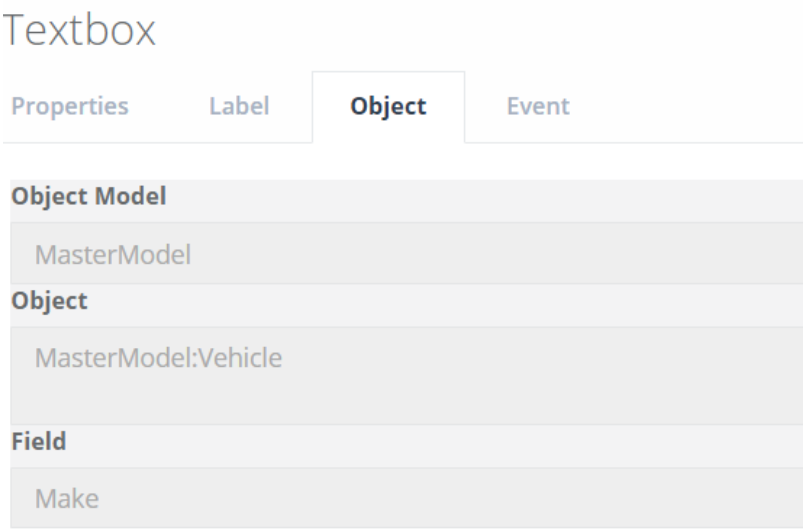
Vehicle page



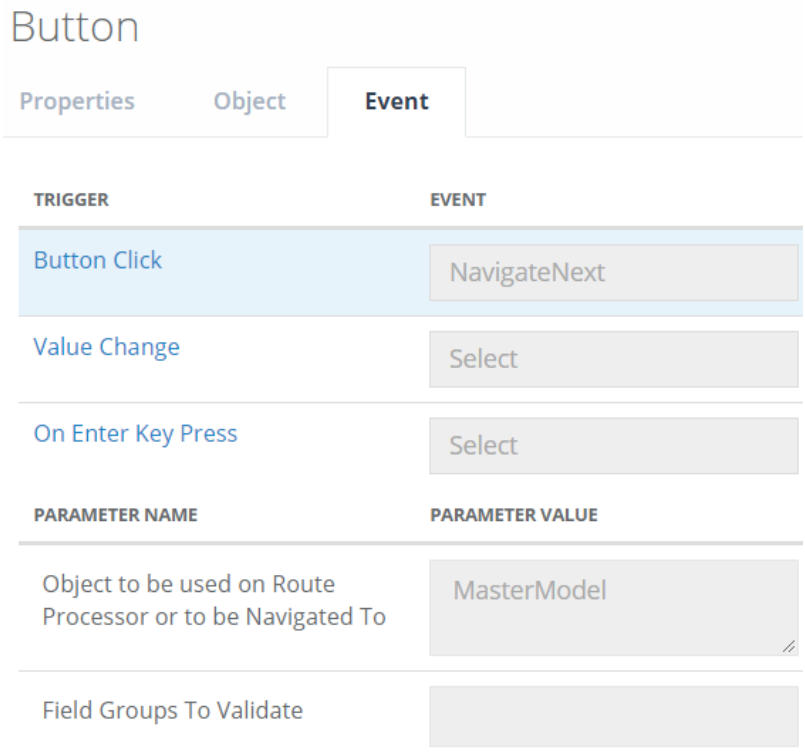
Vehicle Premium field



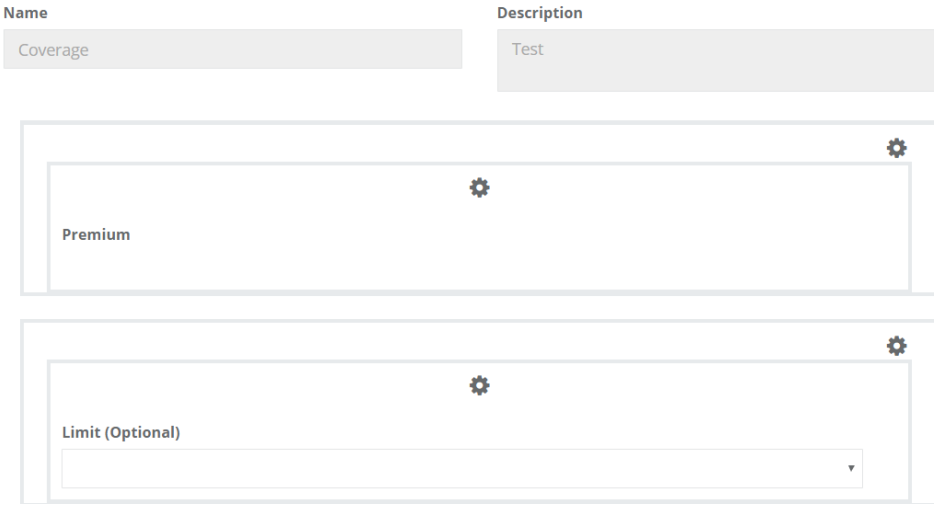
Make field



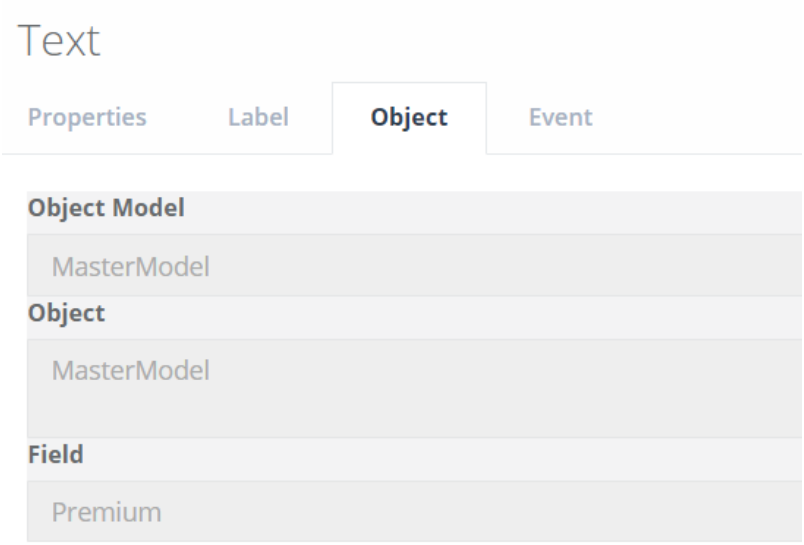
Navigate next button



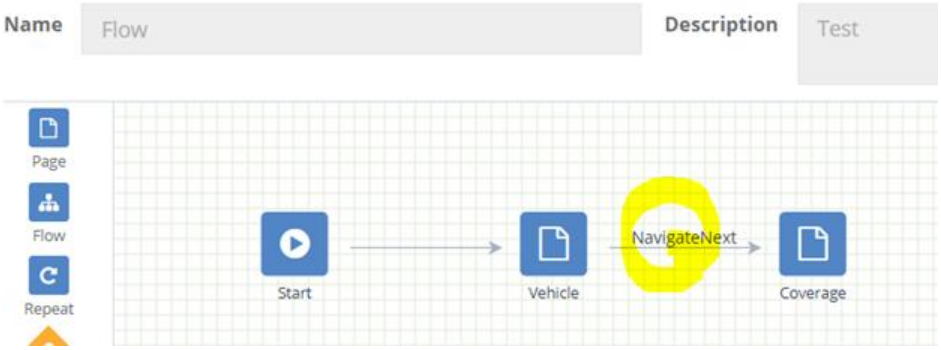
Create Coverage page



Premium field

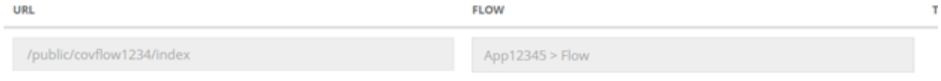


Create flow as below

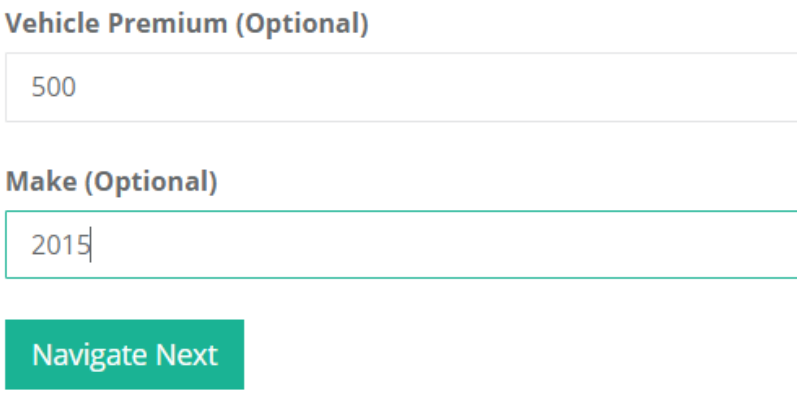


Write DSL on highlighted route

Verify runtime



On vehicle page, enter vehicle Premium and make field value



On coverage page

Premium

12.00000000

9.2.2 for...end Statement

A for statement permit code to iterate/execute the block for specified value. Syntax of for block is

```
for <variable> as <startcounter> to <endcounter> increment by
<incrementcounter> do
<statements>
end
```

Example 1

```
# Increment by keyword is optional and default increment counter is 1.
# e.g. Below for block will iterate for 10 times

def a as 0

for x as 1 to 10 do
a = a + 5
end

return a
# Result of above execution block will be 50
```

Example 2

```
# Increment by will increment the counter by 2
# e.g. Below for block will iterate for 5 times

def a as 0, e as 10

for x as 1 to e increment by 2 do
a = a + 5
```

```
end

return a
# Result of above execution block will be 25
```

10 Branching Statements

The branching statements allow one to define expression to exit from the execution blocks created by flow control statements.

For example DSL can be written on :

- In an App - On a page - any widget > Settings icon > anywhere the '+' icon is available > Advanced logic in when configuration
- In an App - On a page - any widget > Settings icon > anywhere the '+' icon is available > is expression in then condition
- In an App - On a page - any widget > Settings icon > anywhere the '<>' icon is available > Expression editor
- In an App - On a page > In inner row widget > Click here to enter logic' link > Expression editor
- In an App - On a flow route > Settings icon > Event/Expression editor
- In an App - In a report > Expression editor

statement	Description
return	Return statement is used to end the execution of the rule and return the value from execution block to caller
break	Break statement is used in for or foreach statement block to break the execution of repeating block
continue	Continue statement is used to skip the execution of current iteration and move to the next iteration of block

10.1 Return Statement

Return statement is used to end the execution of the rule and return the value from execution block to caller.

In below example, code will verify the Color field of the vehicle, if Color is "Red" then it will return value as "Supported" or else return value as "Not Supported" to the caller. In a scenario where user is writing the Snowflake expression in Rule When construct, user can use only "execute rule" expression.

```
if x = "Red" then
return "Supported"
else
return "Not Supported"
end
```

- In a scenario where user is writing the Snowflake expression in Rule When construct, user can use only "execute rule" expression.

```
        def x as This.Color

 if x = "Red" then
        execute rule
        end
```

10.2 Break Statement

Break statement is used in for or foreach statement block to break the execution of repeating block.

Example

```
def a as 5, z

for x as 1 to a * 2 increment by 2 do
z = x * 5

if x >= 4 then
break
end
end

return z
# Result of above execution block will be 25
```

10.3 Continue Statement

Continue statement is used to skip the execution of current iteration and move to the next iteration of block.

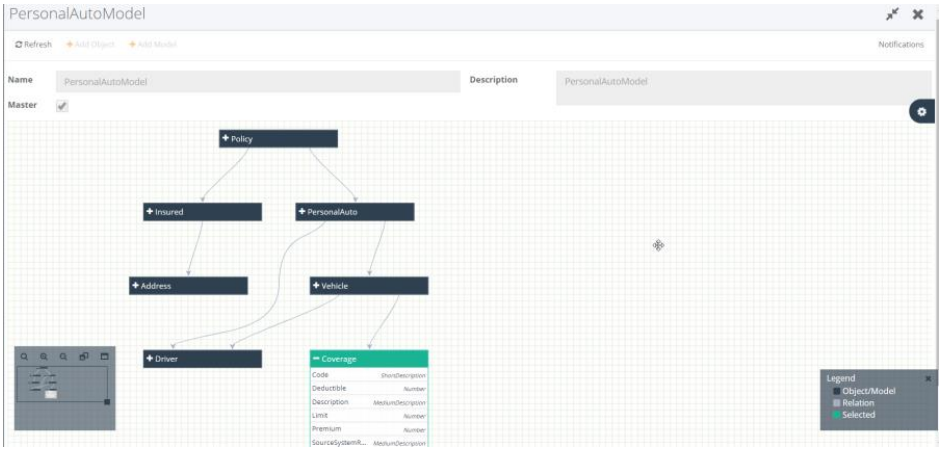
Example

```
def a as 5, y

for x as 1 to a do
if x >= 2 then
continue
end
y = x * 5
end
return y
# Result of the above execution block will be 5
```

11 Model Aware Syntax

Snowflake adapts to a new or evolving domain object model. Its ability to reference the domain objects and its fields with easy to use syntax for basic CRUD operations on the objects makes Snowflake a very powerful domain specific language. The domain objects can be defined or extended through object designer and Snowflake adapts to these changes. Object models consist of objects in tree structure, this essentially puts all objects and (object models within master model) in a parent-child format. For understanding these powerful capabilities and syntax structure, we will use the following example from a hypothetical policy object model:

11.1 Example Model



Follow below relations to achieve above structure of Object Model.

- Above image has an Object Model named as 'PersonalAutoModel'-
- There is a root object named 'Policy' (having fields - 'PolicyNo')
- 1st leg of 'Policy' object=> has a child object 'PersonalAuto' (having fields- 'Premium').
  'PersonalAuto' has a child object 'Vehicle'(having fields- 'Make', 'Premium').
  'Vehicle' has a child object 'Driver'(having fields- 'DriverName') and 'Coverage'(having fields- 'CoverageCode', 'Description', 'Limit', 'Premium').
  Also there is a link so that 'Driver' object is also a child of 'PersonalAuto' object.
- 2nd leg of 'Policy' object=> has a child object 'Insured'(having fields- 'InsuredName').
  'Insured' has a child called 'Address'(having fields- 'Street').

Examples of referencing a 'Vehicle' object and its fields:

Object Reference	Description
return This.Vehicle	Reference to the entire vehicle object

Object Reference	Description
return This.Make	Reference to the Make field from Vehicle object
return Make	Reference the current object's Make field
return This.Driver	Reference to the Driver object
return This.Driver.Name	Reference to the Driver objects Name field that is mapped to vehicle object
return This.Coverage	Reference to the Coverage object or field of the Vehicle object
return This.Coverage \[Code=vCoverageCode (and/or?)\]	Reference to the Coverage object mapped to Vehicle object that uses filter criteria to return a list of Coverages
return This.Coverage \[Code=vCoverageCode (and/or?)\].Premium	Reference to the Coverage object's Premium field mapped to Vehicle object that uses filter criteria to return a list of Coverages based on coverage code
return Parent.Premium return This.Parent.Premium	As current context is Vehicle object and if you want to access the Premium from parent, you can use Parent keyword to refer parent object field. In this example, it will try to get the 'Premium' value from 'PersonalAuto' object.

'This' keyword

- The user will be able to reference the model of an object being evaluated (such as policy, quote, claim, etc.). This keyword is useful for current object and its fields. Also it can be used to traverse its children and parent objects and their respective children. ancestor, child or peer of the current object.(how to access peer object using This keyword? or is there any other keyword for it?)
- If there is business logic associated with the Vehicle object, it can reference its own attributes by simply typing the attribute name. The user can optionally use the keyword **This** to represent the current object.
- If you are writing an advance logic in rule on any field in vehicle, you can reference vehicle field using either This keyword or simply referencing the field name.

'Parent' keyword

- If the user wants to access attributes of its parent, they can use the **Parent** keyword.
- For example, if the user is working on an algorithm associated with the Vehicle and wants to access information regarding the Personal Auto parent object, the **Parent** keyword can be used. The **Parent** keyword will allow the user to access all of the behavior of the parent object, including access of its children, just as if you were within code for the object itself.
- The user will be able to select from the long format of the object and attribute names so it will be more familiar to them, as opposed to the database table and

column names. The long format of each will be captured in the metadata definition during the process of creating the object model.

11.2 Possible usage of above code

Case 1:
If any code for accessing Object Models is being written on flow route between two pages, then make sure Event FQOP of 'Next' button of the previous page is mapped to Vehicle Object. If local variables are created on a route then there value is lost once the execution flow progresses to next page. If the value is stored in object reference variables and then assigned to the current object then that is retained because start transaction() is used at start of the flow hence all of the objects get stored in permanent memory.

Case 2:
If any code for accessing Object Models is being written on a page in an editor of a 'Display Text' widget, then make sure current event is mapped to Vehicle Object through widget's properties. If local variables are created on a route then there value is lost once the execution flow progresses to next page. If the value is stored in object reference variables and then assigned to the current object then that is retained because start transaction() is used at start of the flow hence all of the objects get stored in permanent memory.

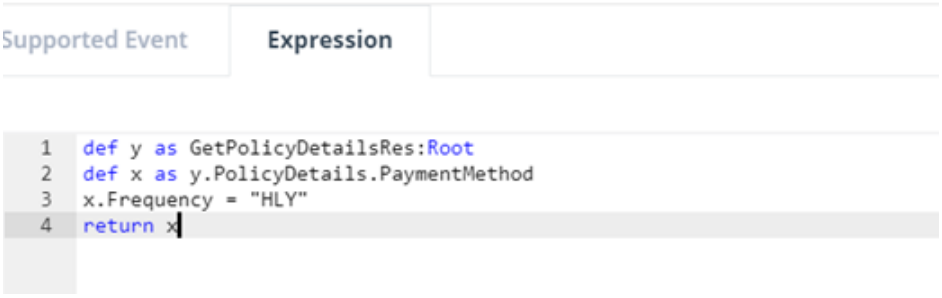
11.3 Use of '>' separator instead of "." Separator

Consider you have objects named Policy, PolicyDetail and PaymentMethod, using this object you created object model GetPolicyDetailsRes (Policy -> PolicyDetail -> PaymentMethod) . Now when you try to access PaymentMethod using object model notation you can use syntax as

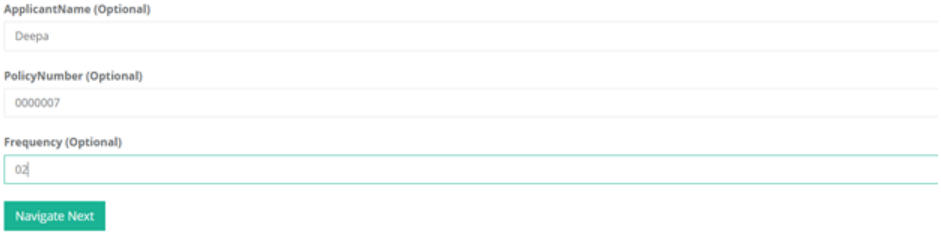
def x as GetPolicyDetailsRes:PolicyDetail.PaymentMethod
Now consider that you have added a field PaymentMethod in PolicyDetail object and you want to access the PaymentMethod relation instead of PaymentMethod field inside PolicyDetail. In that case you can use '>' relation separator instead of "." as default field separator.

def y as GetPolicyDetailsRes:Root def x as y.PolicyDetails>PaymentMethod x.Frequency = "MONTHLY" return x

Refer attached screen for more details. App name : AppPolicyDetails Flow name : PolicyFlow

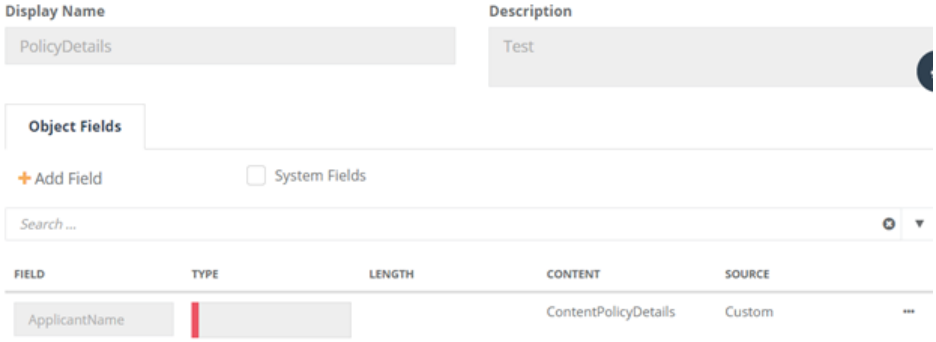


Take input on PolicyDetails Page



Check output on PaymentMethod page

Now add, PaymentMethod field under PolicyDetails object Original

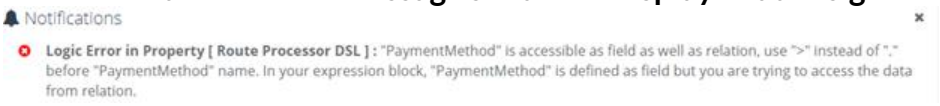


Add PaymentMethod field

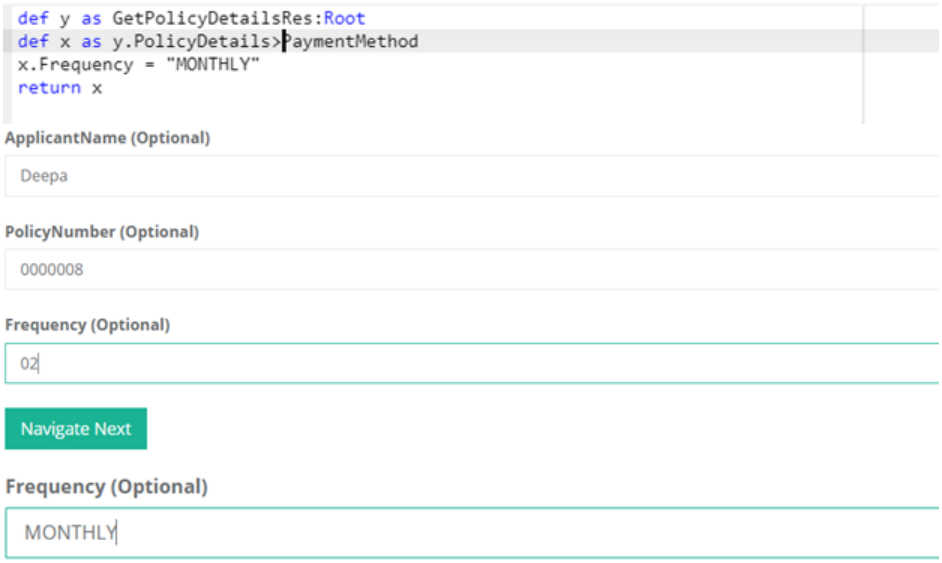
Now try to change DSL by setting x.Frequency = “MONTHLY”

```
def y as GetPolicyDetailsRes:Root
def x as y.PolicyDetails.PaymentMethod
x.Frequency = "MONTHLY"
return x
```

Below mentioned error message should be displayed at designer level.



As 'PolicyDetails' object now has 'PaymentMethod' as both 'child object' and 'object field'; To access the 'PolicyDetails' and 'PaymentMethod' relation we have to use relation separator.



12 Data Access with Predicates/Criteria

Predicates in Object aware statement, allows to access the Object values only if Predicate conditions are satisfied.

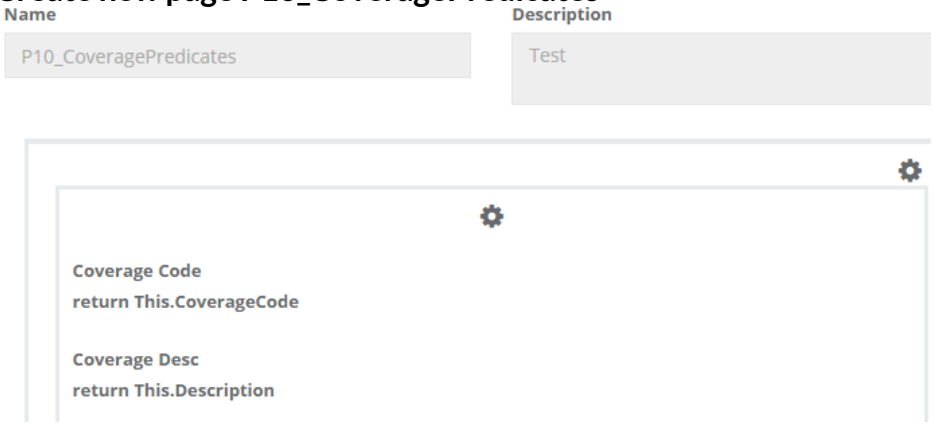
Consider a scenario where there are multiple coverage's stored under Vehicle Object. Now you want to access a Coverage object with CoverageCode as 123, If Coverage with 123 code is not available then we want to create the new Coverage Object instance and assign it to existing vehicle instance. Below statement will allow you to select the specific coverage object under vehicle object and if it's not available then we can create a new object instance and assign/map it to existing vehicle instance.

```
def x
x = lookup RefCoverage to return RefCoverage.Code as code ,
RefCoverage.Description as desc
if This.Vehicle = null then
 def veh as new MasterModel:PersonalAuto.Vehicle
This.Vehicle = veh end
foreach x as y do def code as y.code
if This.Vehicle.Coverage[CoverageCode=123] = null
 then def cov as new MasterModel:PersonalAuto.Vehicle.Coverage
cov.Description = "Bodily injuries & PTSD "
cov.CoverageCode = y.code
This.Vehicle.Coverage = cov
end
end
```

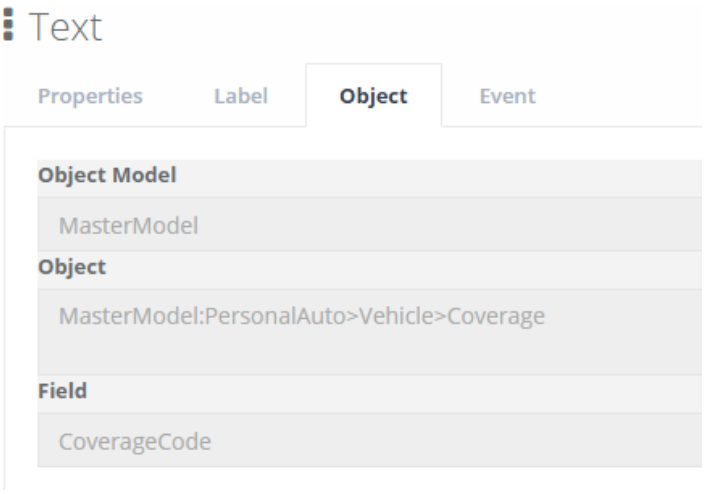
At designer level, we can continue with same example which is mentioned under 'multiple snowflake expressions'

Make following changes at designer level to verify above code.

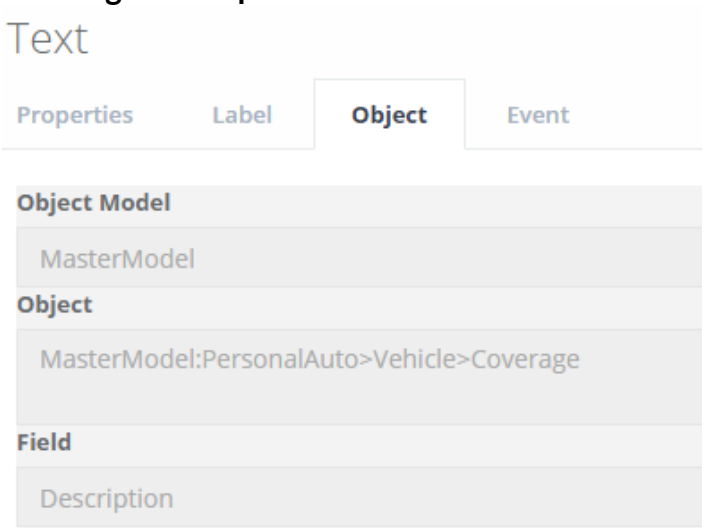
Create new page P10_CoveragePredicates



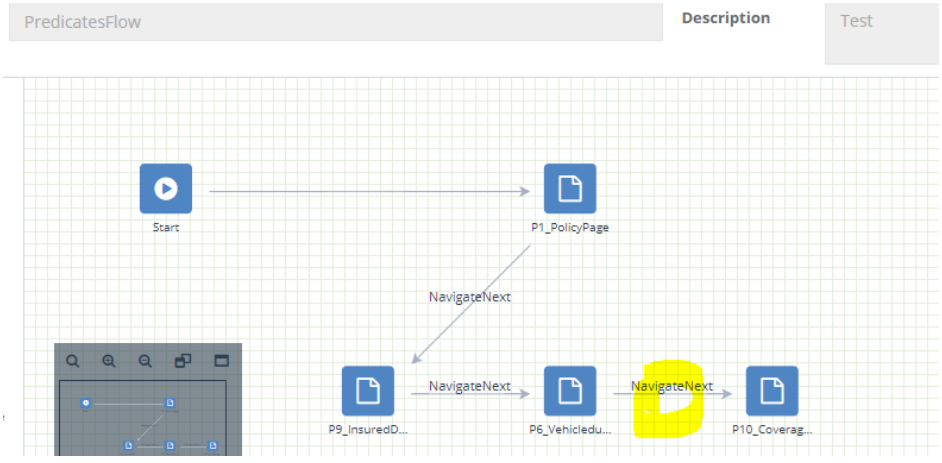
Coverage code field



Coverage Description field



Create new flow - PredicatesFlow



DSL

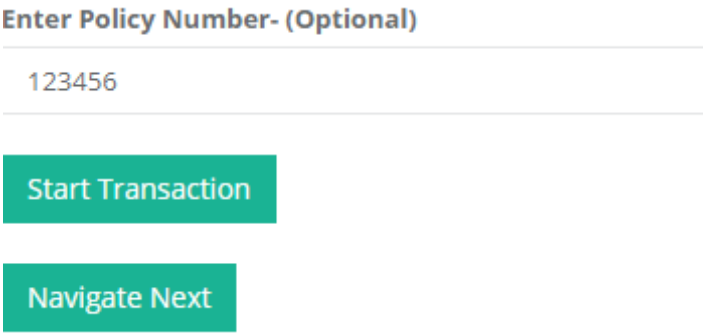
```
def x
x = lookup RefCoverage to return RefCoverage.Code as code , RefCoverage.Description as desc
if This.Vehicle = null then
def veh as new MasterModel:PersonalAuto.Vehicle
This.Vehicle = veh
end

foreach x as y do
def code as y.code
if This.Vehicle.Coverage[CoverageCode=123] = null then
def cov as new MasterModel:PersonalAuto.Vehicle.Coverage
cov.Description = "Bodily injuries & PTSD "
cov.CoverageCode = y.code
This.Vehicle.Coverage = cov
end
end
```

Runtime url

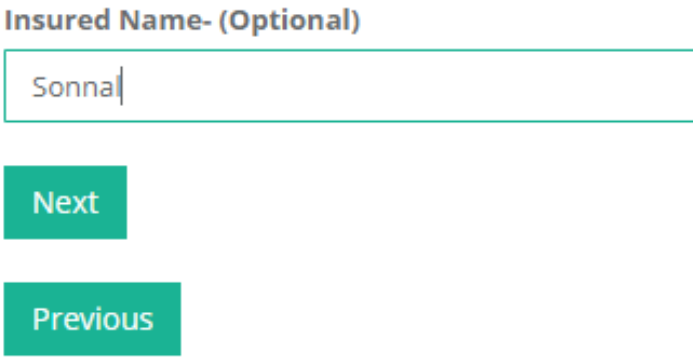
/public/predicate/index PolicyModelAccessApp > PredicatesFlow

Runtime outputs

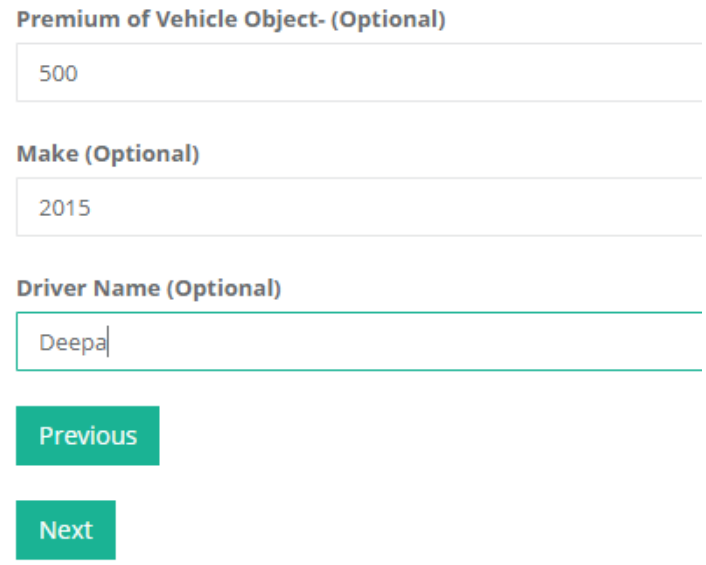
Enter Policy number and Click on navigate next.



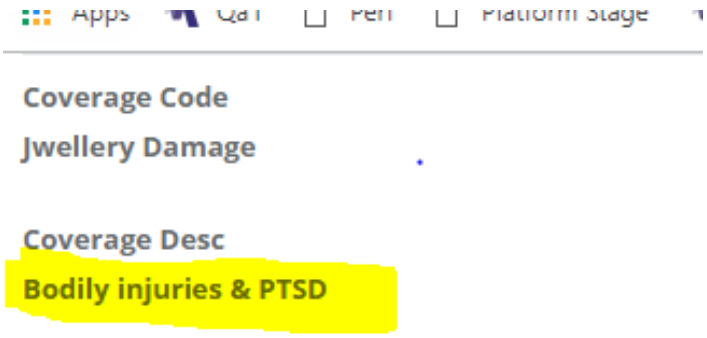
Enter Insured number and click on naviagte next.



Enter Premium amount, make and Driver name and click on navigate next



Verify DSL gets executed on Vehicle to Coverage Page and based on Coverage code value 123, its respective description gets changed to 'Bodily Injuries & PTSD'



13 Snowflake lookup statement for calling Policy Core System

Connector Service (Microservice) will host a service to fetch Lookup List from to Policy Core System. It will allow fetching of all Lookups available.

Snowflake will call this service by using below mentioned code:

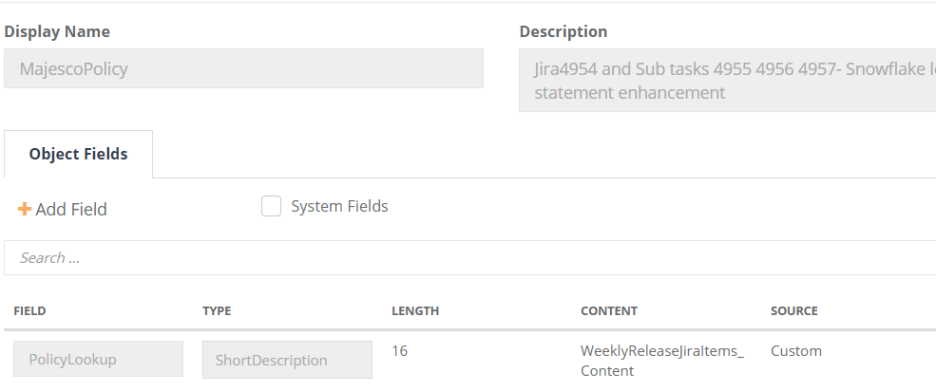
majescopolicy.lookup "Policy.CCurrency#836AB31910F87230E053E9F9FF0A7356" to return CCurrency,

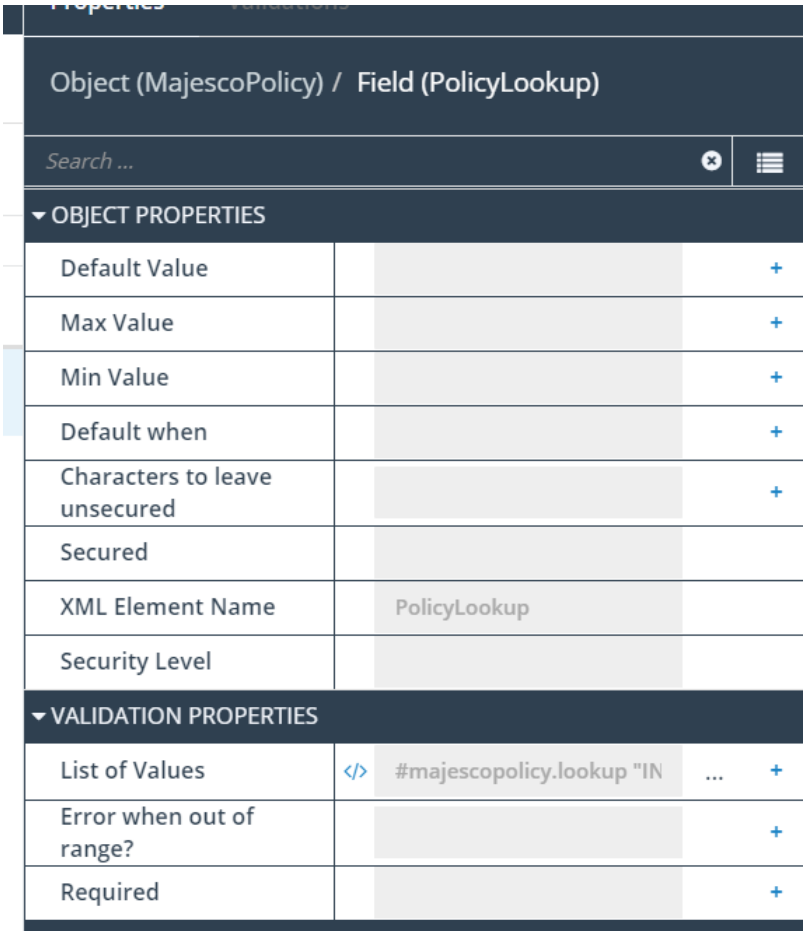
CCurrency having CPolicyStateCode="11", CModality="Riego", CCropName="Maíz", EffectiveDate="01/01/2018", CCycle="Primavera Verano"

Above code fetched the list of values under CCurrency column from 'Policy core system' based on having clause.

Refer below mentioned steps to use 'Policy core system' in Digital 1st:

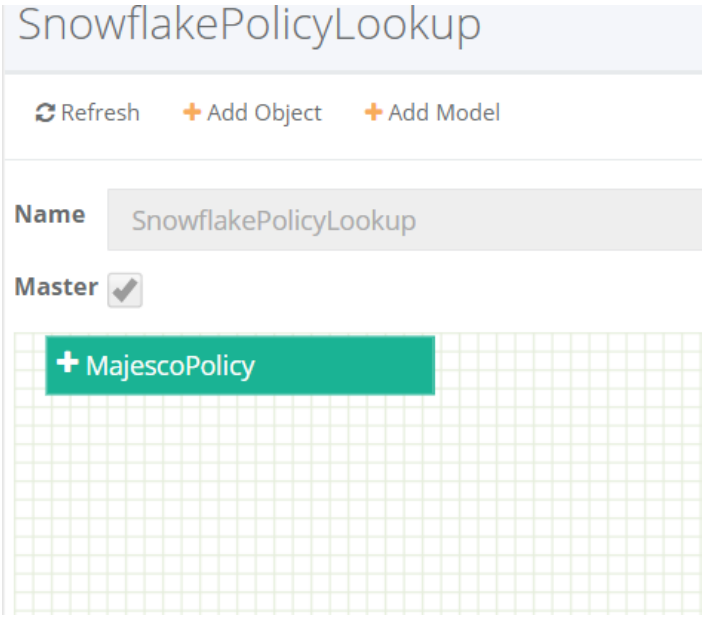
Create object as described below



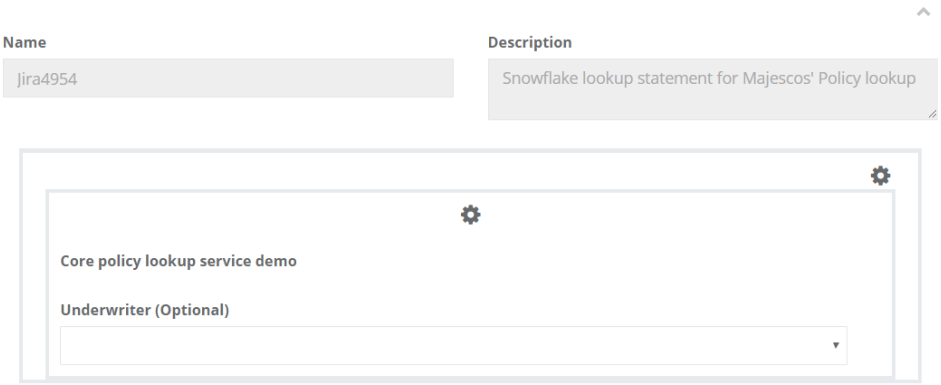


The above mentioned code should be written under 'Expression editor' or list of values property of an object filed

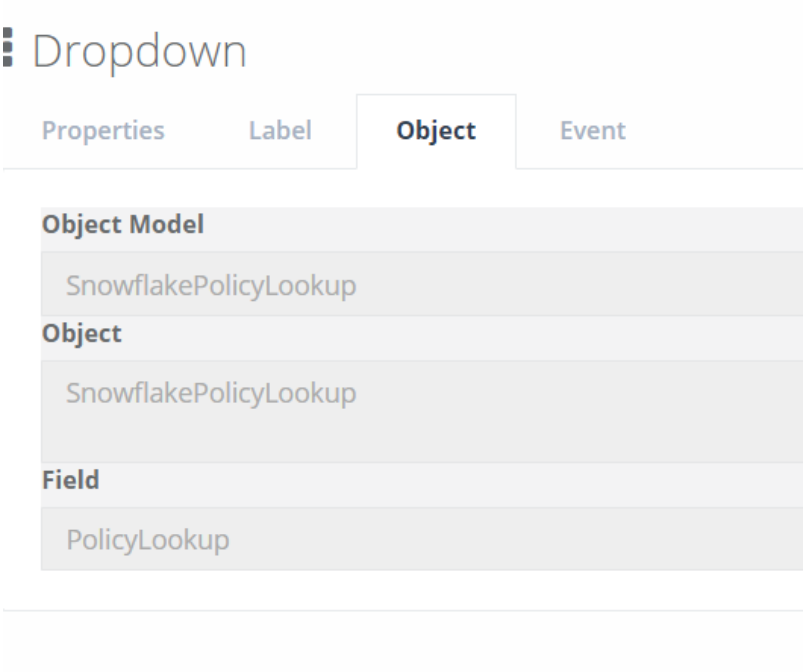
Further, create a master object model with this object



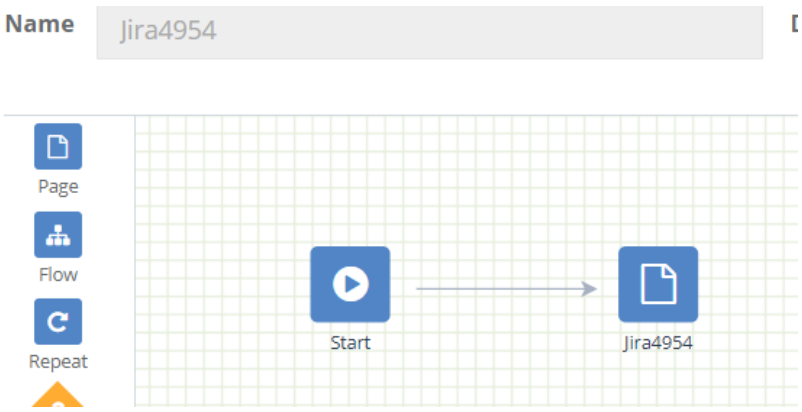
Go ahead; create a page described below, with dropdown widget



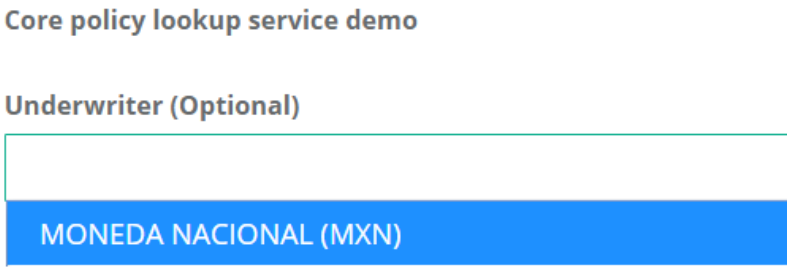
Now map, above object to this dropdown field.



Now, create a flow



Further utilize it at runtime



14 Select statement with conditions

Select statement is used in Report building to get the data from transaction tables as well as in advance logic, when you want to get the values from database without considering current context information or using Model aware statements. Select statement is mostly used in Report component for Reports and Charts. To achieve this it can be used with aggregate functions. In addition, it can be used with different clauses - order by, group by, having clauses. Below is format and syntax for achieving this

14.1 From clause

Case 1:

```
select statement without aggregate function and without any conditional
clauses =>  from + select

from {in memory} <modelName>  select <fieldReference> , <fieldReference>

keywords used in { } are optional keywords. In above expression "in memory"
is an optional keyword and can be used to execute the select expression
against in memory database. Default execution of select expression is to
executed queries against permanent database.
```

To show a report of the list of Quotes in pending status, we can use the Select statement in report module as below:-

```
from Policy  select Policy:QuoteNumber, Policy:EffectiveDate,
Policy:ExpirationDate,  Policy:Insured.MailingAddress.State  where
Policy:Status = "Pending"
```

14.2 Where clause

Case 2:

```
select statement without aggregate function and without any conditional
clauses => with where clause

from {in memory} <modelName>  select <fieldReference> ,
<fieldReference>  where <fieldCondition>
```

To show a report of the list of Quotes in pending status, we can use the Select statement in report module as below:-

```
from Policy  select Policy:QuoteNumber, Policy:EffectiveDate,
Policy:ExpirationDate,  Policy:Insured.MailingAddress.State  where
Policy:Status = "Pending"
```

14.3 Having clause

Case 3:

```
select statement without aggregate function and with a clause=>  having
from {in memory} <modelName>  select <fieldReference> ,
<fieldReference>  where <fieldCondition>,  having <fieldReference &
fieldCondition>
```

14.4 Group by clause

Case 4:

```
select statement without aggregate function and with a clause=>  group by

from {in memory} <modelName>  select <fieldReference> ,
<fieldReference>  where <fieldReference & fieldCondition>,   group
by <fieldReference>
```

To show a report of the list of Quotes where policy in pending status grouped by PolicyType, we can use the Select statement in report module as below:-

```
from Policy select Policy:QuoteNumber, Policy:EffectiveDate,
Policy:ExpirationDate,  where Policy:Status = "Pending"   group
by  Policy:PolicyType
```

14.5 Order by clause

Case 5:

```
select statement without aggregate function and with a clause=>  order by
from {in memory} <modelName>  select <fieldReference> ,
<fieldReference>  where <fieldCondition>,   order by <fieldReference>
```

14.5.1 Aggregate functions

Usage - To do some numerical functions on few fields, few functions are allowed to be used with select statement. Make sure parameters for these functions are columns of number data type.

Case 6:

```
select statement with aggregate function and with a group by clause=> sum()
+  group by

from {in memory} <modelName>  select <fieldReference> ,
<fieldReference>  where <fieldReference
& fieldCondition>,  sum(fieldReference)  group by <fieldReference>
```

To show a report of the list of Quotes having sum of all premiums where policy in pending status, we can use the Select statement in report module as below:-

```
from Policy select Policy:QuoteNumber, Policy:EffectiveDate,
Policy:ExpirationDate,  where Policy:Status =
"Pending"  sum(Policy:Premium)  group by  Policy:PolicyType
```

Case 7:

```
select statement with aggregate function and with a group by
clause=> avg() + group by

from {in memory} <modelName>  select <fieldReference> ,
<fieldReference>  where <fieldReference
& fieldCondition>,  avg(fieldReference)  group by <fieldReference>

from Policy select Policy:QuoteNumber, Policy:EffectiveDate,
Policy:ExpirationDate,  where Policy:Status =
"Pending"  avg(Policy:Premium)  group by  Policy:PolicyType
```

Case 8:

```
select statement with aggregate function and with a group by
clause=> count() + group by

from Policy select Policy:QuoteNumber, Policy:EffectiveDate,
Policy:ExpirationDate,  where Policy:Status =
"Pending"  count(Policy:Premium)  group by  Policy:PolicyType
```

Case 9:

```
select statement with aggregate function and with a group by
clause=> min() + group by

from Policy select Policy:QuoteNumber, Policy:EffectiveDate,
Policy:ExpirationDate,  where Policy:Status =
"Pending"  min(Policy:Premium)  group by  Policy:PolicyType
```

Case 10:

```
select statement with aggregate function and with a group by
clause=> max() + group by

from Policy select Policy:QuoteNumber, Policy:EffectiveDate,
Policy:ExpirationDate,  where Policy:Status =
"Pending"  max(Policy:Premium)  group by  Policy:PolicyType
```

Case 11:

Aggregate functions can be used with =>

- only having clause

- only group by clause
- only order by clause
- order by and group by clauses together
- order by and group by and having clauses

Many combinations are possible. Below is the example for using all of the above:-

```
select statement with aggregate function and with all clauses=>

from {in memory} <modelName>  select <fieldReference> ,
<fieldReference>  where <fieldCondition>,  sum(fieldReference)  order by
<fieldReference>

from Policy select Policy:QuoteNumber, Policy:EffectiveDate,
Policy:ExpirationDate,  where Policy:Status =
"Pending"  max(Policy:Premium)  group by  Policy:PolicyType having
Policy:PolicyState="NJ"
```

14.5.2 Sorting

Keyword	Description
ASC	Sorts the records in ascending manner
DESC	Sorts the records in descending manner

If sorting condition is not specified then system takes ascending order by default.

Case 12:

```
select statement with aggregate function and with a clause=>  sum() + group
by + having + order by + ASC/DESC

from {in memory} <modelName>  select <fieldReference> ,
<fieldReference>  where <fieldCondition>,  group by <fieldReference>  having
<fieldReference>   ASC/DESC
```

14.6 All combinations

Case 13:

```
select statement with aggregate function and with a clause=>  sum() + group
by + having + order by + ASC/DESC

from {in memory} <modelName>  select <fieldReference> ,
<fieldReference>  where <fieldCondition>,  sum(fieldReference)  group by
<fieldReference>  having <fieldReference>  order by
<fieldReference>  ASC/DESC
```

14.6.1 Use of 'new' and 'def' keywords

Keyword	Description
def	defines a local variable
def <variable> as new <ObjectModel:Root>	defines an object instance of Root object
def <variable> as new <ObjectModel:RelationPath>	defines an object instance for the object of mentioned relation path.(can be used for any object)
new	creates an object instance without allocation of any value. (rarely used without 'def')

The new object statement is used to create a new local object instance that can be later assigned to the existing object instance for saving data into the database.

Case i] def :-

covered under defining variables section. Please navigate to variables.

Case ii] def <variable> as new <ObjectModel:Root> :-

To refer fields from root of the object model :-

def <variablename> as new <ModelName>:Root # To refer fields from root of the object model

Consider that we want to verify a Vehicle Vin using a third party Vin verification service. To verify the Vehicle information with third party service, you need to send the data from your Vehicle instance from UI to the Third Party Vehicle object. With the assumption that the advance logic with Snowflake expression is written at a place where you have access to the Policy > Personal Auto >Vehicle object. The third party service expects the vehicle information in specific data format, we can use below new statement to create the instance of third party model and set the vehicle data from existing Policy > Personal Auto > Vehicle instance.

```
def tpVehicleVerification as new ThirdPartyVehicleVerification:Root
/*
This Snowflake expression will create a new local instance of
ThirdPartyVehicleVerification object model which can be later referenced using
local variable named "tpVehicleVerification"
*/
```

Case iii] def <variable> as new <ObjectModel:RelationPath> :-
To refer fields from any Object of the object model by giving correct hierarchy of that object

def <variablename> as new <ModelName>:<RelationPath> # To refer fields from any Object of the object model by giving correct hierarchy of that object
def tpVehicle as new ThirdPartyVehicleVerification:VehicleRisk
/*
Above statement will create a new instance of VehicleRisk object that will be used to copy data from current vehicle instance under personal auto
*/
tpVehicle.VehicleMake = This.Make
tpVehicle.VehicleYear = This.Year
tpVehicle.VehicleModel = This.Model
tpVehicle.VehicleIdentification = This.VIN
tpVehicleVerification. VehicleRisk = tpVehicle
statements will map the newly created VehicleRisk object to tpVehicleVerification instance */

Referring the root instance of the model instantiated above
return tpVehicleVerification

Case iv] new :-
To create instance of object/object model.
syntax - new MasterModel:Root # instance of object model
new MasterModel:Insurance # instance of Insurance object created

Linking Child Objects with 'using' Statement

- This keyword can be used in a DSL written in Page
- This keyword can be used in a DSL written on a flow route
- This keyword can be used in a DSL written in an Object Model

New Association statement is used to relate one object record with another object. E.g. In Personal Auto example using Snowflake expression, we can create a PersonalAuto object instance using def DSL statement. Once this instance is available, we can add this instance to existing policy instance using new association statement.
Syntax of new association statement is-

```
new <fieldReference> using <variableReference>
```

Example

App Name - PolicyModelAccessApp, Flow Name - ForUsingKeyword

We can use below Snowflake expressions to create a PersonalAuto instance and associate it with existing policy instance using DSL statements. Consider that currently you are defining an integration block with reference to Policy instance. This DSL can be written on a flow.

```
def dn as new Policy:PersonalAuto.Vehicle.Driver
dn.DriverName = "Harry Potter"
dn.Type = "Personal"
new This.Driver using dn
```

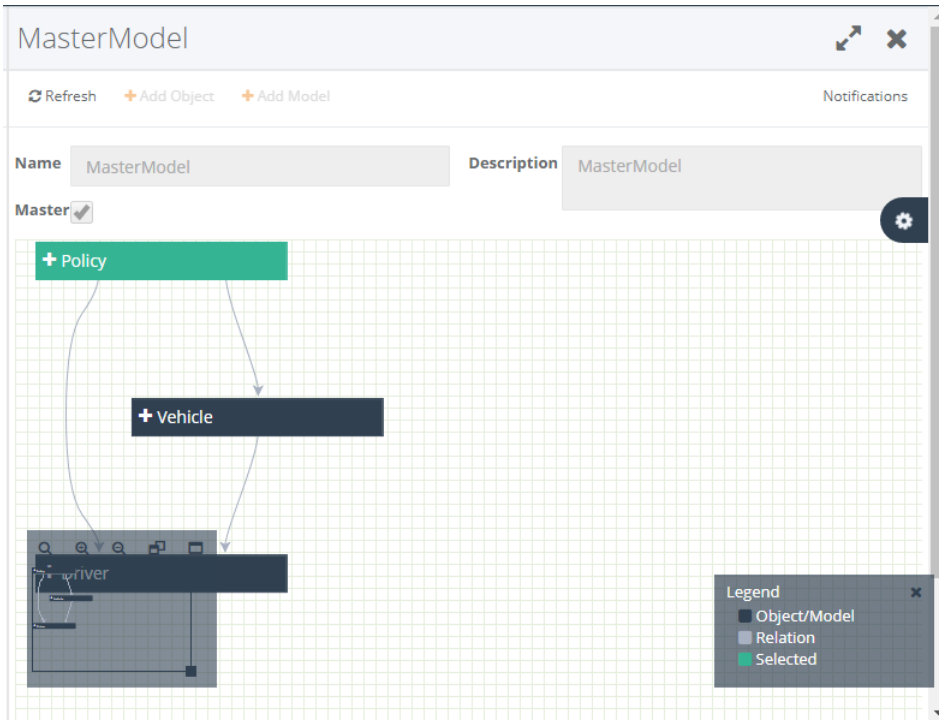
Alternate implementation of above Snowflake expression block is as below:

```
def dn as new Policy:PersonalAuto
dn.State = "Harry Potter"
dn.Type = "Personal"
This.PersonalAuto = dn
```

15 Deleting objects and Links between two objects

Keyword	Description
del	deletes an object
del association	deletes the link between the two objects OR deletes the link between an object model and an object (TBC) (in both cases the object is intact but is only hidden from accessing the data)

- This keyword can be used in a DSL written in Page
- This keyword can be used in a DSL written on a flow route
- This keyword can be used in a DSL written in an Object Model (TBC)



Case i] del

Example:
App Name - DeleteObjectAssociation1

If you want to delete the Driver object along with the link between Vehicle and Driver, you can use the 'del' statement without association keyword

```
def x as MasterModel:Root
del association x.Driver
```

Case ii] del association

Delete association statement is used to delete an association between two object instances. For example, if you want to delete a Coverage from under vehicle instance, we can use the delete association statement to delete the existing coverage. Syntax:-

```
del {association} <fieldReference>
```

Example:
App Name - DeleteObjectAssociation1

Considering you have access to the Vehicle instance, you can select a specific coverage object instance and delete link between Vehicle and Driver using delete statement. Delete association expression just delete the link, but keep the coverage object in database.

```
def x as MasterModel:Root
del association x.Driver
```

16 Sample code with Multiple Snowflake Expressions

Below Snowflake code should be written on a route navigation while going from Vehicle Page to Coverage page.

```
def x
x = lookup RefCoverage to return RefCoverage.Code as code ,
RefCoverage.Description as desc
if This.Vehicle = null then
def veh as new MasterModel:PersonalAuto.Vehicle
This.Vehicle = veh
end
```

/* in above code RefCoverage is a reference table which holds values for Coverage codes and Coverage descriptions. With lookup statement we are creating instances for all the coverage codes defined under RefCoverage reference table. Going ahead in above block, observe 'This.Vehicle = null', here we are checking whether 'Vehicle' instance is not yet initialized, and if this condition is true then we are creating a new reference 'veh' for Vehicle object and assigning that reference to current instance. [Point1, Point2 are achieved here.] */

```
foreach x as y do
def code as y.code
if This.Vehicle.Coverage[CoverageCode=code] = null then
def cov as new MasterModel:PersonalAuto.Vehicle.Coverage
cov.Description = y.desc
cov.CoverageCode = y.code
This.Vehicle.Coverage = cov
end
end
```

/* Now through above block first with 'foreach' loop we are iterating over all the instances of coverages (i.e. coverages defined in RefCoverage reference table) and for each instances we are doing below steps.
First checking whether 'if This.Vehicle.Coverage[CoverageCode=code] = null then' i.e. Now we want to access a Coverage object with 'CoverageCode' same as 'code' fetched through current iteration of 'RefCoverage' table, If Coverage with 'code' defined in 'RefCoverage' table is not available then we have to create the new Coverage Object instance and assign it to existing vehicle instance, which we are achieveing through last four lines of above code.
[Point3 is achieved here.]*/

```
foreach This.Vehicle.Coverage as cov do
if cov.CoverageCode <> null then
def limit
limit = lookup RefLimit to return RefLimit.Value as value having
RefLimit.CoverageCode = cov.CoverageCode
cov.Limit = limit.value
cov.Premium = limit.value / 100
end
end
```

/* Now through above block we are iterating through all the 'cov' instances which are now present in Coverage object, and for every instance performing following steps.
Frist checking whether current instance of coverage does not holds a null value for CoverageCode field, and if this condition is true then for every 'CoverageCode' creating a

limit reference which points to the corresponding limit 'Value'.
Further storing limit values in limit field of current instance of coverage, further performing arithmetic operation on limit value to calculate premium and store that in current instance of coverage object.
[Point4, Point5 are achieved here.]*/

```
def totalPremium as 0
foreach This.Vehicle.Coverage.Premium as prem do
if prem <> null then
totalPremium = totalPremium + prem
end
end
This.Premium = totalPremium
```

/*Now with above block, we are defining totalPremium as zero, and now iterating through every coverage instance and checking whether corresponding Premium field holds null value, which won't be a case in this scenario as we are already defining that in 'RefLimit' table and from here fetching and storing that in coverage instance. So with every iteration summation of premium will happen and we will get total premium for all coverages
[Point6 is achieved here.]*/

Above snowflake code helps to achieve below mentioned points.

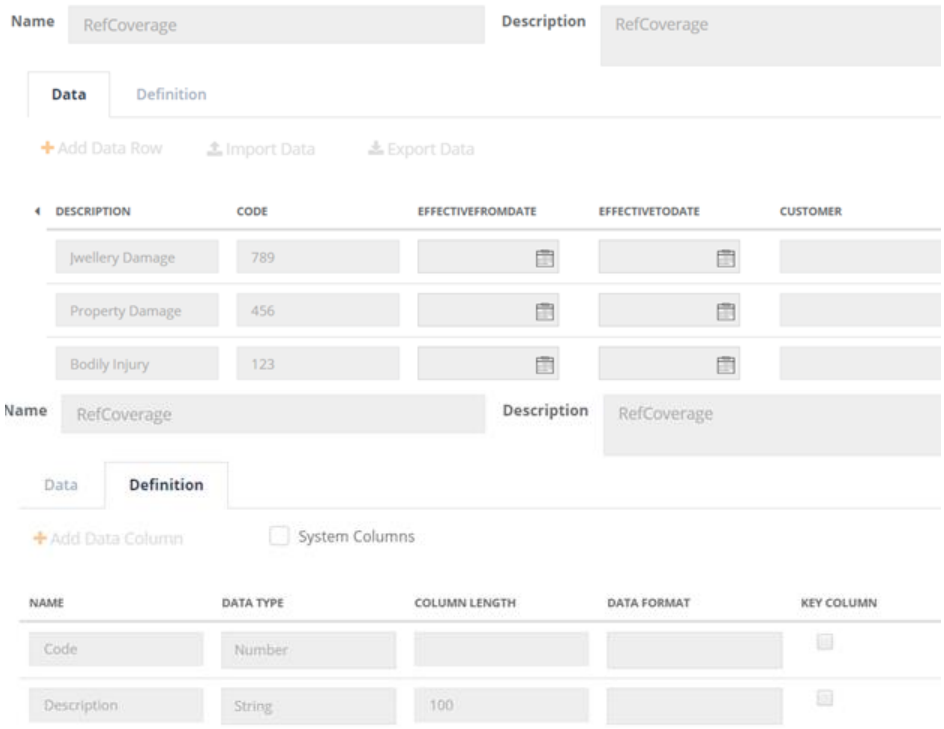
1. If Vehicle instance is not available, create a new vehicle instance and assign it to current Personal Auto object
2. Get the list of Coverage's available in Reference table [named RefCoverage]
3. Create the coverage instance under Vehicle object using number of coverage's available under RefCoverage reference table.
4. Get default limit value from Reference Table [named RefLimit] and Assign it to the vehicle coverage instance
5. While saving the limit value, calculate the premium using arithmetic operation and save it to coverage's premium field.
6. Find out the total coverage premium by looping over all the coverages available under vehicle and save it at Personal Auto's premium field.

To evaluate this entire scenario consider below mentioned objects, object models, pages and flow and runtime inputs. [App Name for reference: 'PolicyModelAccessApp' this belongs to QA1 environment and please don't update or delete anything]. If the page on which DSL is to be written is current page then, make sure that Navigate Next button of previous page has FQOP mapped as Vehicle object.

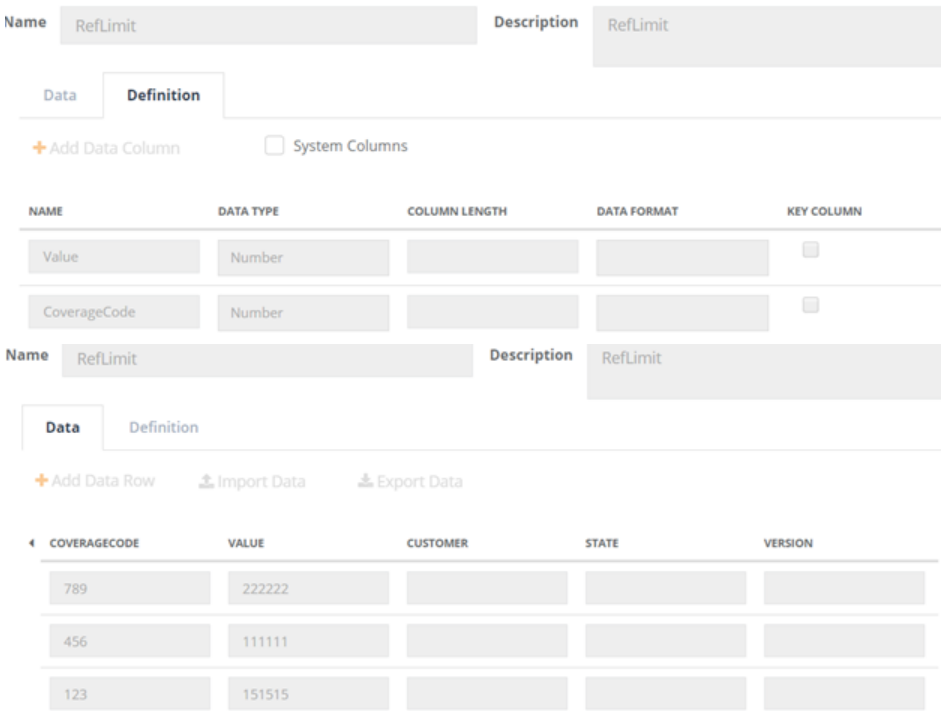
Create Reference tables as below

REFERENCE TABLE NAME	DESCRIPTION	CONTENT	SOURCE
RefCoverage	RefCoverage	PolicyCont	Custom
RefLimit	RefLimit	PolicyCont	Custom

RefCoverage Reference table

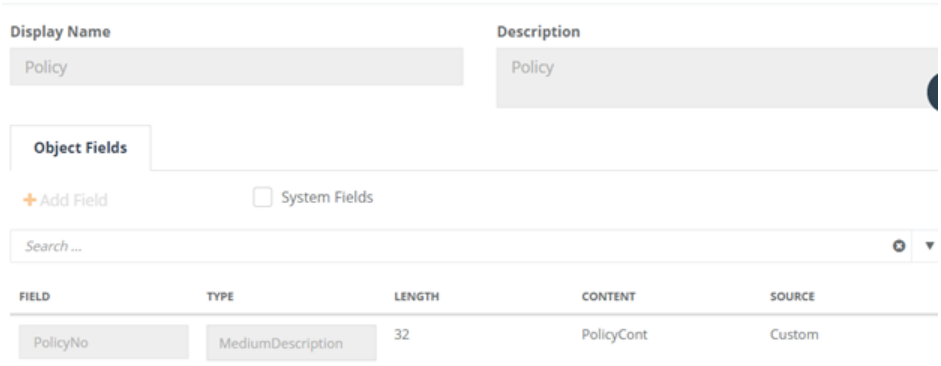


RefLimit Reference table

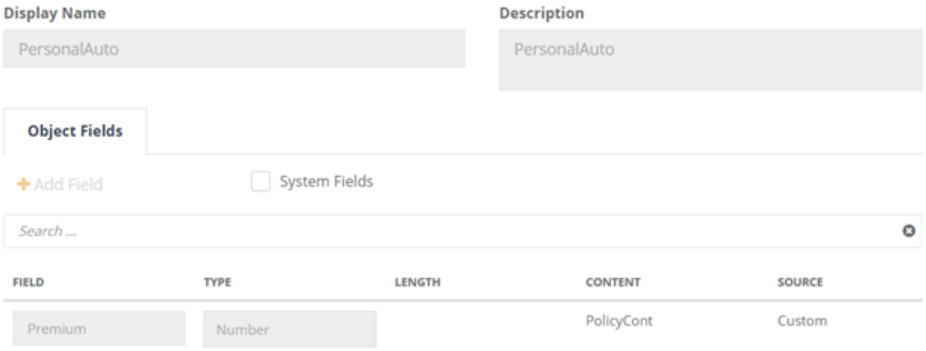


Create objects as below

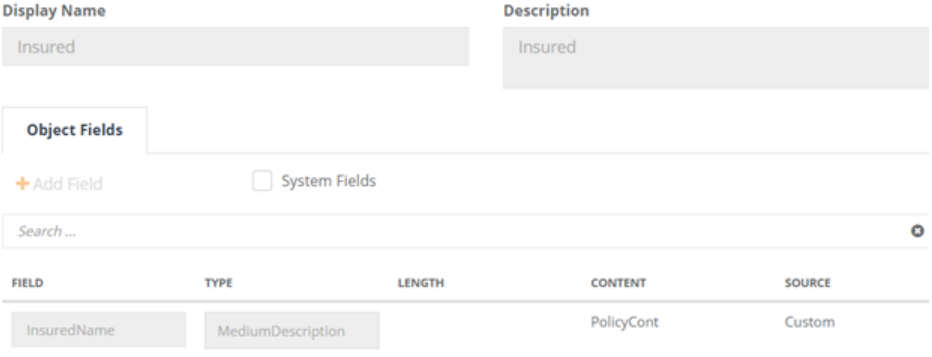
Policy object



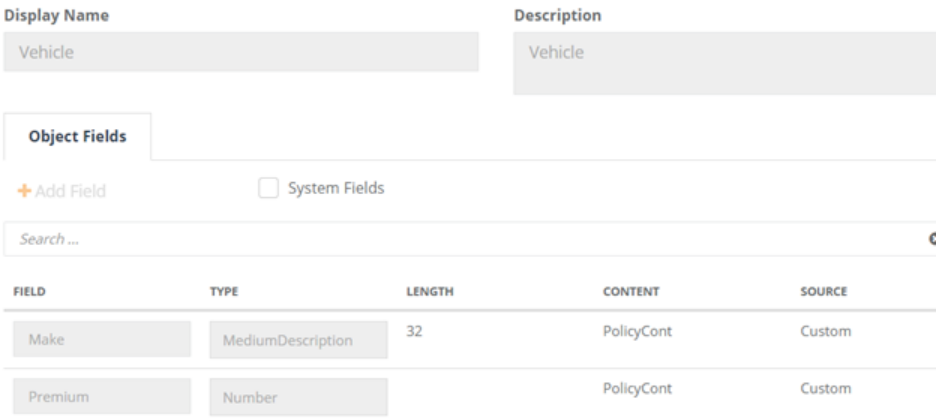
PersonalAuto object



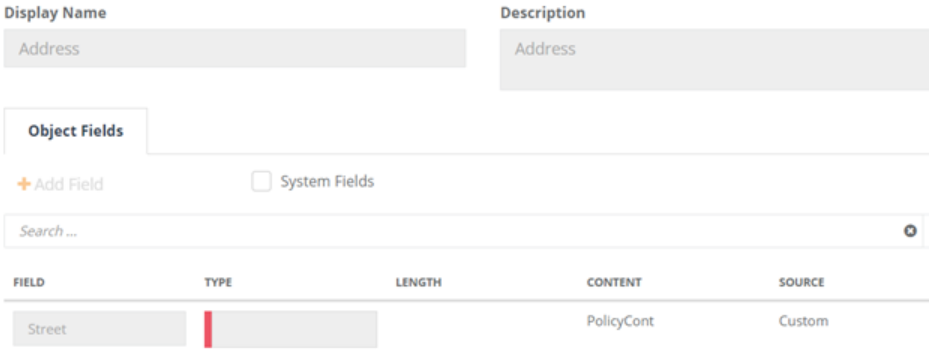
Insured object



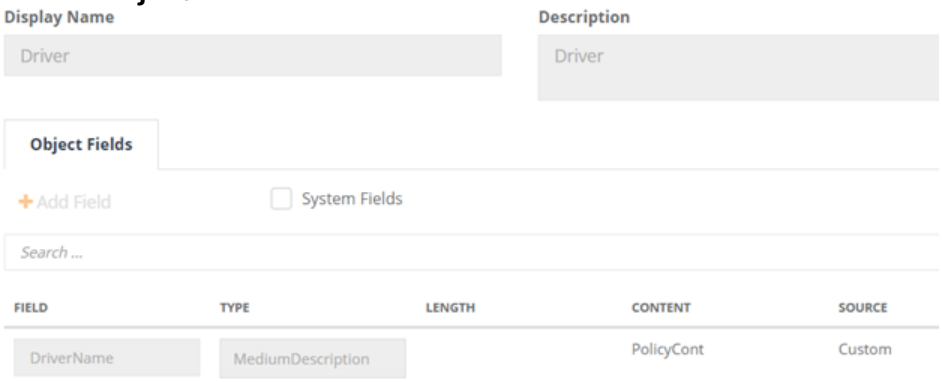
Vehicle object



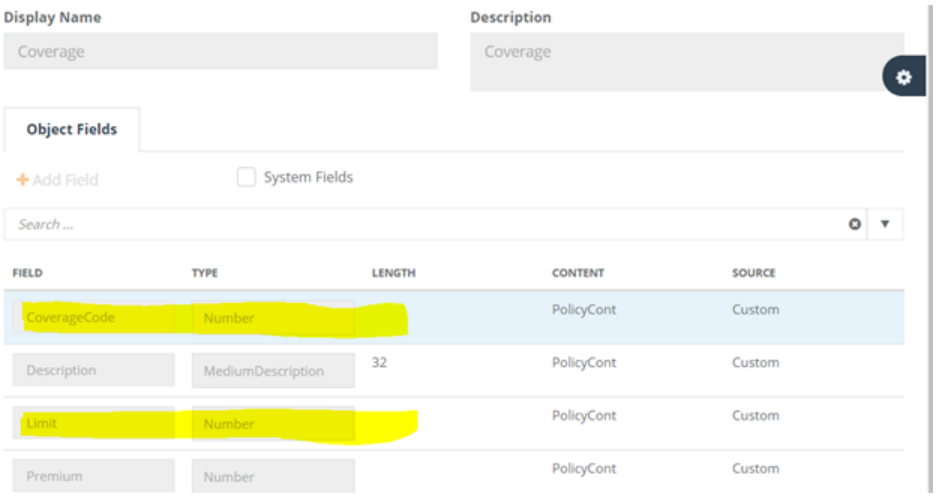
Address object



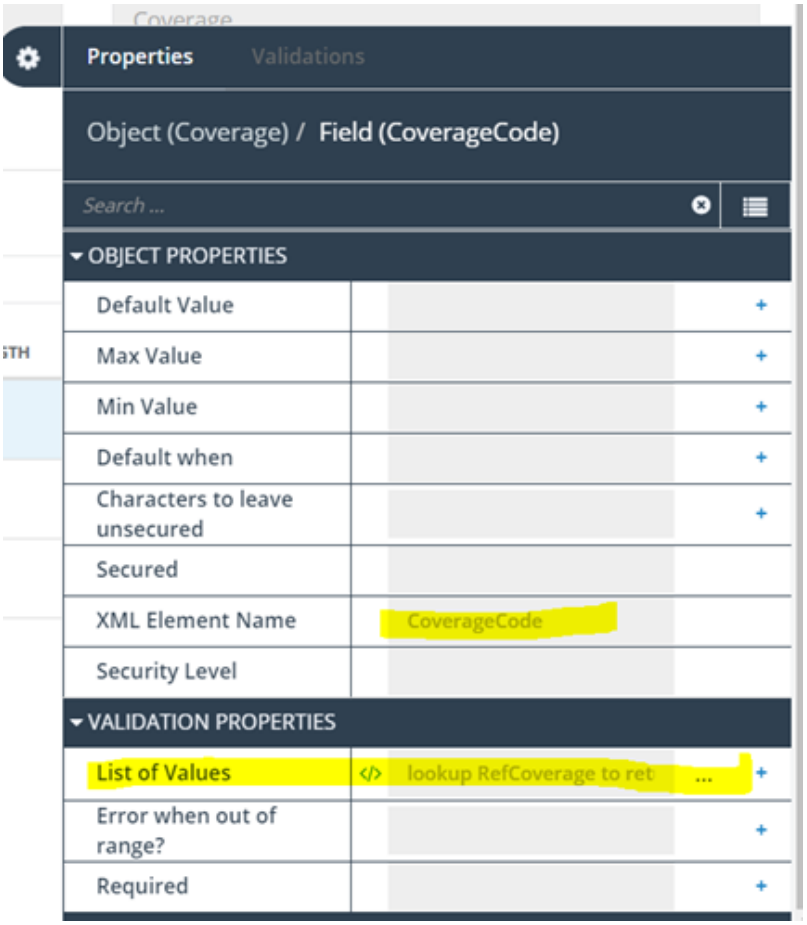
Driver object

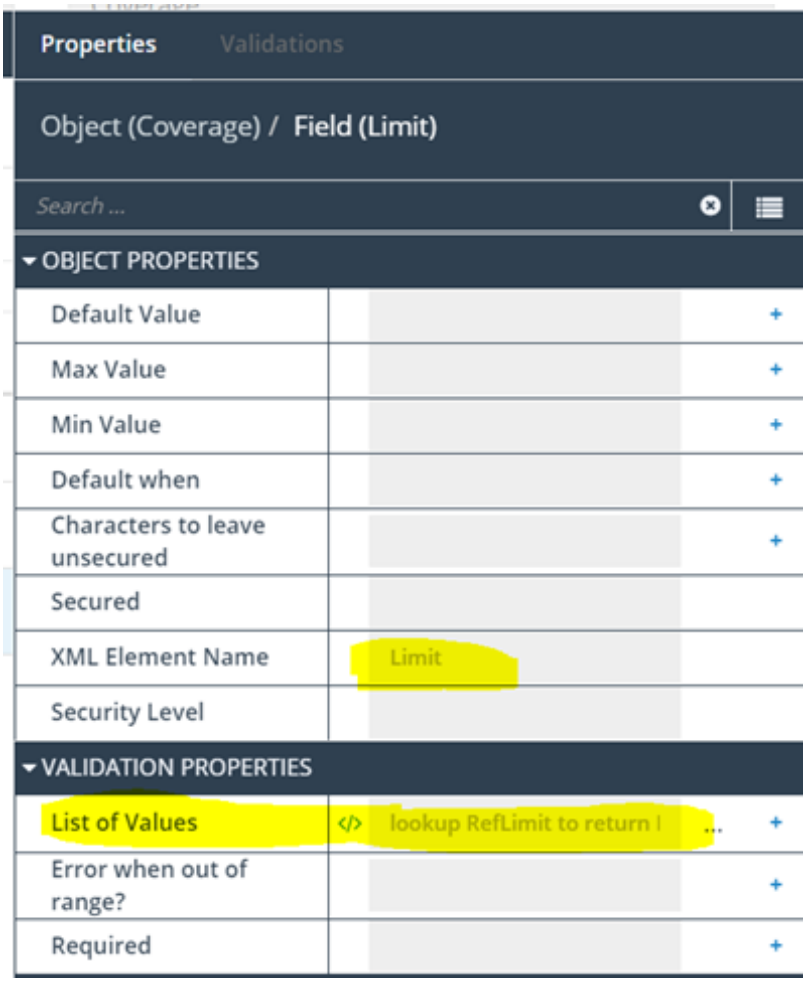


Coverage object



Lookup statement should be written on 'List of values' properties of these two object fields





Lookup queries are as below:

On coverage code field:

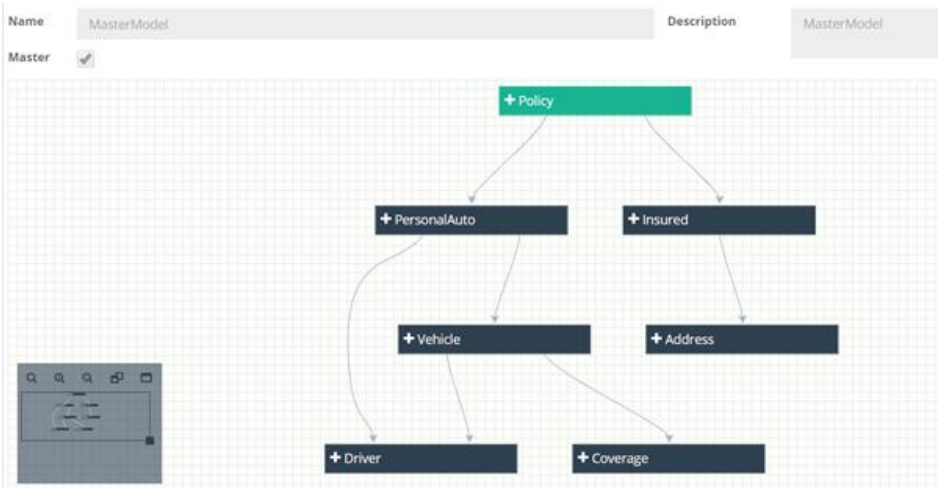
```
lookup RefCoverage to return RefCoverage.Code as Cid, RefCoverage.Description
as CDesc
```

On Limit field

```
lookup RefLimit to return RefLimit.CoverageCode as Lcode, RefLimit.Value as
Lid
```

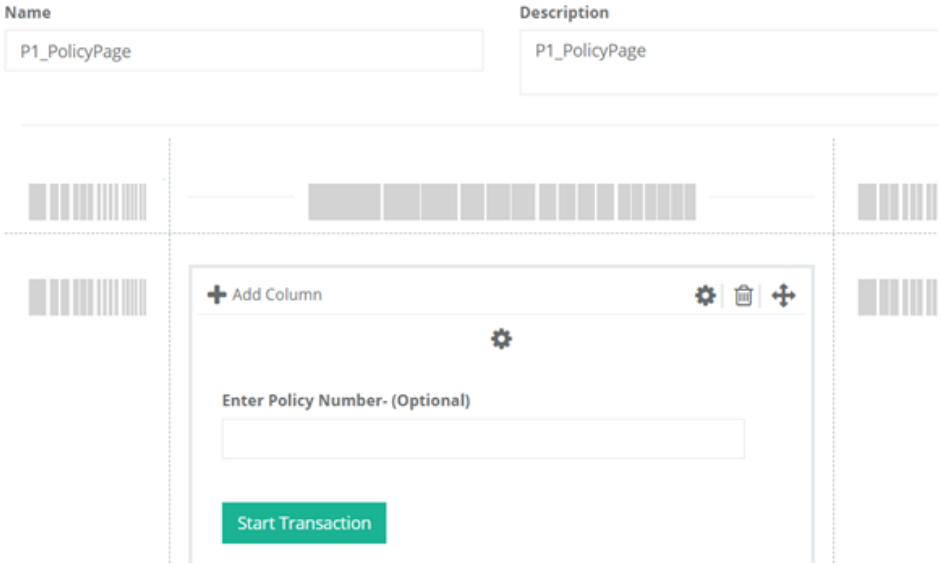
Create object model as below

MasterModel

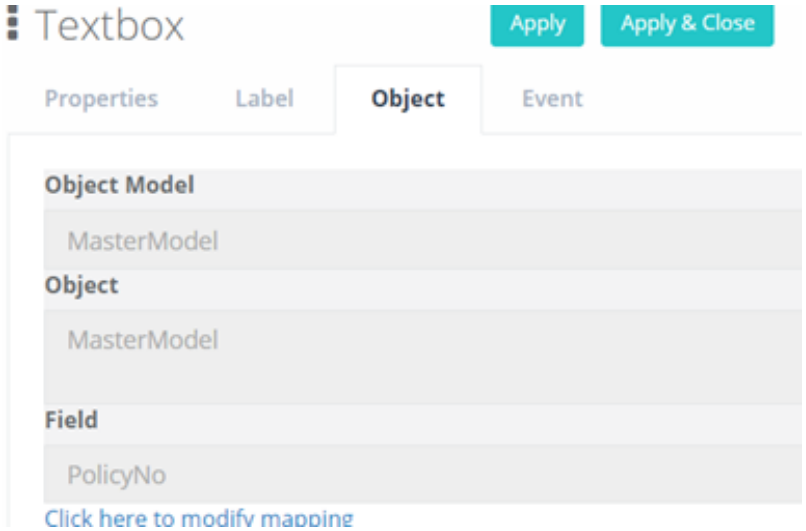


Create Pages as below

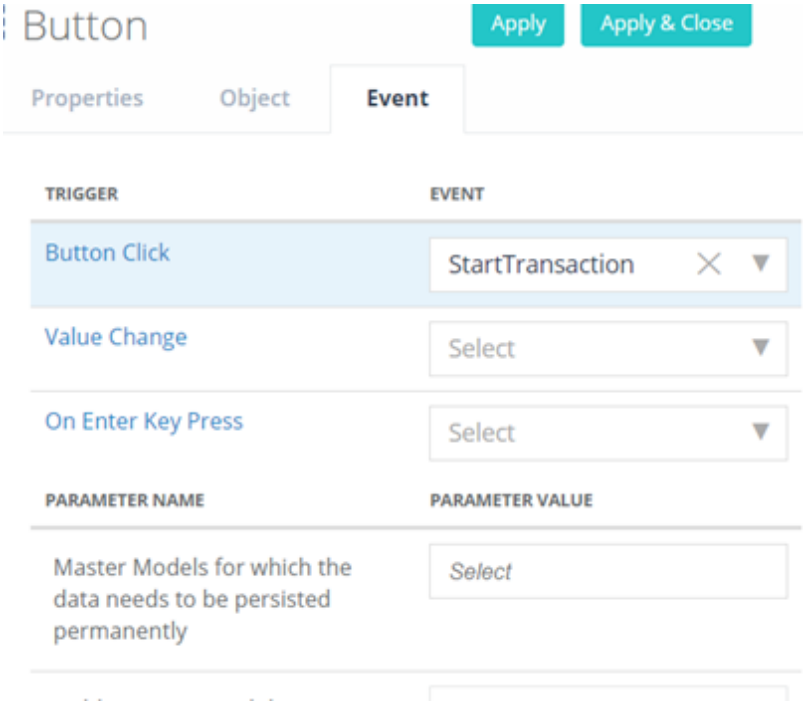
P1_PolicyPage



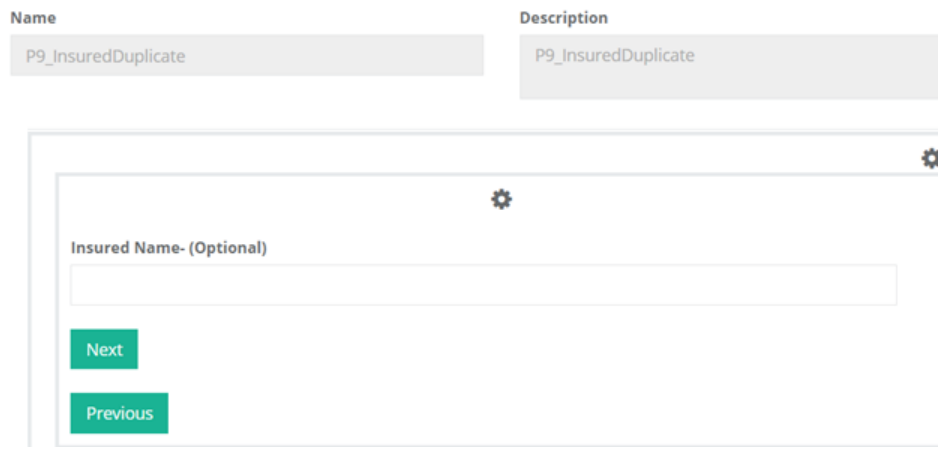
Enter Policy number field



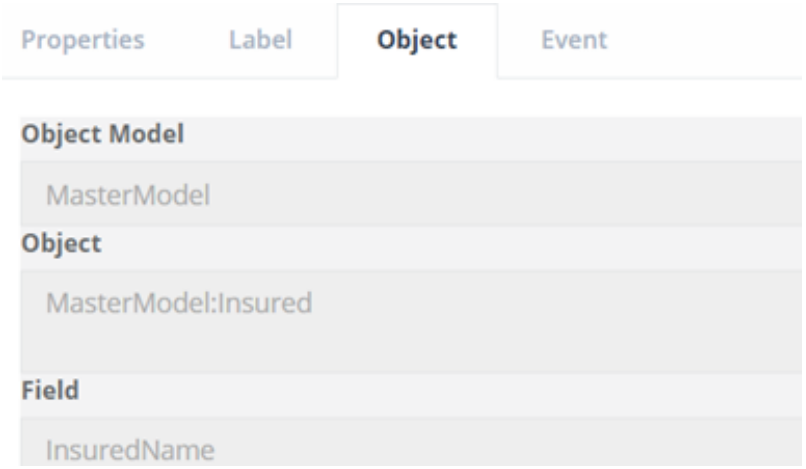
Start Transaction button



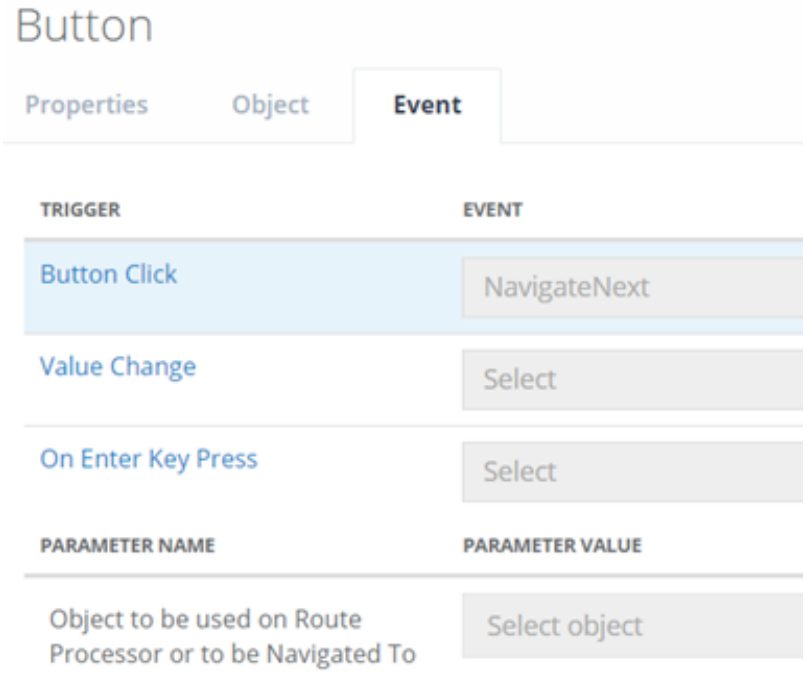
P9_InsuredDuplicate Page



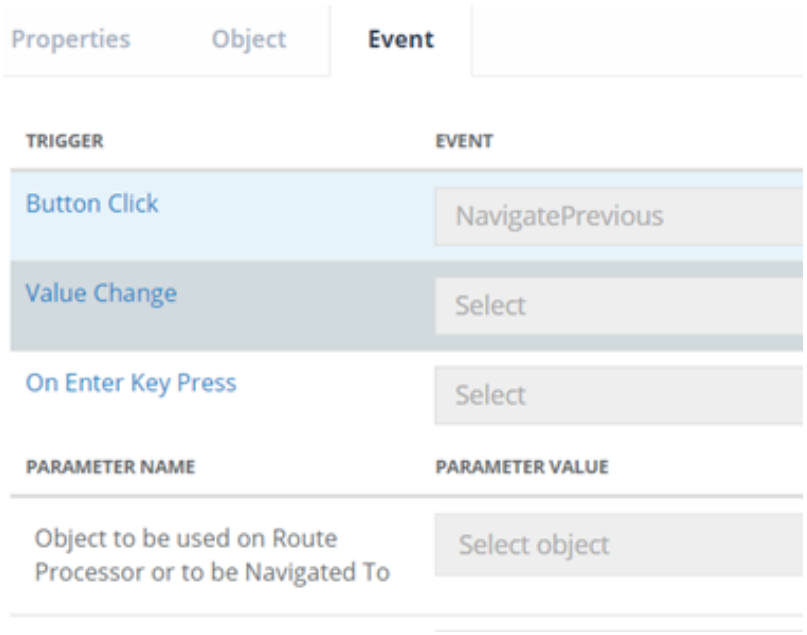
Insured Name field



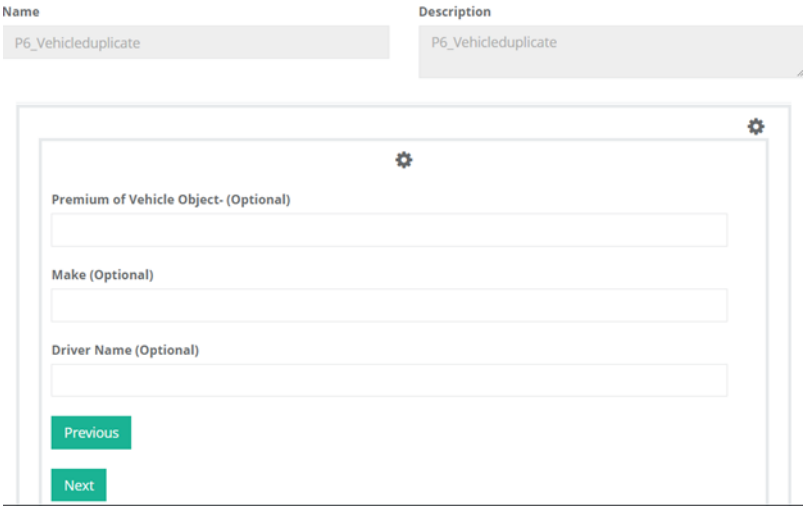
Next button



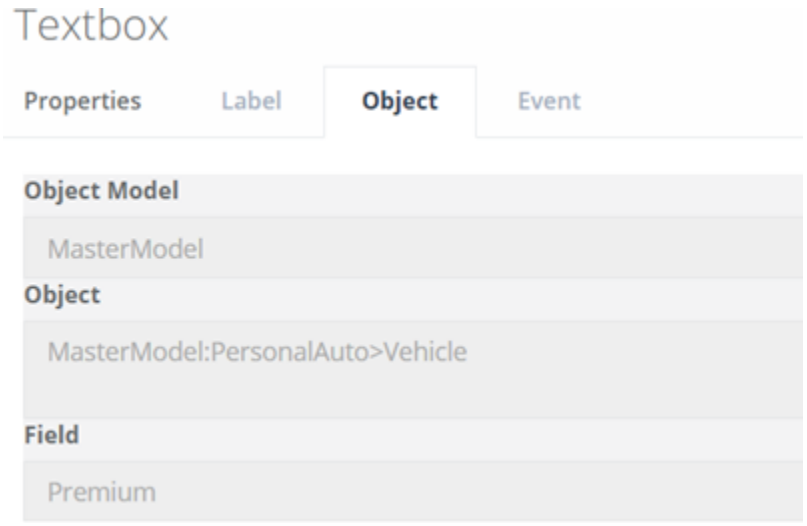
Previous Button



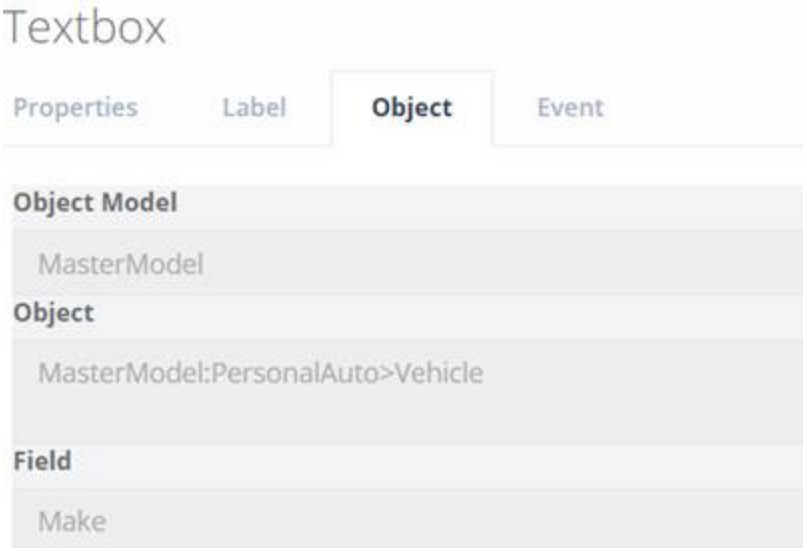
P6_Vehicleduplicate Page



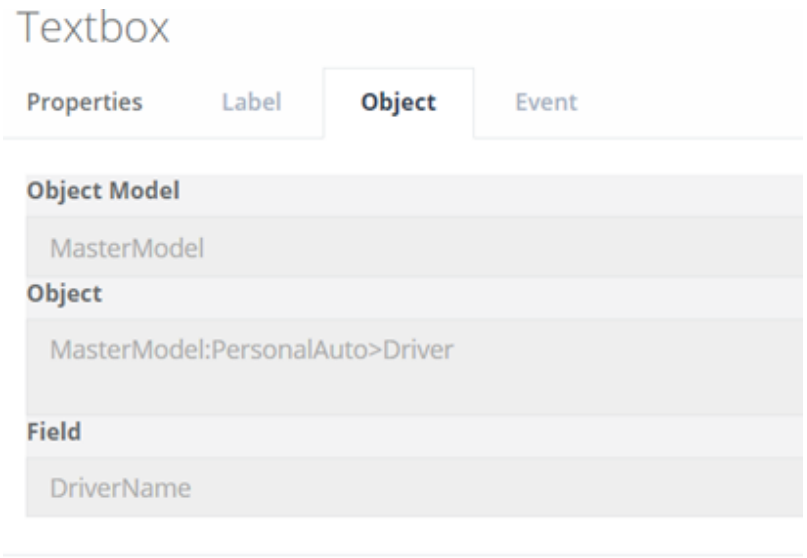
Premium Vehicle object field



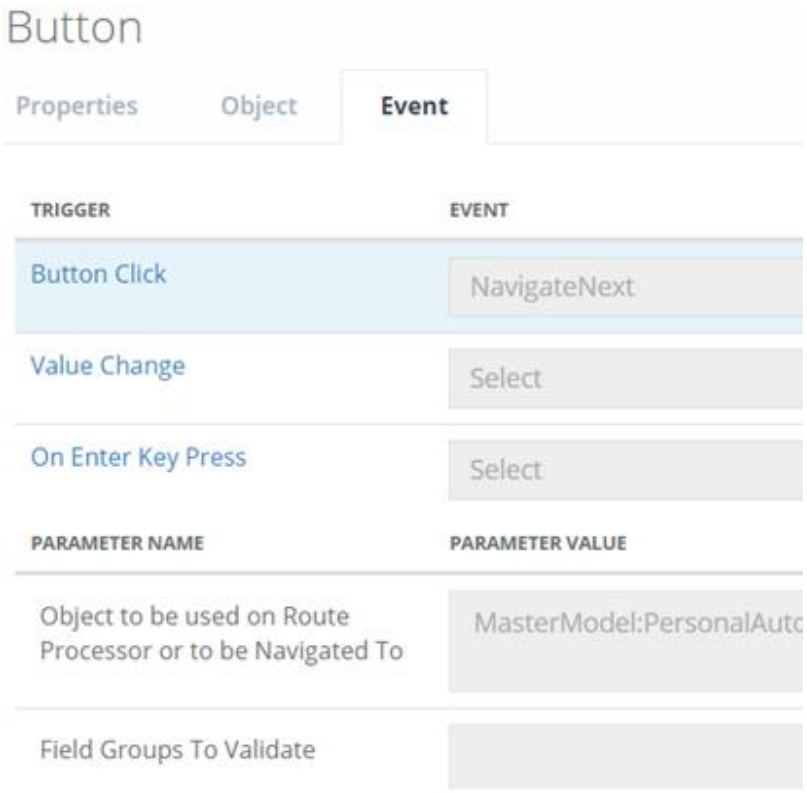
Make field



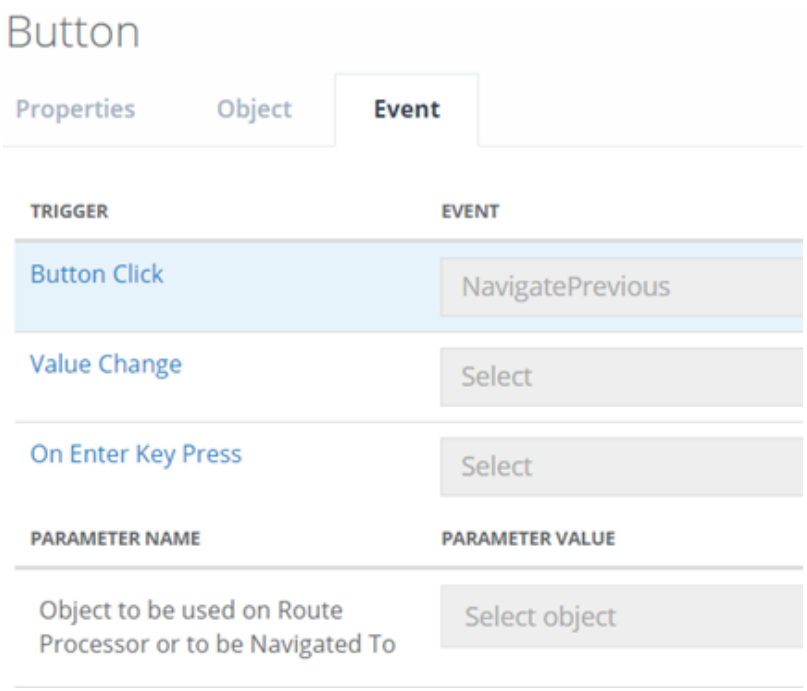
Driver name field



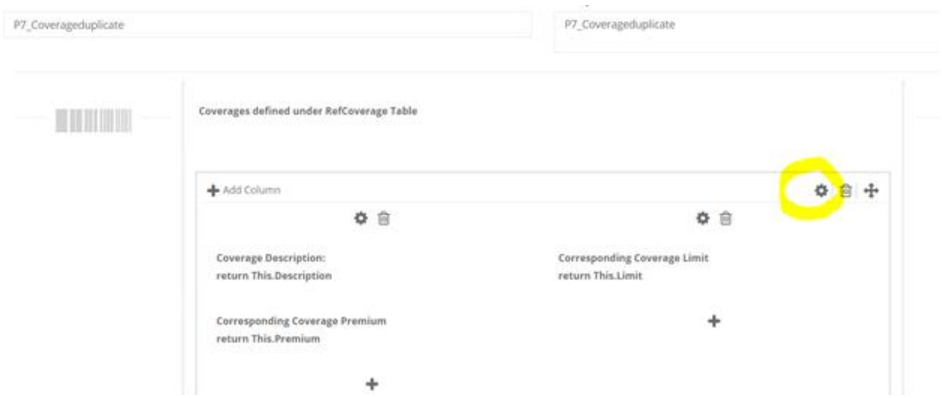
Next button



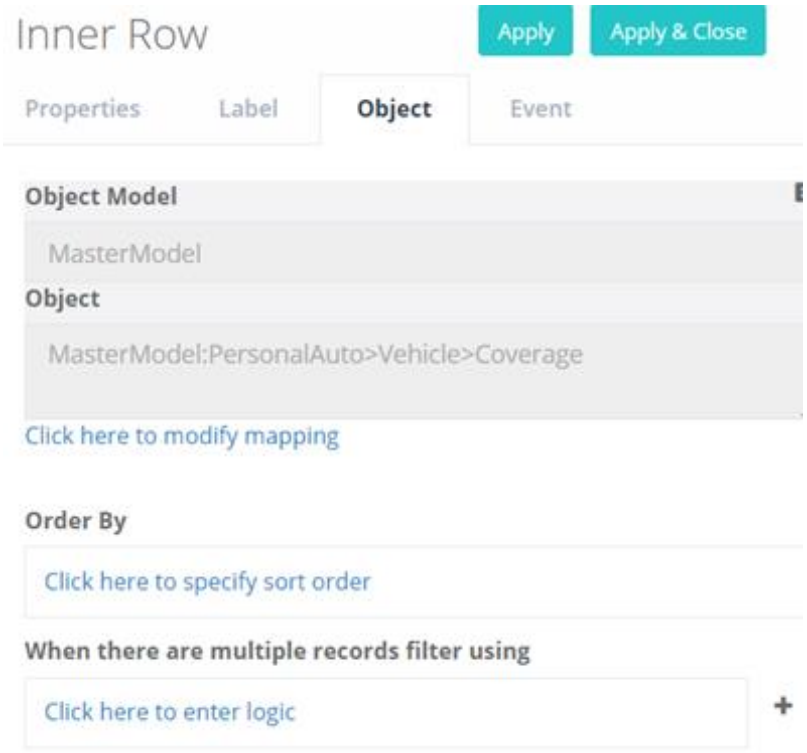
Previous button

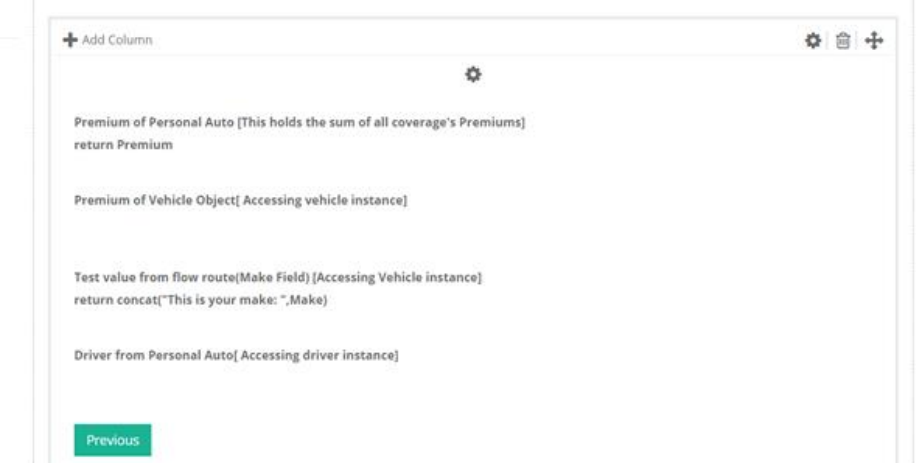


P7_Coverageduplicate Page

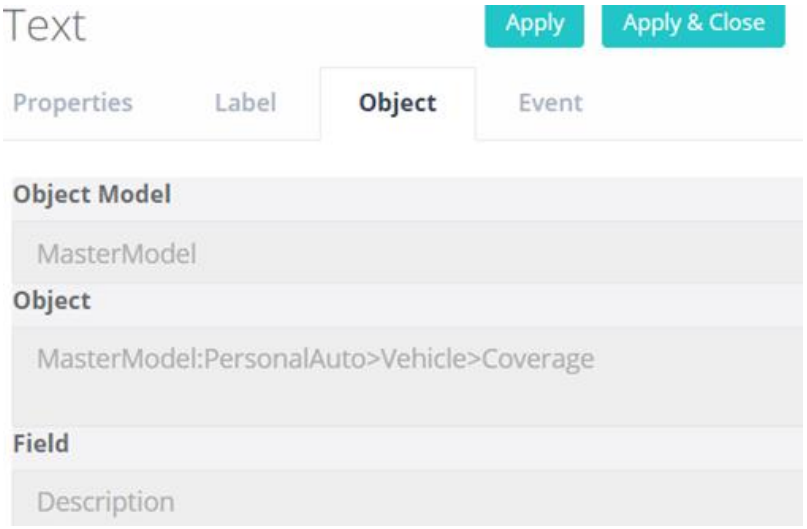


Update inner row as below

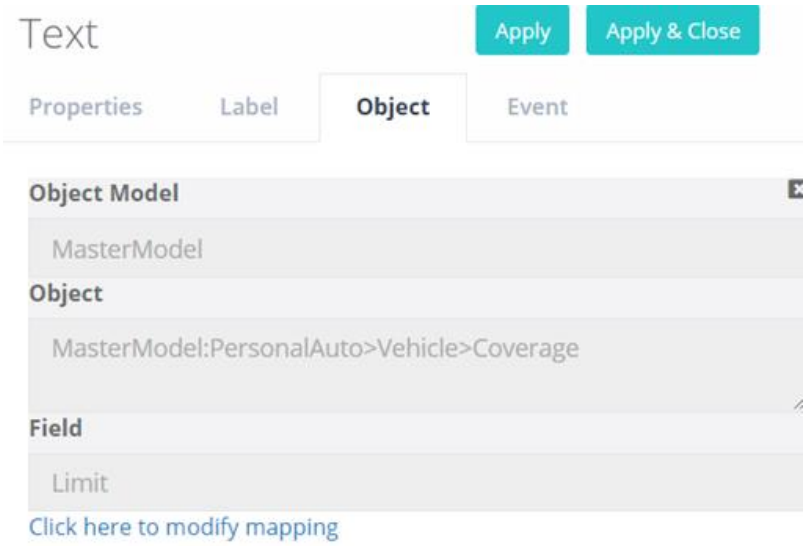




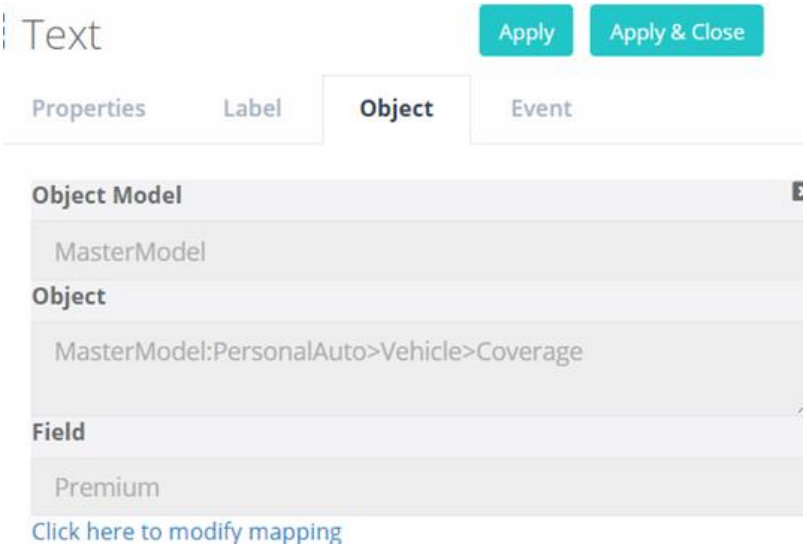
Coverage Description field



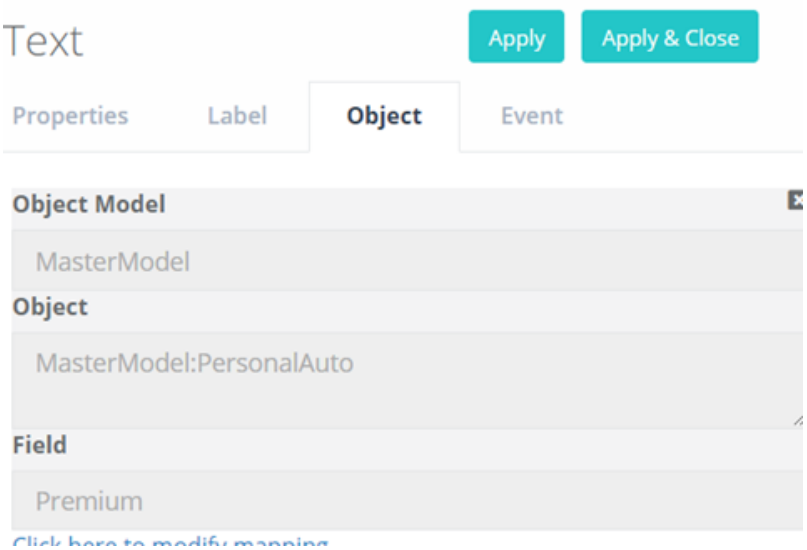
Corresponding Coverage Limit field



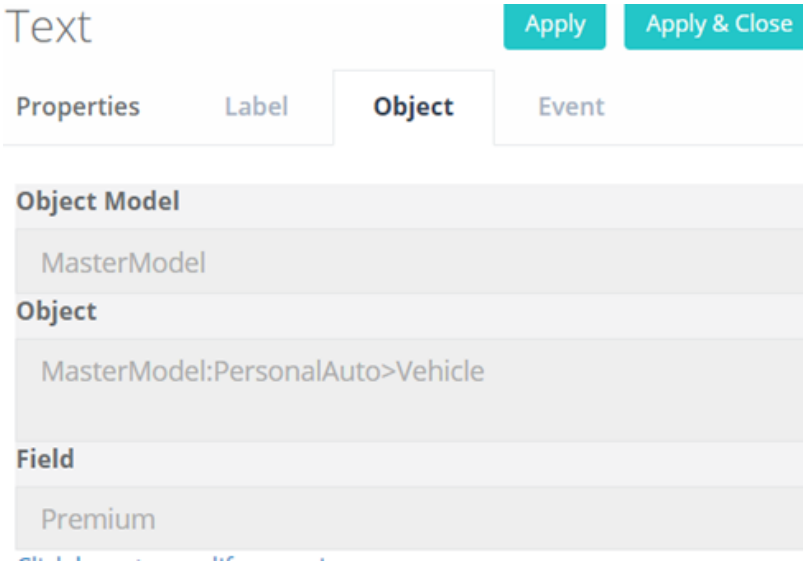
Corresponding Coverage Premium field



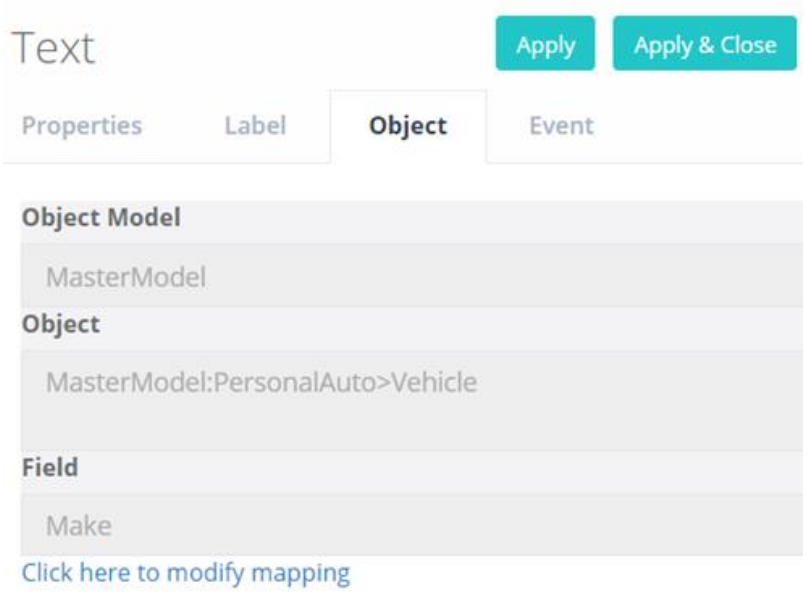
Premium of Personal Auto [This holds the sum of all coverage's Premiums] field



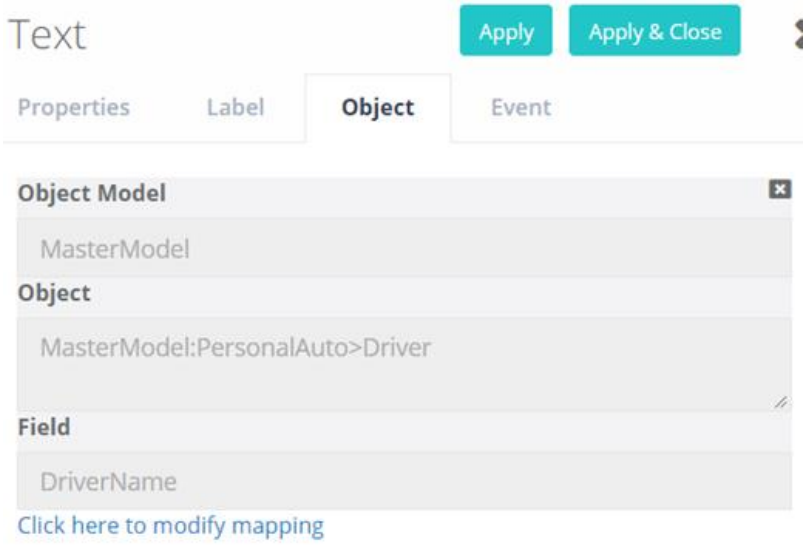
Premium of Vehicle Object [Accessing vehicle instance] field



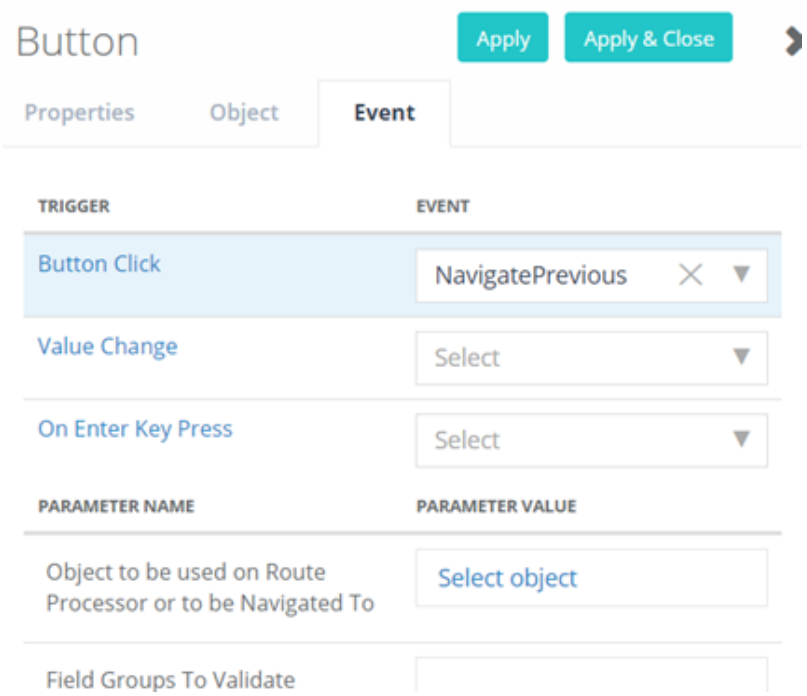
Test value from flow route (Make Field) [Accessing Vehicle instance] field



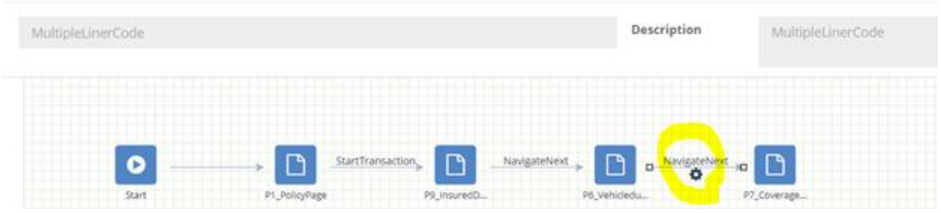
Driver from Personal Auto [Accessing driver instance] field



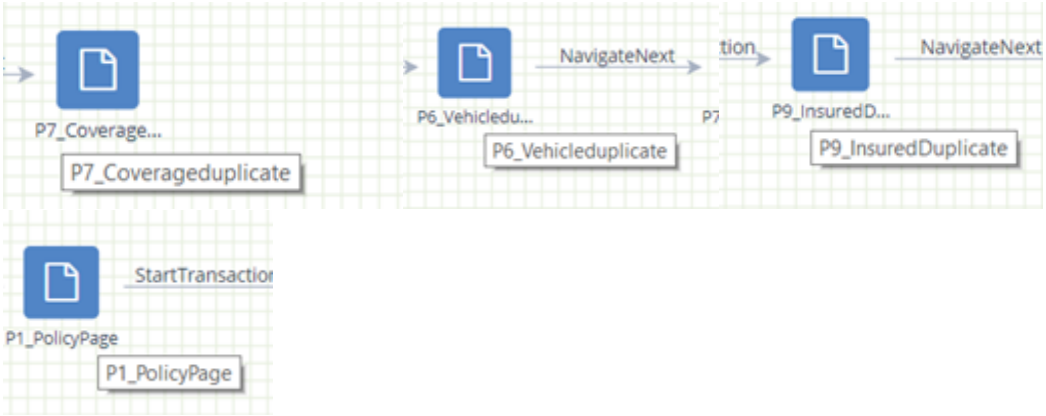
Previous button



Create below flow – write DSL on highlighted route.



Below screenshot show the full names of pages used in above flow



DSL

Supported Event | Expression

```
skip rule execution /*To improve performance*/

def x
x = lookup RefCoverage to return RefCoverage.Code as code , RefCoverage.Description as desc
if This.Vehicle = null then
def veh as new MasterModel:PersonalAuto.Vehicle
This.Vehicle = veh
end
foreach x as y do
def code as y.code
if This.Vehicle.Coverage[CoverageCode=code] = null then
def cov as new MasterModel:PersonalAuto.Vehicle.Coverage
cov.Description = y.desc
cov.CoverageCode = y.code
This.Vehicle.Coverage = cov
end
end

foreach This.Vehicle.Coverage as cov do
if cov.CoverageCode <> null then
def limit
limit = lookup RefLimit to return RefLimit.Value as value having RefLimit.CoverageCode = cov.CoverageCode
cov.Limit = limit.value
cov.Premium = limit.value / 100
end
```

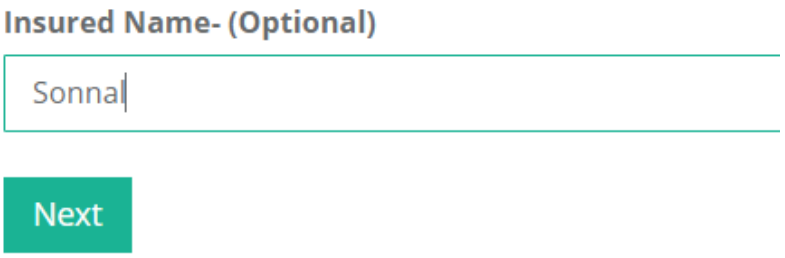
```
end

def totalPremium as 0

foreach This.Vehicle.Coverage.Premium as prem do
if prem <> null then
totalPremium = totalPremium + prem
end
end



resume rule execution /*to imporve performance*/

This.Premium = totalPremium
```

Verify at runtime

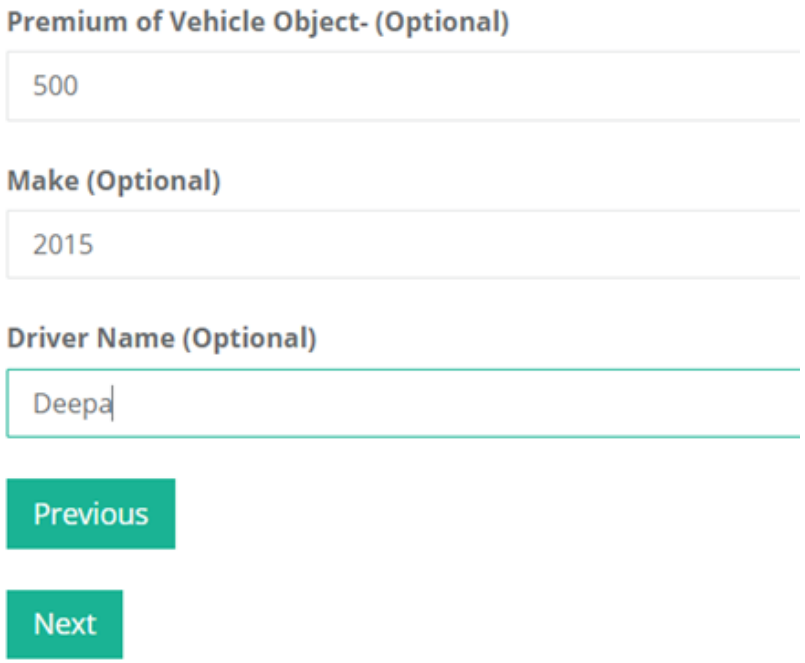
/public/sh7/multiplelinercode | PolicyModelAccessApp > MultipleLinerCode

On P1_PolicyPage, Enter policy no and Click on Start transaction button

On P9_InsuredDuplicate page, Enter insured name and click on next button



On P7_VehicleDuplicate Page, enter details as displayed below and click on next button



On P7_CoverageDuplicate Page – Verify outputs.

17 Functions

Functions can take zero or more variables or functions as input to execute a logic and return the result that can be assigned to another variable or used in any of the statements where a variable can be used.

Functions are great way to eliminate the duplication of common logic. This improves the productivity and make the code more readable and maintainable.

General syntax for using functions is:

variable = FunctionName (variable1, variable2, Another Function(...), ...) OR
return FunctionName (variable1, variable2, Another Function(...), ...)

Functions supported by Snowflake, can be written on below mentioned editors available at designer level

- In an App - On a page - any widget > Settings icon > anywhere the '+' icon is available > Advanced logic in when configuration
- In an App - On a page - any widget > Settings icon > anywhere the '+' icon is available > is expression in then condition
- In an App - On a page - any widget > Settings icon > anywhere the '<>' icon is available > Expression editor
- In an App - On a page > In inner row widget > Click here to enter logic' link > Expression editor
- In an App - On a flow route > Settings icon > Event/Expression editor
- In an App - In a report > Expression editor

17.1 String Functions

Function	Description
trim left()	Return a string containing a copy of a specified String with no leading spaces
trim right()	Return a string containing a copy of a specified String with no trailing spaces
trim()	Return a string containing a copy of a specified String with no leading and trailing spaces
upper()	Return a string or character containing the specified string converted to uppercase
lower()	Return a string or character containing the specified string converted to lowercase
title()	Return a string or character containing the specified string converted to title case

Function	Description
left pad()	Return a string containing a copy of a specified String padded from left by specified pad string
right pad()	Return a string containing a copy of a specified String padded from right by specified pad string
length()	Return an integer that contains the number of characters in the String
replace()	Replace single instance of old string with new string
replace all()	Replace all instances of old string with new string
replace first()	Replace first instance of old string with new string
replace last()	Replace last instance of old string with new string
left()	Return a string containing a specified number of characters from left side of specified string
mid()	Return a string containing a specified number of characters from specified string
right()	Return a string containing a specified number of characters from right side of specified string
compare()	Returns values (-1, 0 ,1) based on the result of a string comparison. Return 0 if both string are equals, -1 if string 1 sorts ahead of string 2, 1 if string 2 sorts ahead of string 1.
contains()	Return true if string contains the specified sequence of character value.
not contains()	Return true if string does not contains the specified sequence of character value
concat ()	Join two or more strings
sentence()	Returns a string containing cases(upper/lower) like a sentence, from the given string

Few of above mentioned functions can be used with the keyword 'of' instead of function brackets '()'

Function	Description
left of ___ starting at __	same as above
right of___ starting at __	same as above
min of ___	same as above
max of ___	same as above

1. trim left
return trim left(" Hello ")
Result of above execution block will be "Hello "

2. trim right
return trim right(" Hello ")

Result of above execution block will be " Hello"

3. trim
return trim (" Hello ")
Result of above execution block will be "Hello"

4. upper
return upper("Hello")
Result of above execution block will be "HELLO"

5. lower
return lower ("HELLO")
Result of above execution block will be "hello"

6. title
return title("this is a Test for Sentence.lets see how it works.")
Result of above execution block will be "This Is A Test For Sentence.Lets See How It Works."

7. left pad
return left pad("Sample",5, "000")
Result of above execution block will be "Sample"

8. right pad
return right pad("Sample",15, "000")
Result of above execution block will be "Sample000000000"

9. length
return length ("Hello")
Result of above execution block will be "5

10. replace
return replace ("SampleTextSampleTeSamplext", "Text", "")
Result of above execution block will be "SampleSampleTeSamplext"

11. replace all
return replace all ("SampleTextSampleTeSamplext","[a-z]","-")
Result of above execution block will be "S----T-S----T-S-------"

12. replace first
return replace first("SampleTextSampleTextSampleText", " Text", "")
Result of above execution block will be "SampleSampleTextSampleText"

13. replace last
return replace last("SampleTextSampleTextSampleText", " Text", "")

Result of above execution block will be "SampleTextSampleTextSample"

14. left
return left("HelloWorld", 6)
Result of above execution block will be "HelloW"

15. mid
return mid("HelloWorld", 3,3)
Result of above execution block will be "loW"

16. right
return right("HelloWorld", 3)
Result of above execution block will be "rld"

17. compare
return compare("Hello","hello")
Result of above execution block will be "-1"

18. compare
return compare("Hello", "Hello")
Result of above execution block will be "0"

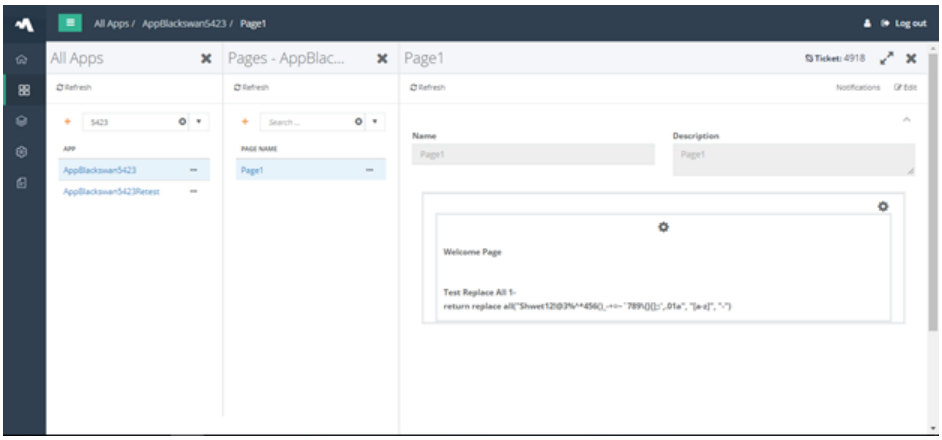
19. contains
if contains ("Digital", "i") then
return true
end
Result of above execution block will be true

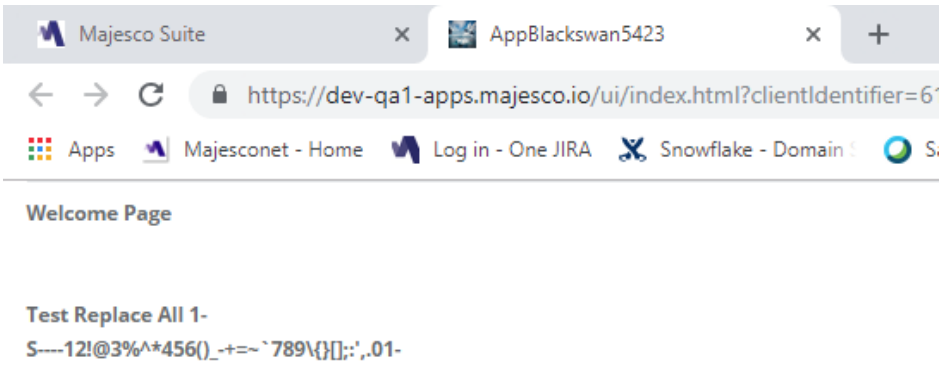
20. not contains
if not contains ("Digital", "digital") then
return true
end
Result of above execution block will be true

21. concat
def insName as Insured.Name
return concat ("Hello !", insName)
Result of above execution block will be=> Hello ! DigitalUser

22. sentence
return sentence ("hEllo.i Have a solution.", insName)
Result of above execution block will be=> Hello.I have a solution.

Designer level illustration of example replace all() function is below:-





17.2 Number Functions

Function	Description
ceiling()	Return the nearest number value greater than or equals to specified number value
floor()	Return the nearest number value less than or equals to specified number value
absolute()	Return the absolute value of number
round()	Return the value of number rounded to a specific number of digit
min()	get the min value from given values
max()	get the max value from given values
convert to number()	Converts string to number data type

17.2.1 Syntax & examples

1. ceiling
return ceiling (100.25)

Result of above execution block will be 101

2. floor
return floor (100.25)
Result of above execution block will be 100

3. absolute
return absolute (100.25)
Result of above execution block will be 100.25

4. round
return round (275.00 , 0)
#Result of the above execution will be 275
return round (275, 4)
#Result of the above execution will be 275.0000

5. min
get the min value from given values
def x as 1, y as 2, z as 3
return min (x , y , z)
Result of above execution block will be 1

6. max
get the max value from given values
def x as 1, y as 2, z as 3
return max (x , y , z)
Result of above execution block will be 3

7. convert to number
convert the string data type to number data type
def x as "123"
return convert to number (x)
Result of above execution block will be numeric 123

Few of above mentioned functions can be used with the keyword 'of' instead of function brackets '()'

Function	Description
min of ___	same as above
max of ___	same as above

17.2.2 Syntax & examples:-

1. min

get the min value from given values
def x as 1, y as 2, z as 3
return min of x , y , z
Result of above execution block will be 1

2. max
get the max value from given values
def x as 1, y as 2, z as 3
return max x , y , z
Result of above execution block will be 3

17.3 Date Functions

Function	Description
current date()	Return current system date
date()	Return a date instance for a specified date string, provided date string is specified as "date-month-year" format (dd/MM/yyyy)
date with format()	Return a date instance for a specified date string and provided date format string
format date()	Returns text value in a mentioned format from a date instance
date difference()	Return a number value for number of days, months or year between two days.
change date days()	Return a new date instance by adding number of days
change date months()	Return a new date instance by adding number of months
change date years()	Return a new date instance by adding number of years
date with timezone()	Returns a date instance for specified date, date format and time zone
change date timezone()	Returns a date instance for specified date, from time zone and to time zone

1. current date
return current date
Result of above execution block will be "2018-03-29"

2. date – with date format specified along with date value
return date ("03/28/2018", "dd/MM/yyyy")
Result of above execution block will be "2018-03-29"

3. date – without date format, in this case date string value is expected in dd-MMM-yyyy format
return date ("29-Mar-2018")

Result of above execution block will be "29-Mar-2018"

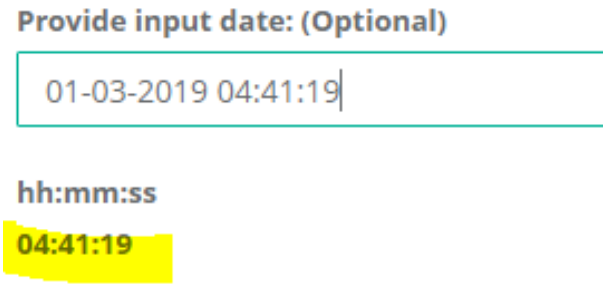
4 date difference – get the difference between two date in days, weeks, months or years
return date difference (date("02-May-2017") , date("02-May-2018"), days)
Result of above execution block will be "365"
return date difference (date("02-May-2017") , date("02-May-2018"), years)
Result of above execution block will be "1"
return date difference (date("02-May-2017") , date("02-May-2018"), months)
Result of above execution block will be "12"
return date difference (date("02-May-2017") , date("02-May-2018"), weeks)
Result of above execution block will be "52"
return date difference (date("02-May-2018") , date("02-May-2018"), years)
Result of above execution block will be "0"

5. change date years – increase the date by year
return change date years (date("02-Mar-2018"), 1)
Result of above execution block will be "02-Mar-2019"

6. change date months – increate the date by months
return change date months (date("02-Mar-2018"), 3)
Result of above execution block will be "02-Jun-2018"

7. change date days – increase the date by days
return change date days (date("02-May-2018"), 30)
Result of above execution block will be "01-Jun-2018"

8.format date - takes two input parameters
date(runtime input) and date/time form and returns output in text format
Syntax : return format date (input date, "format")
return format date (This.EffectiveFromDate, "hh:mm:ss") - this returns current time.



Examples for different formats supported as of now for fetching date/ time
format date(Policy:EffectiveFromDate, "hh:mm:ss")
format date(Policy:EffectiveFromDate, "DD/MM/YYYY")
format date(Policy:EffectiveFromDate, "DD/MM/YY")
format date(Policy:EffectiveFromDate, "MM/DD/YY")

format date(Policy:EffectiveFromDate, "MMM/DD/YY")
format date(Policy:EffectiveFromDate, "Month DD,YYYY")
format date(Policy:EffectiveFromDate, "Day,DD Mon,YYYY")
format date(Policy:EffectiveFromDate, "Day,DD-mon-YY")

**Discussion in progress for static date input and other time formats like HH, HH12, HH24, MM, MM:SS, HH:MM.

9. date with timezone – It will accept date in string format in first parameter, translate it with format specified in second parameter and consider its timezone as mentioned in third parameter.

Example 1 : return date with timezone ("20/05/1994","dd/MM/yyyy", "Asia/Calcutta")
Result of above execution block will be "1994-05-20"

Example 2 : return date with timezone ("20/05/1994","dd/MM/yyyy", "APP_TIMEZONE")
APP_TIMEZONE can be specified if application timezone is required.

10. change date timezone – It will accept date in first parameter, consider its timezone as mentioned in second parameter and will convert specified date to the time zone mentioned in third parameter.

Example 3 : return change date timezone(current date,"UTC","APP_TIMEZONE")
current date will be considered as UTC and will be converted in applications time zone.

17.4 Object Functions

reference for	Create a reference to instance, that can be mapped to another instance. Mostly this function will be used in case of reference relation, where One master model is connected to another master model.

```
def x as new UserBillPayRegistration:Root
def userName as This.UserName
def user
user = from UserAccount
       select UserAccount:InstanceKey as instanceKey
       where UserAccount:UserName = userName
def iKey = reference for ( UserAccount:Root , user.instanceKey )
x.UserAccount_UserAccount = iKey
```

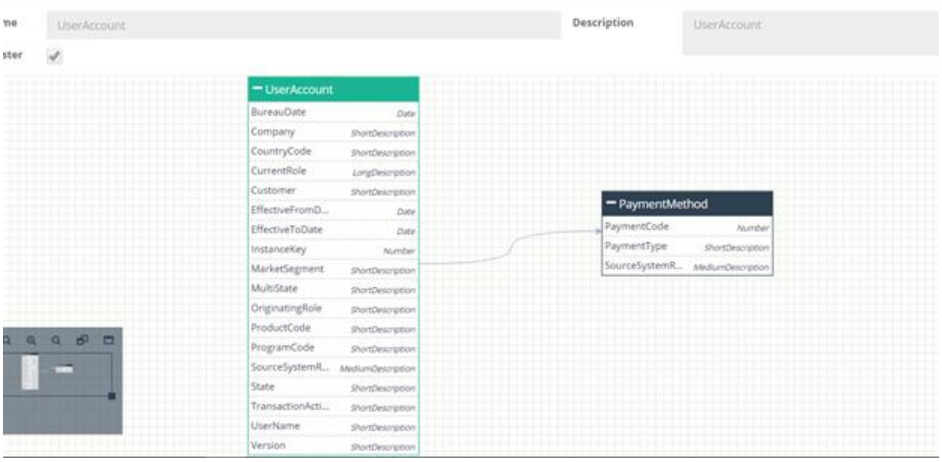
above block statement will result in getting a UserAccount from database where UserName is same as logged in user. Later create a reference instance using 'reference for' function and use this referenced instance to map to the UserBillPayRegistration.

To evaluate above snowflake code, follow below mentioned steps.

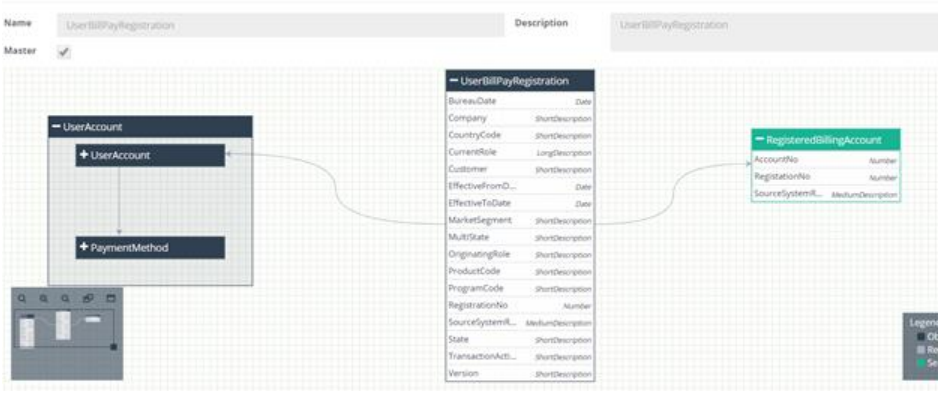
Create objectmodels as displayed in below screenshot

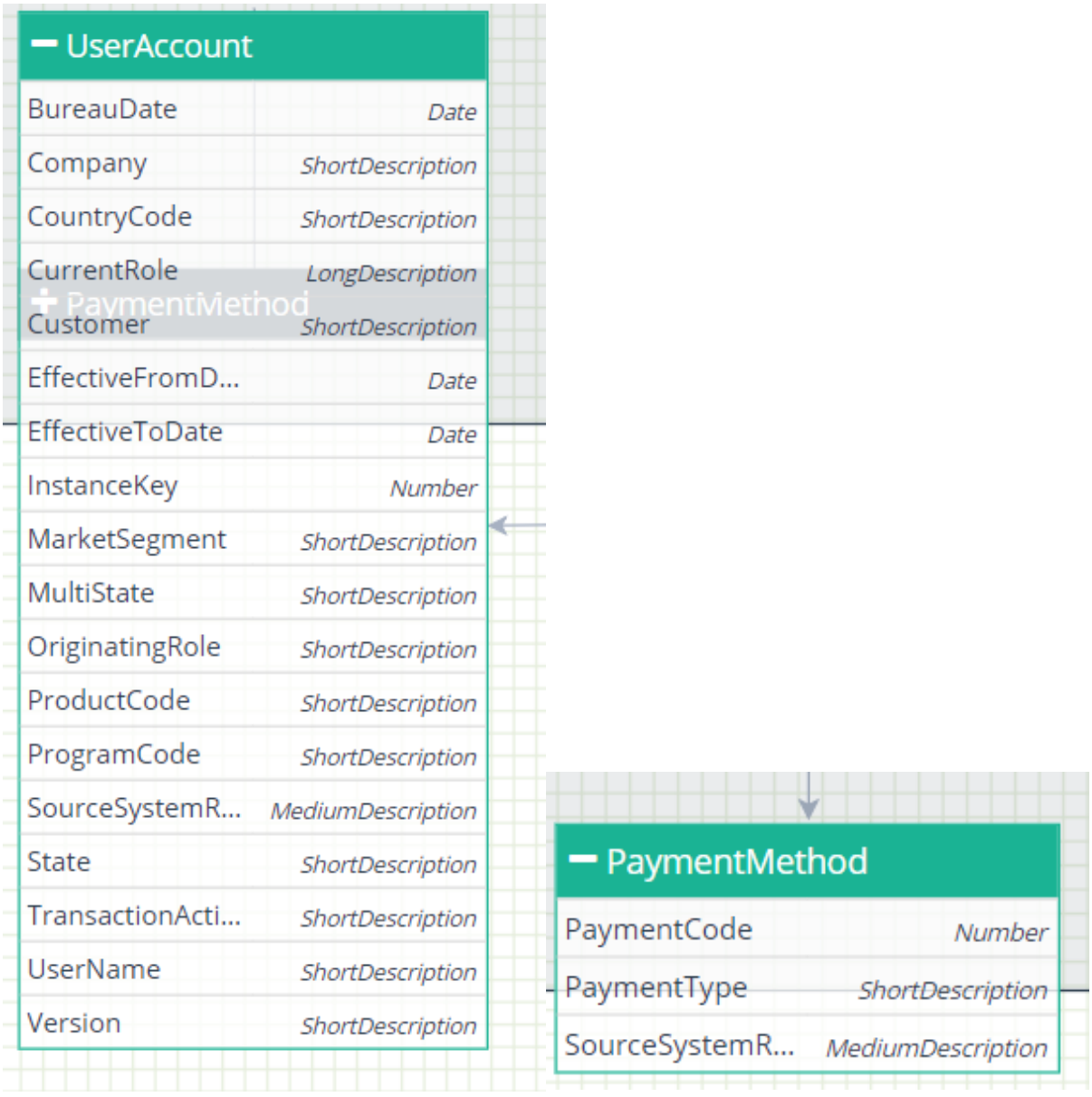
RegisterUserReq

UserAccount



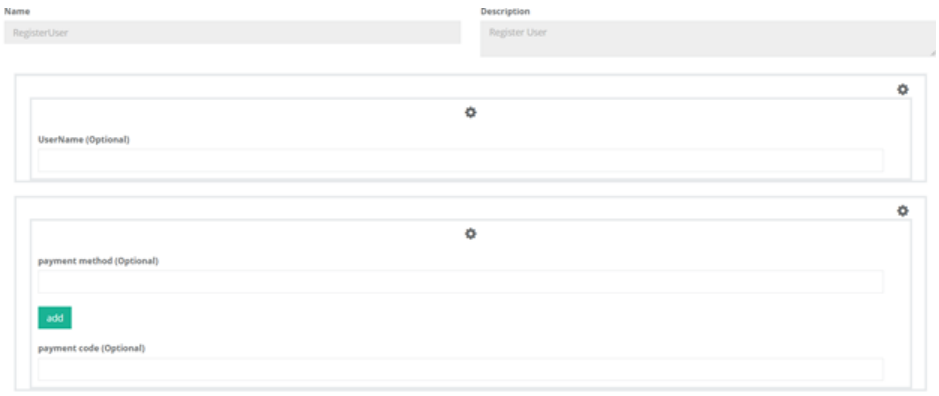
UserBillPayRegistration



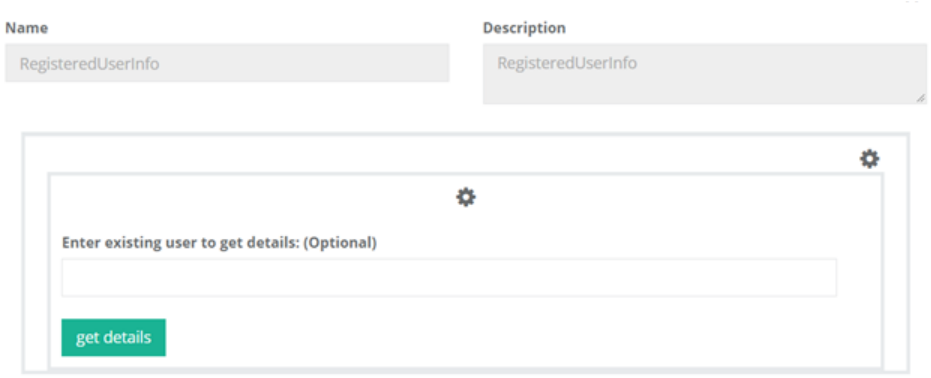


Create Pages as below

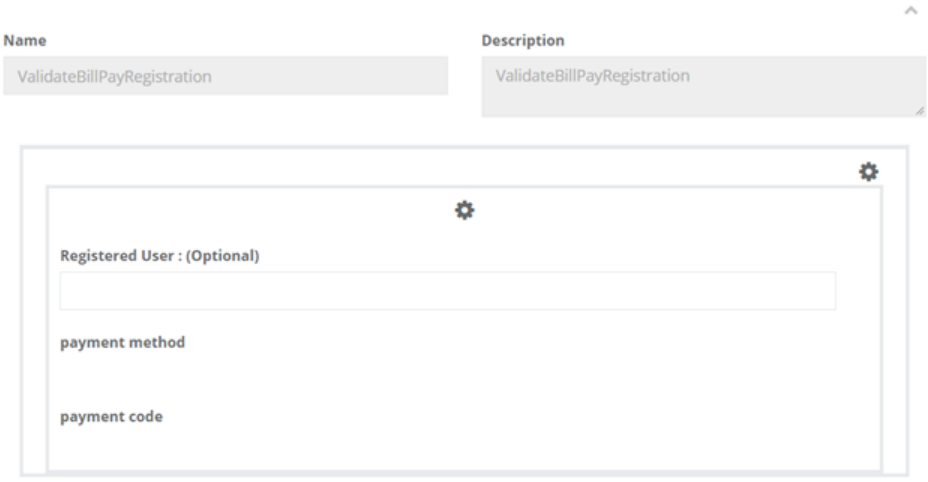
RegisterUser Page



RegisteredUserInfo Page

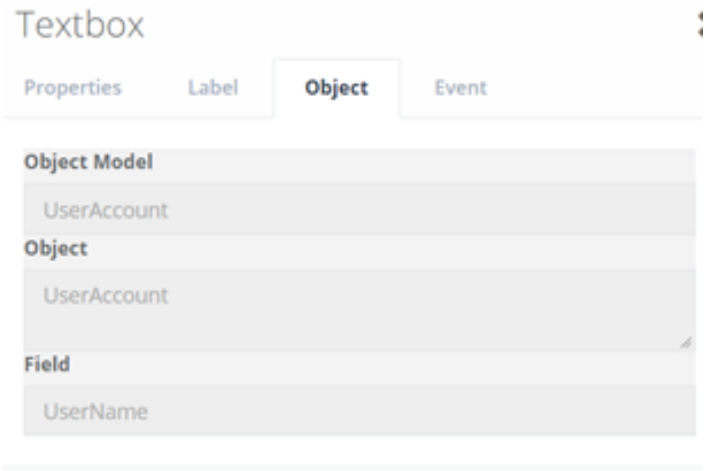


ValidateBillPayRegistration

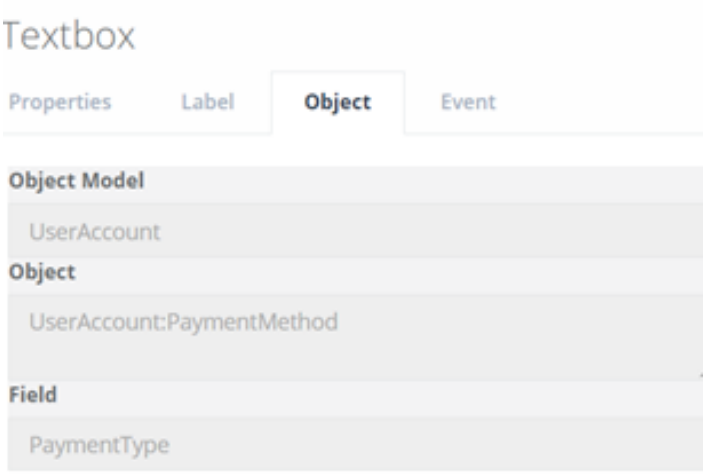


At Page level, object mapping should be as below:

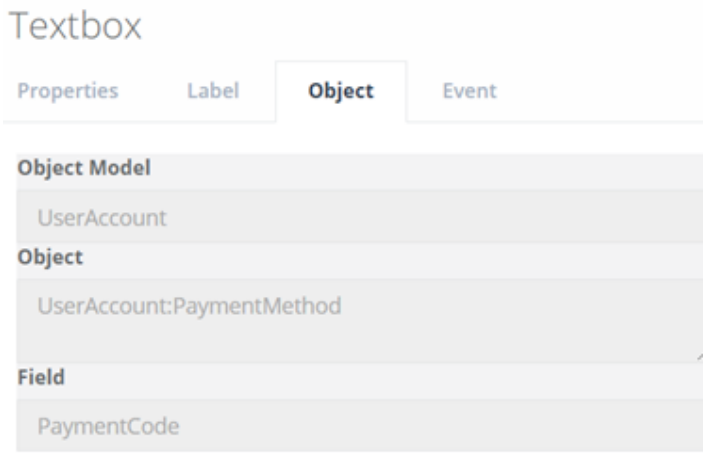
RegisterUser page – UserName field



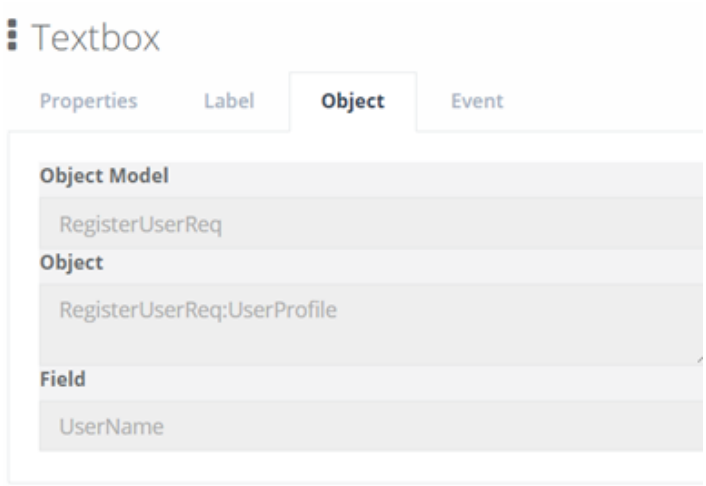
RegisterUser page – PaymentType field



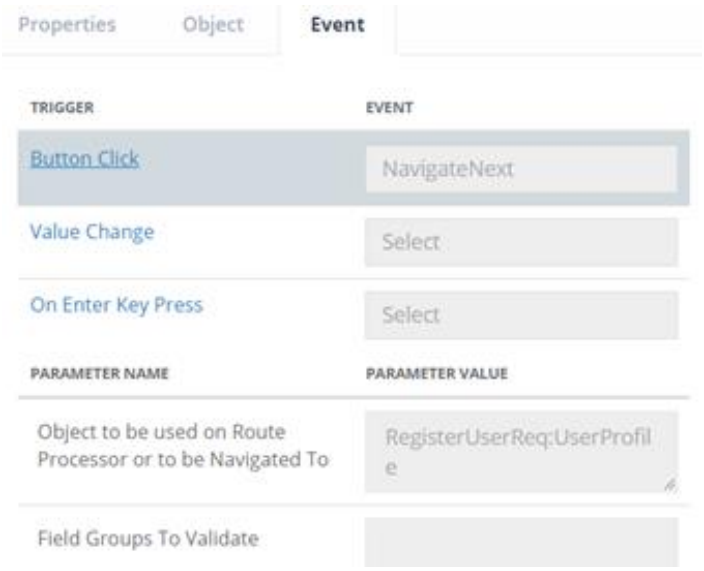
RegisterUser page – PaymentCode field



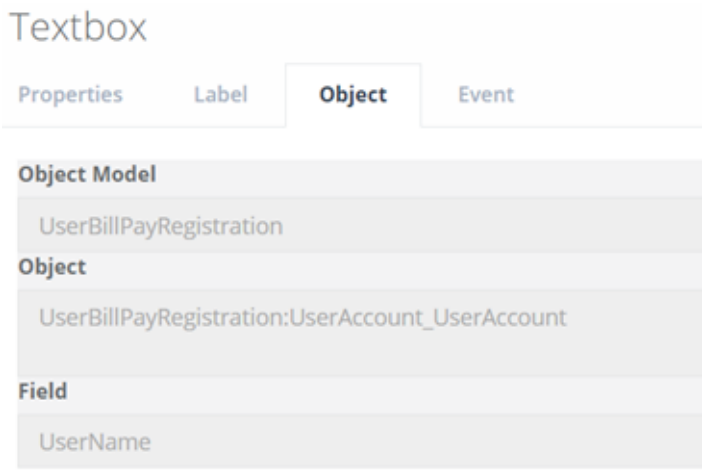
RegisteredUserInfo - Enter existing user to get details field



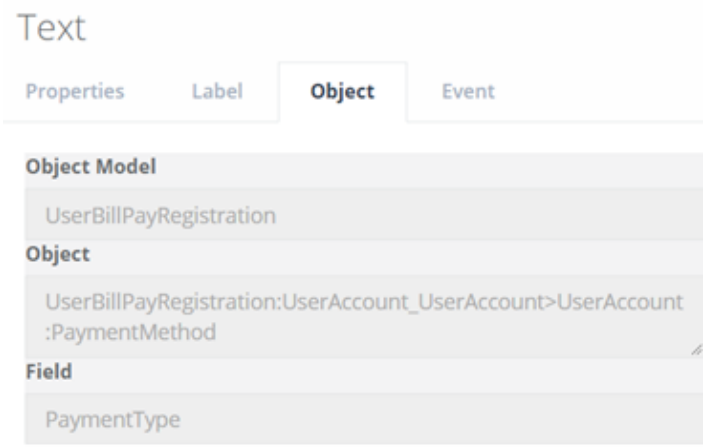
RegisteredUserInfo - Get details button



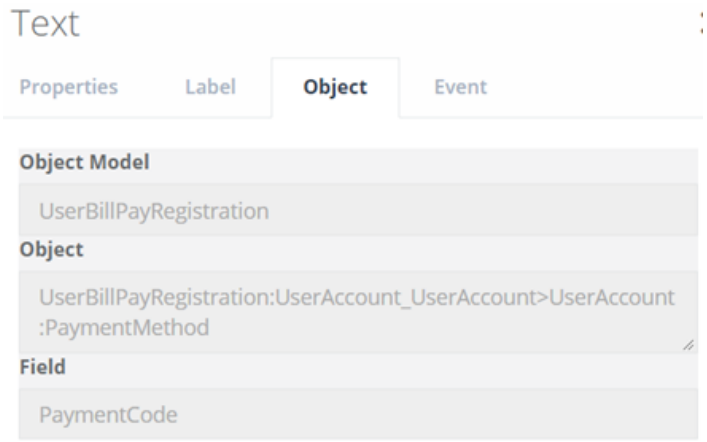
ValidateBillPayRegistration - Registered User field



ValidateBillPayRegistration – Payment Method field

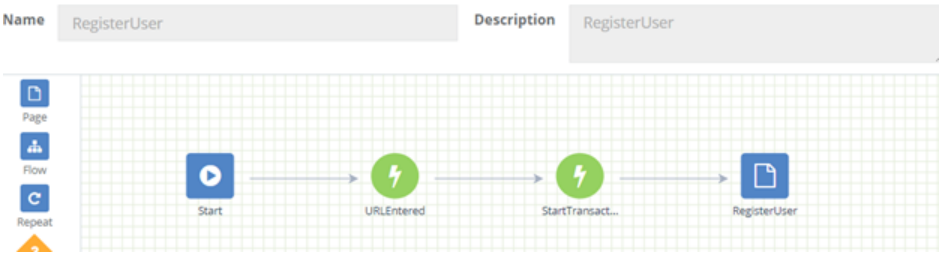


ValidateBillPayRegistration – Payment Code field

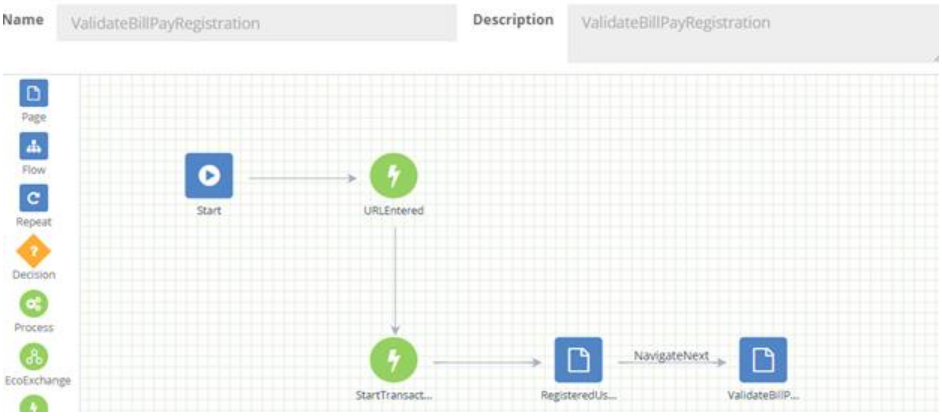


Create Flows as below

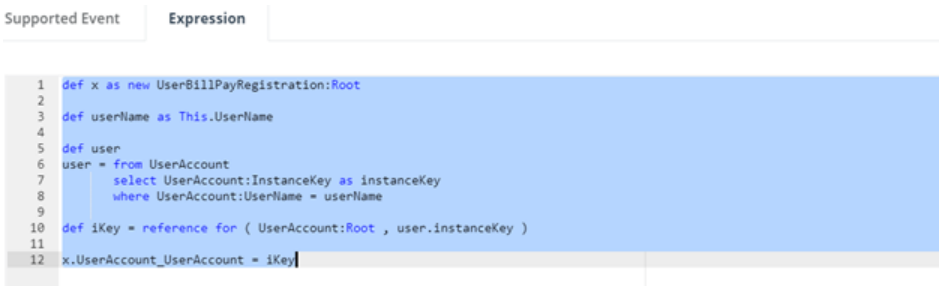
RegisterUser Flow



ValidateBillPayRegistration Flow

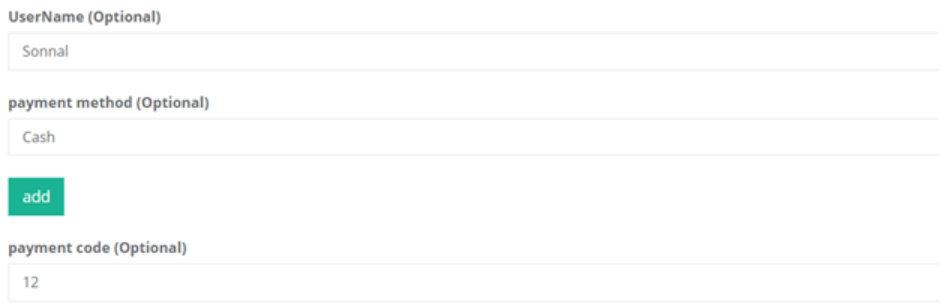


DSL to be written on route that connects 'RegisterUserInfo' and 'ValidateBillPayRegistration' Page



Verify at runtime

Hit on RegisterUser Flow, Enter data and Click on Add button



Hit on ValidateBillPayRegistration flow

Enter username and click on get details, user will be navigated to next page 'ValidateBillPayRegistration'

Enter existing user to get details: (Optional)

Sonnal

get details

Registered User : (Optional)

Sonnal

payment method

Cash

payment code

12

18 Report Functions

18.1 Case and coalesce function

These both Functions can be used at report level for Conditional display of the values in each record of the report. for example to:

1. Case function : CASE expression is the same as IF/ELSE statement in other programing languages. In this general form, each condition is an expression that returns a Boolean value, either true or false. If the condition evaluates to true, it returns the result which follows the condition, and all other CASE branches do not process at all. If all conditions evaluate to false, the CASE expression will return the result in the ELSE part. If you omit the ELSE clause, the CASE expression will return null.

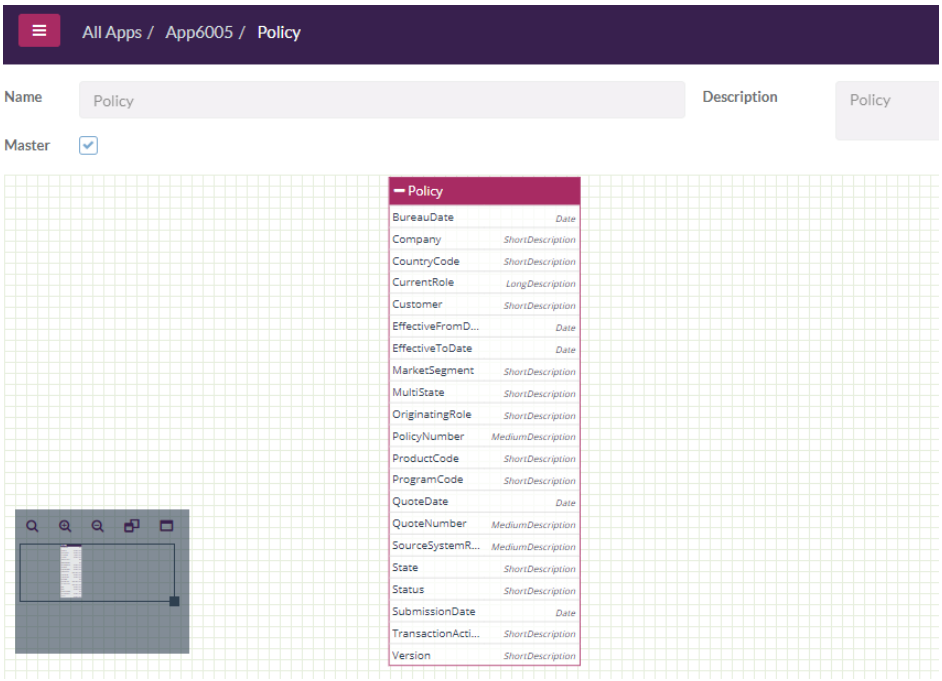
2. Coalesce function : To return the first of its arguments that is not null. Null is returned only if all arguments are null. It is often used to substitute a default value for null values when data is retrieved for display

18.1.1 Syntax

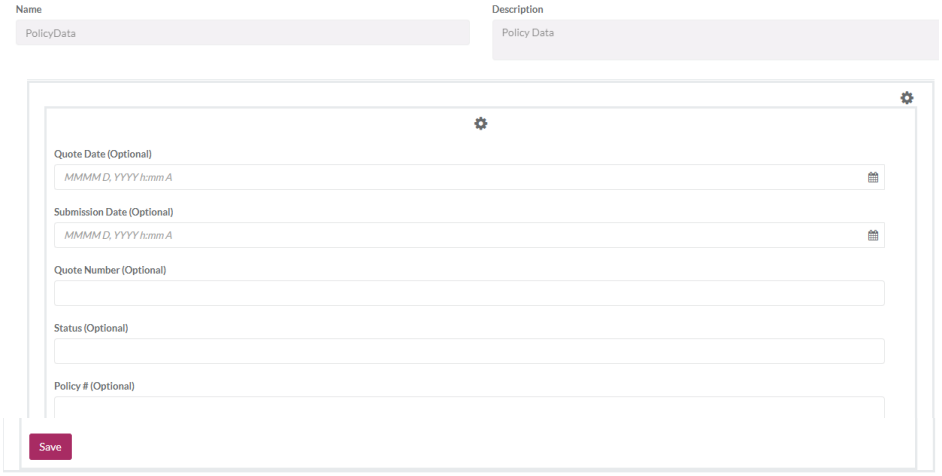
18.1.1.1 Case function

```
case ( <field> when <field|value to match> then <field|value>  when
<field|value to match> then <field|value>  else <field|value> )
Syntax of coalesce function :
coalesce ( <field|value> , <field|value> , <field|value> )
```

Create object model as shown below:



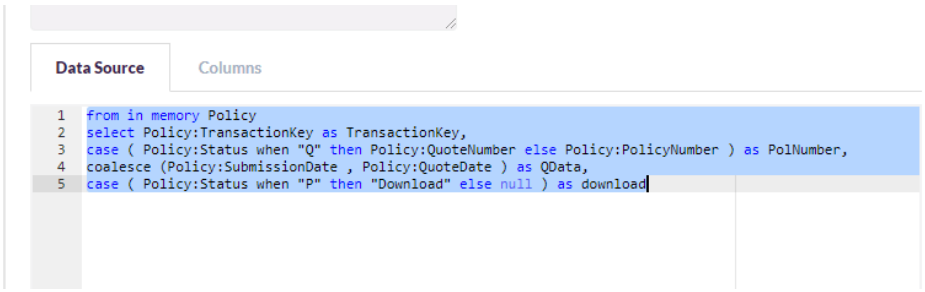
Create Page as show below to capture data



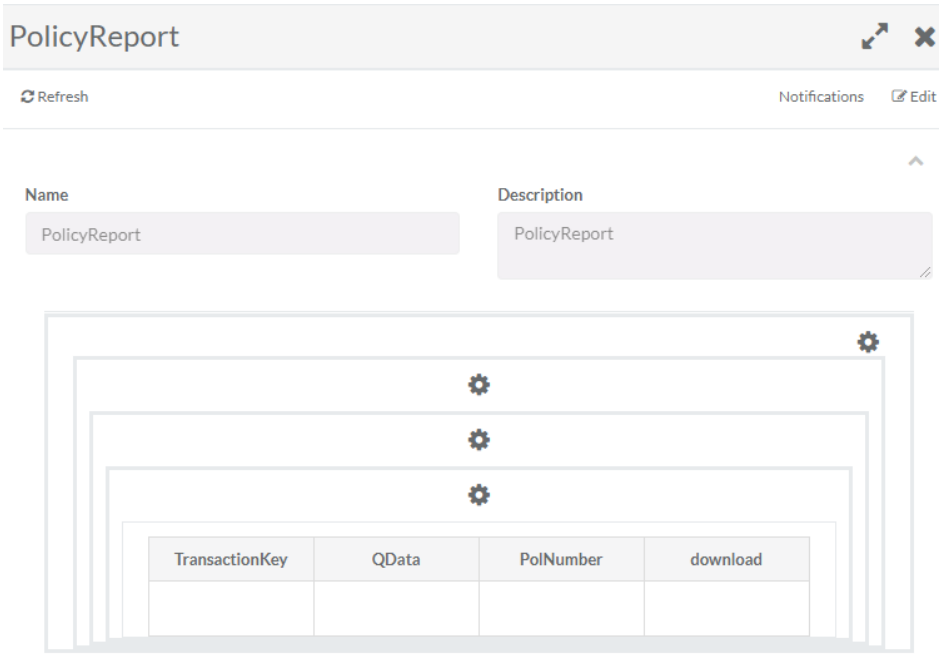
18.1.2 Sample DSL code

```
from in memory Policy  select Policy:TransactionKey as TransactionKey,
case ( Policy:Status when "Q" then Policy:QuoteNumber else
Policy:PolicyNumber ) as PolNumber,
coalesce (Policy:SubmissionDate , Policy:QuoteDate ) as QData,
case ( Policy:Status when "P" then "Download" else null ) as download
```

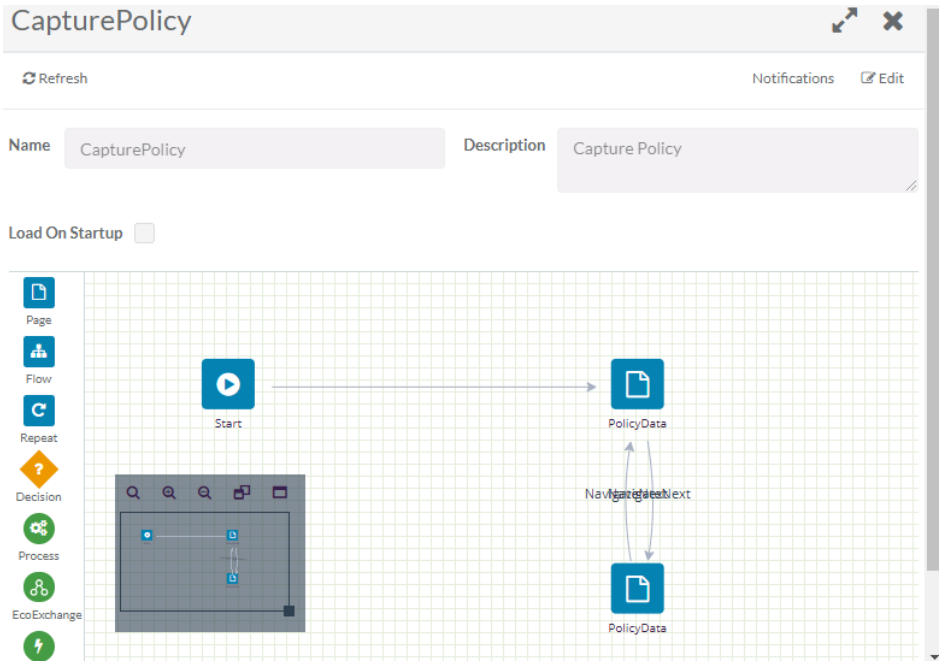
Write it in Data source expression editor of a Report



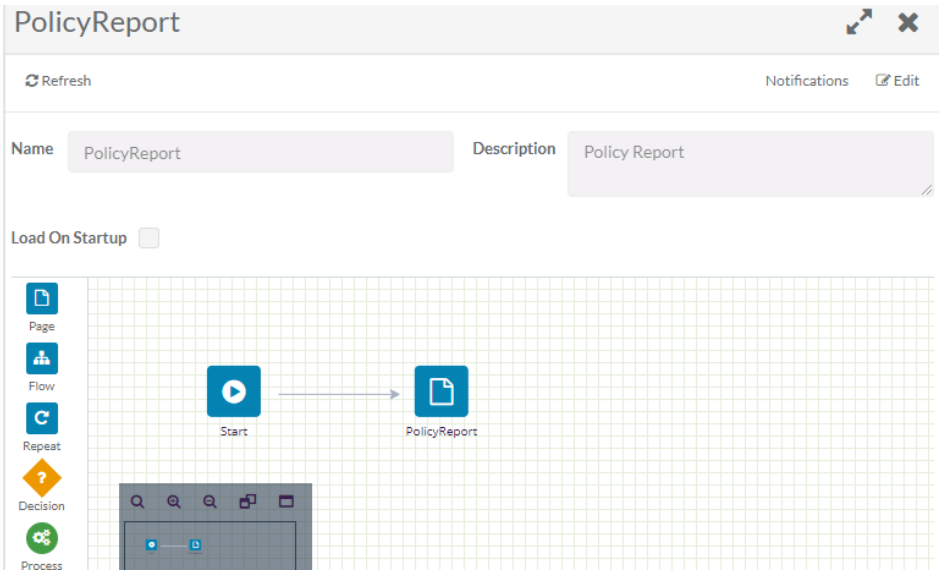
Use this report in a Page with Report widget



Create a flow to capture data

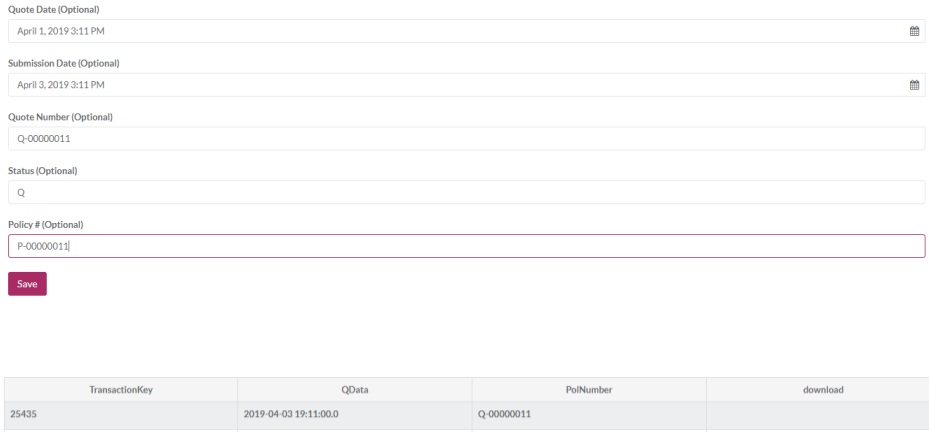


Create one more flow to show report

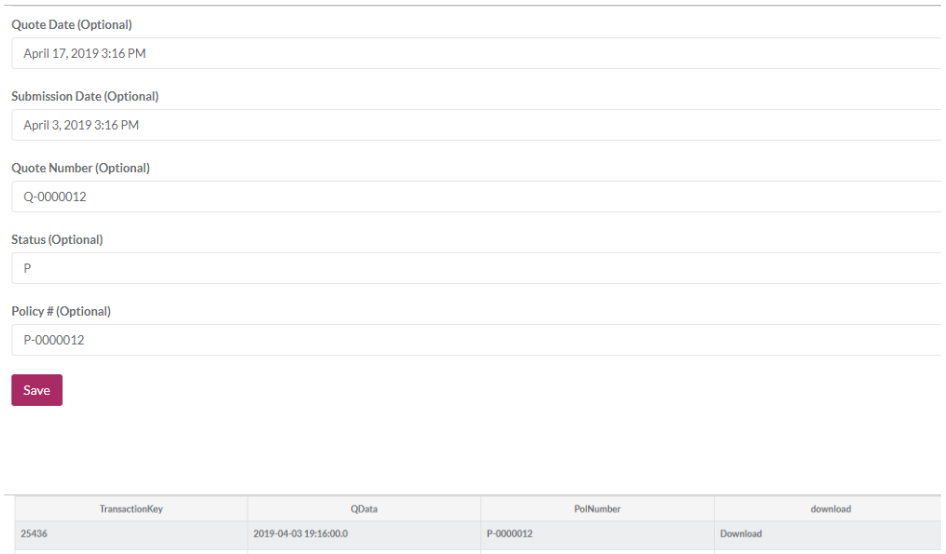


Now check at runtime

1. Use of case function when status is Q, Quote number should be displayed and download column should remain empty.



2. Use of case function when status is P, Policy number should be displayed and under download column 'Download' text value should be displayed.



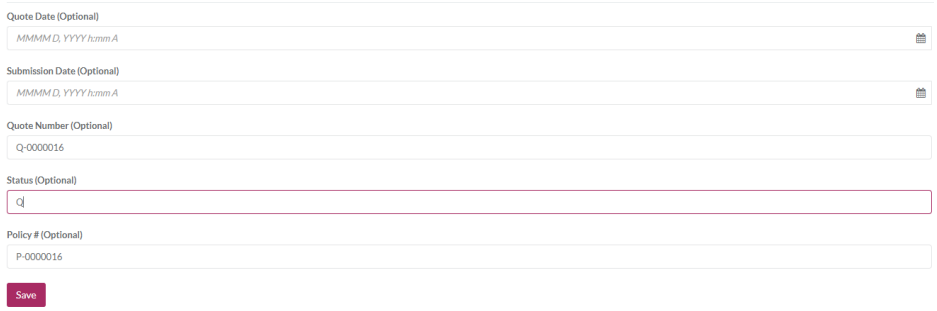
3. Use of coalesce function when SubmissionDate is null and QuoteDate has some value, then at report level QuoteDate should be displayed

TransactionKey	QData	PolNumber	download
25437	2019-04-10 19:21:00.0	P-0000014	Download

4. Use of coalesce function when SubmissionDate has some value and QuoteDate is null, then at report level SubmissionDate should be displayed

TransactionKey	QData	PolNumber	download
25437	2019-04-09 19:22:00.0	P-0000015	Download

5. Use of coalesce function when SubmissionDate and QuoteDate is null, then at report level column should display null value.



TransactionKey	QData	PolNumber	download
25438		Q-0000016	

6. Use of case function when status is null, then column value should policy number

Quote Date (Optional)

April 10, 2019 3:41 PM

Submission Date (Optional)

April 9, 2019 3:41 PM

Quote Number (Optional)

Q-0000018

Status (Optional)

Policy # (Optional)

P-0000018

Save

TransactionKey	QData	PolNumber	download
25439	2019-04-09 19:41:00.0	P-0000018	

19 Application Context Aware Expression

Snowflake provide access to application context information like logged in user, application variable (default screen size, default layout) using context aware expressions.

For example DSL can be written on :

- In an App - On a page - any widget > Settings icon > anywhere the '+' icon is available > Advanced logic in when configuration
- In an App - On a page - any widget > Settings icon > anywhere the '+' icon is available > is expression in then condition
- In an App - On a page - any widget > Settings icon > anywhere the '<>' icon is available > Expression editor
- In an App - On a page > In inner row widget > Click here to enter logic' link > Expression editor
- In an App - On a flow route > Settings icon > Event/Expression editor
- In an App - In a report > Expression editor

19.1 Security Context Access

General syntax to access Security context is
security.<ContextVariable> {parameter}

Context Variable	Description
CURRENT_ROLE	Return current role of the logged in user
USER_NAME	Return user name of logged in user
USER_LAST_LOGIN	Return last login time in string format
HAS_ROLE <roleIdParameter>	Return whether user has specified user role assigned

1. security.CURRENT_ROLE
return security.CURRENT_ROLE
Above snowflake expression will return logged in user' s role

2. security.USER_NAME
return security.USER_NAME
Above snowflake expression will return logged in user's name

3. security.USER_LAST_LOGIN
return security.USER_LAST_LOGIN
Above snowflake expression will return logged in user's last login time

4. security.HAS_ROLE
return security.HAS_ROLE underwriter

Above snowflake expression will return true if logged in user has role assigned as underwriter

19.2 Special Context

Context is made available for users that has some special client aware syntax.
For example DSL can be written on :

- In an App - On a page - any widget > Settings icon > anywhere the '+' icon is available > Advanced logic in when configuration
- In an App - On a page - any widget > Settings icon > anywhere the '+' icon is available > is expression in then condition
- In an App - On a page - any widget > Settings icon > anywhere the '<>' icon is available > Expression editor
- In an App - On a page > In inner row widget > Click here to enter logic' link > Expression editor
- In an App - On a flow route > Settings icon > Event/Expression editor
- In an App - In a report > Expression editor

Context Variable	Description	Sample values that will be made available
Context Variable	Description	Sample values that will be made available
context.CLIENT_DEVICE_TYPE	Returns the device type the client is using.	mobile, tablet, desktop, unknown
context.CLIENT_BROWSER	Returns the client's device browser.	chrome, edge, firefox, ie, opera, safari, unknown
context.CLIENT_ORIENTATION	Returns the orientation the client is using currently.	portrait, landscape, unknown
context.CLIENT_OS	Returns the client's OS.	ios, iphone, ipad, ipod, android, blackberry, macos , windows, fxos, meego, television, unknown
context.CLIENT_SCREEN_WIDTH	Returns the client's screen width size	Actual number indicating screen width.

20 Use of transaction expressions

Snowflake allows to use events in an expression. As of now only Start transaction and complete transaction, these two base events can be accessed through DSL code.

Expression	Description
start transaction	To initiate a transaction or resume transaction
complete transaction	To mark/update a transaction as completed

Start transaction

Syntax: start transaction (MasterModel:Root , x.transactionKey) or start transaction (MasterModel:Root ,8005)

Above mentioned DSL codes can be written in a flow, on a route connecting to 'Start' step and any other step.

Complete transaction

Syntax: complete transaction (MasterModel:Root , x.transactionKey) or complete transaction (MasterModel:Root ,8005)

Above mentioned DSL codes can be written in a flow, on a route connecting to 'Start' step and 'URL entered' event step.

1. start transaction – to initiate a transaction or resume transaction

start transaction (PolicyModel:Root , 1)

Above block statement will result in resume transaction with transaction id as 1

2. start transaction – to initiate a transaction or resume transaction

def x

x = from PolicyModel

```
        select PolicyModel:TransactionKey as transactionKey
        where PolicyModel:PolicyNumber = "Policy1234"
  start transaction ( PolicyModel:Root , x. transactionKey )
   # Above block statement will result in resume transaction with
transaction id received after querying database with PolicyNumber as
"Policy1234"
```

3. start transaction – to initiate a transaction or resume transaction

start transaction (PolicyModel:Root)
 # Above block statement will result in start transaction

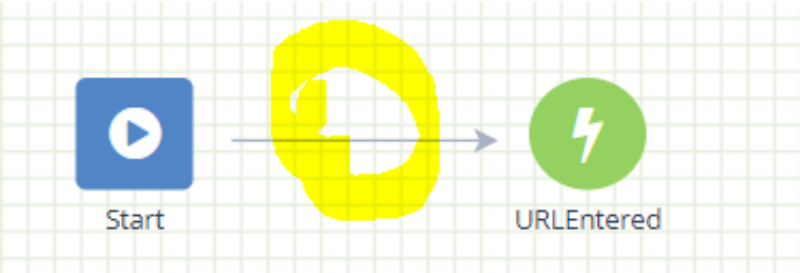
4. complete transaction – to a complete transaction
complete transaction (PolicyModel:Root, 1)
 # Above block statement will result in marking transaction with id 1 as completed

5. complete transaction – to complete a transaction

```
def x
 x = from PolicyModel
       select PolicyModel:TransactionKey as transactionKey
       where PolicyModel:PolicyNumber = "Policy1234"
 complete transaction ( PolicyModel:Root , x. transactionKey )
  # Above block statement will result in marking a transaction as complete
with transaction id received after querying database with PolicyNumber as
"Policy1234"
```

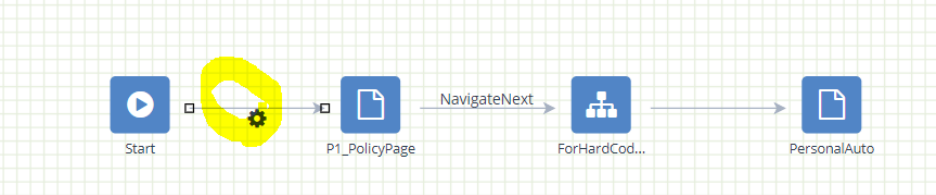
Refer below mentioned screens to which illustrates usage of start and complete transaction.

Create a flow as shows below and write DSL on highlighted route




```
def y as  MasterModel:Root
def ins as new MasterModel:PersonalAuto
def x
x = from MasterModel
select MasterModel:TransactionKey as transactionKey
where MasterModel:PolicyNo = "Policy1234"
ins.Premium = x.transactionKey
y.PersonalAuto = ins
return y
```

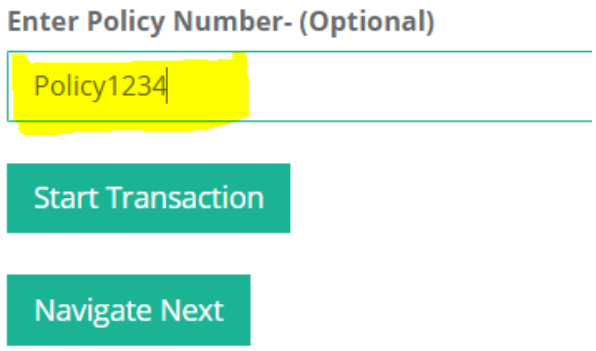
Create one more flow and used above flow in it



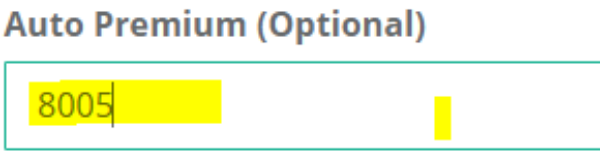
Write DSL as show below

```
def x
x = from MasterModel
select MasterModel:TransactionKey as transactionKey
where MasterModel:PolicyNo = "Policy1234"
start transaction ( MasterModel:Root , x.transactionKey )
```

Check at runtime, Policy number is auto populated based on DSL execution.



For objectmodel relation transaction is started with transactionKey = 8005. Hence the output is 8005.



21 Snowflake code for Policy Life Cycle

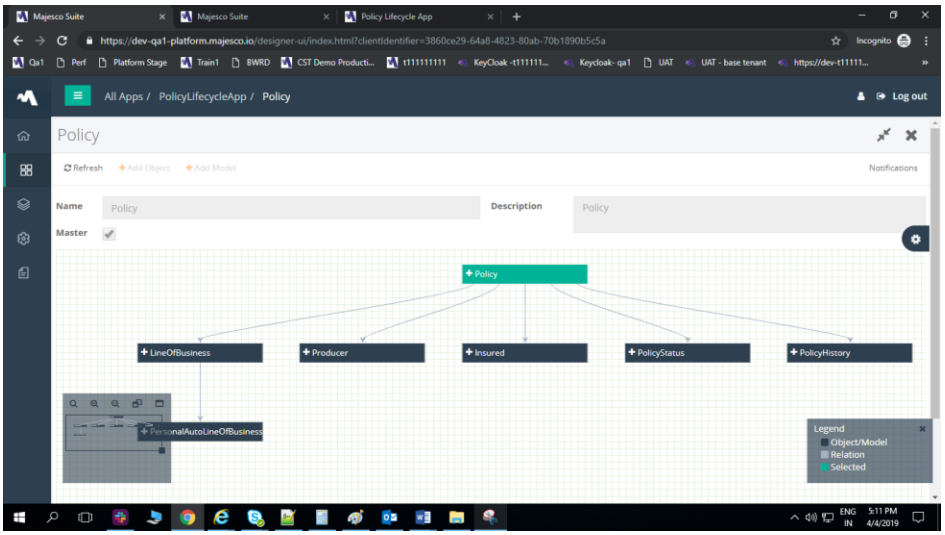
Policy Life cycle involves transactions like Quote, Issuance Transaction, Endorsement, Under notice, Cancellation, Reinstatement, Renewal, Revise and Clone transaction.

This snowflake document covers only Issuance and clone transactions as of now.

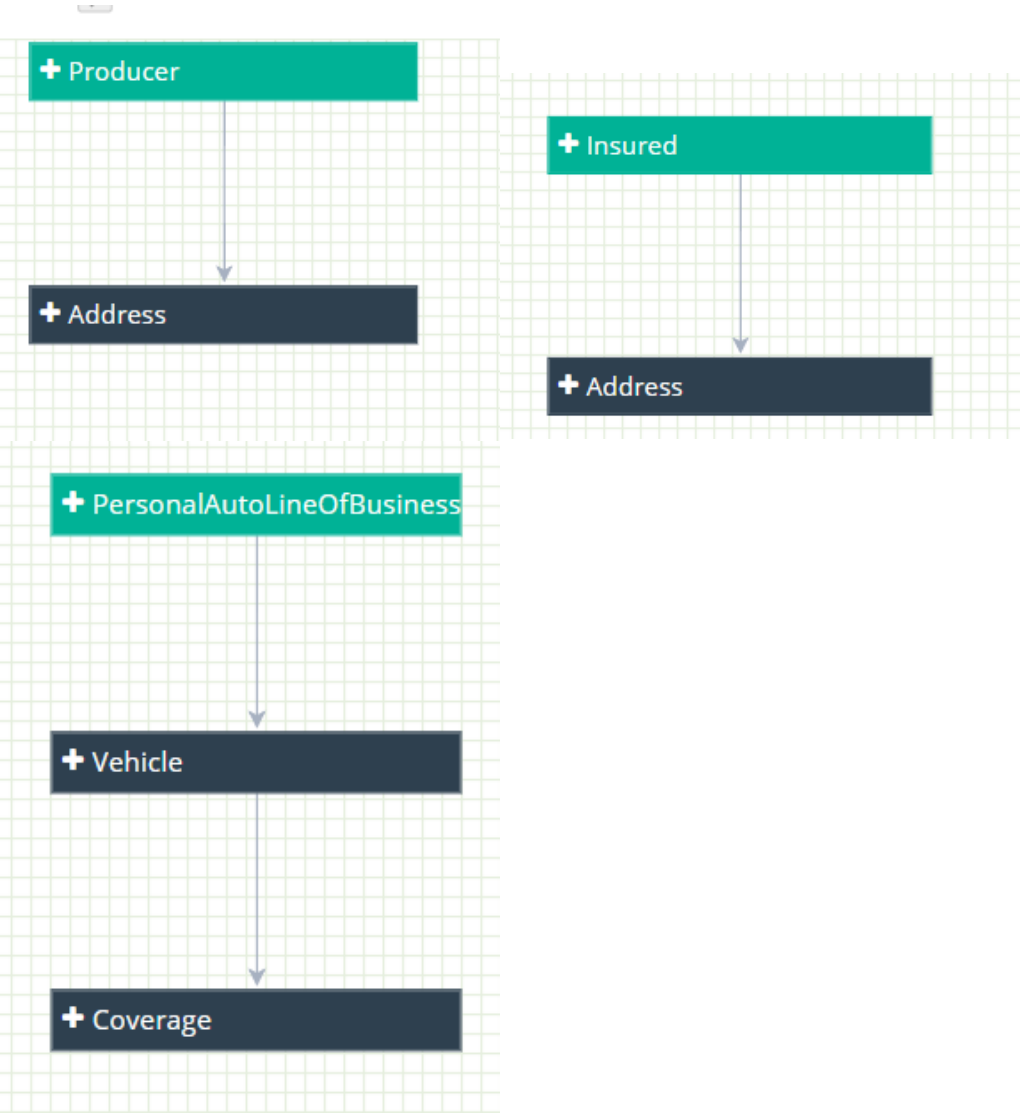
Refer below snowflake code, which should be written in Expression editor available for route drawn from Start step to start the existing transaction.

To perform Policy issuance transaction, perform below mentioned steps.

Create Master object model as described below



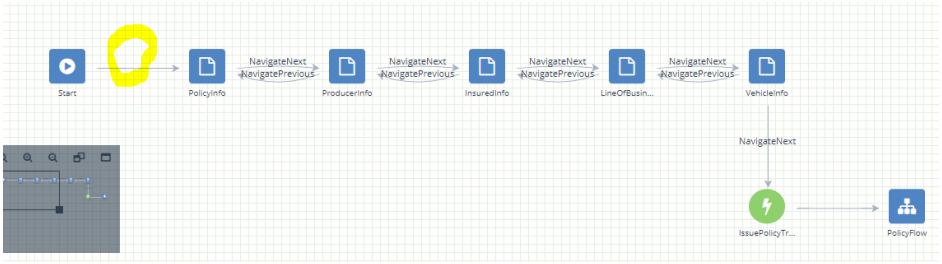
With Produce, Insured, PersonalAutoLineBusiness as child master models as shown below.



Create Pages as shown below by using above objects and object models relations

PAGE NAME	APP	SOURCE
InsuredInfo	PolicyLifecycleApp	Custom
LineOfBusinessInfo	PolicyLifecycleApp	Custom
PolicyInfo	PolicyLifecycleApp	Custom
ProducerInfo	PolicyLifecycleApp	Custom
VehicleInfo	PolicyLifecycleApp	Custom

Create Flow as displayed below, write above DSL on highlighted route.



Check at runtime the changes on PolicyInfo page.

PolicyInfo Page

State (Optional)

NJ

Policy Type (Optional)

Personal

Premium (Optional)

500

Status (Optional)

Quote

Renewal number

PrevLifeCycleTransactionNoRef

Next

8692

Original transaction number

Revision number

Latest

Locked

LifeCycleTransactionNoRef

Transaction code

ProducerInfo Page

FirstName (Optional)

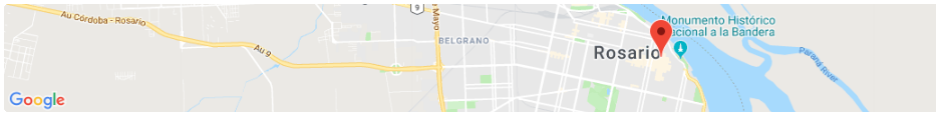
Deepa

Last Name (Optional)

Pingle

8694

Address line1 (Optional)



Santa Fe 1055, Rosario, Santa Fe Province S2000, Argentina

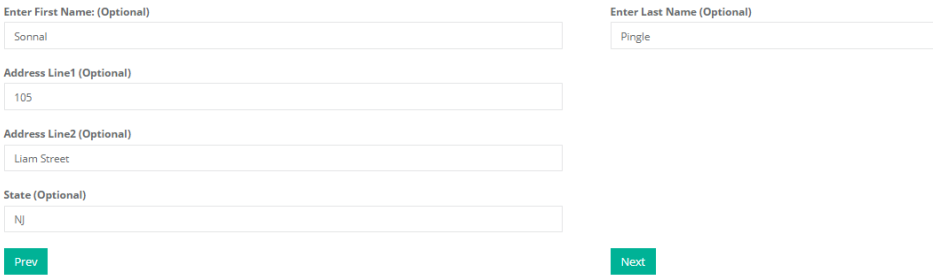
State (Optional)

NJ

Prev

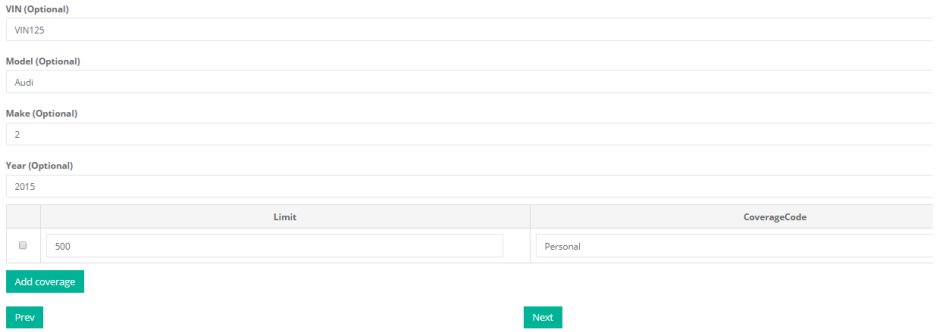
Next

InsuredInfo Page

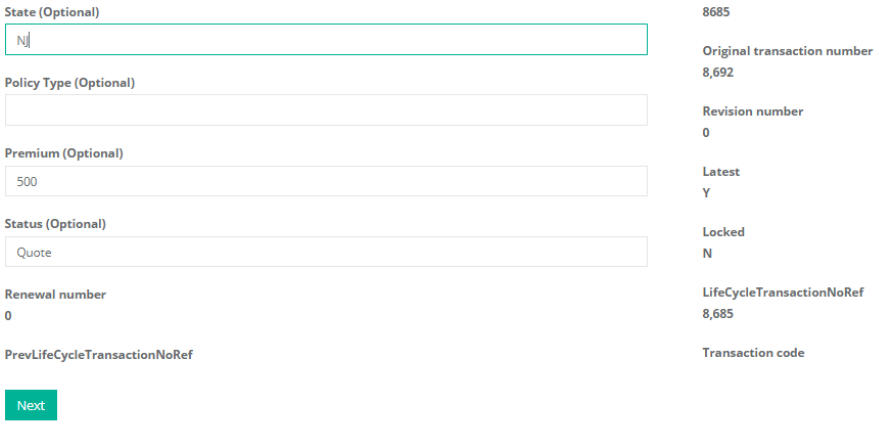


LineofBusiness Page

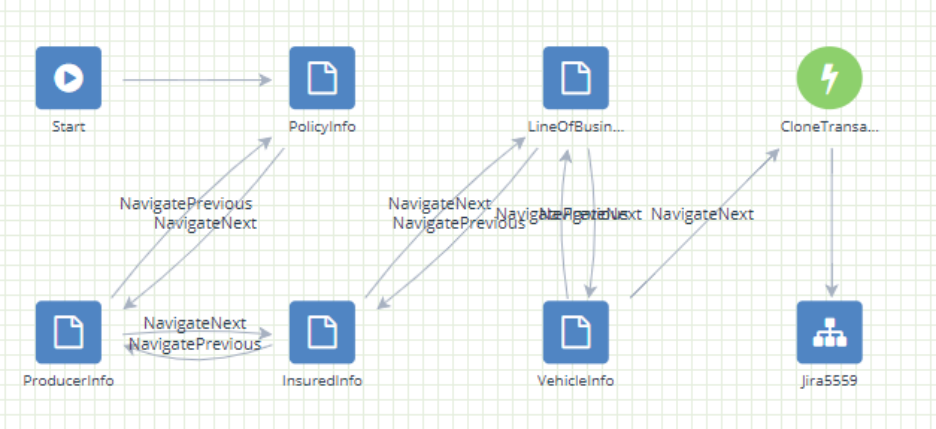
VehicleDetails Page

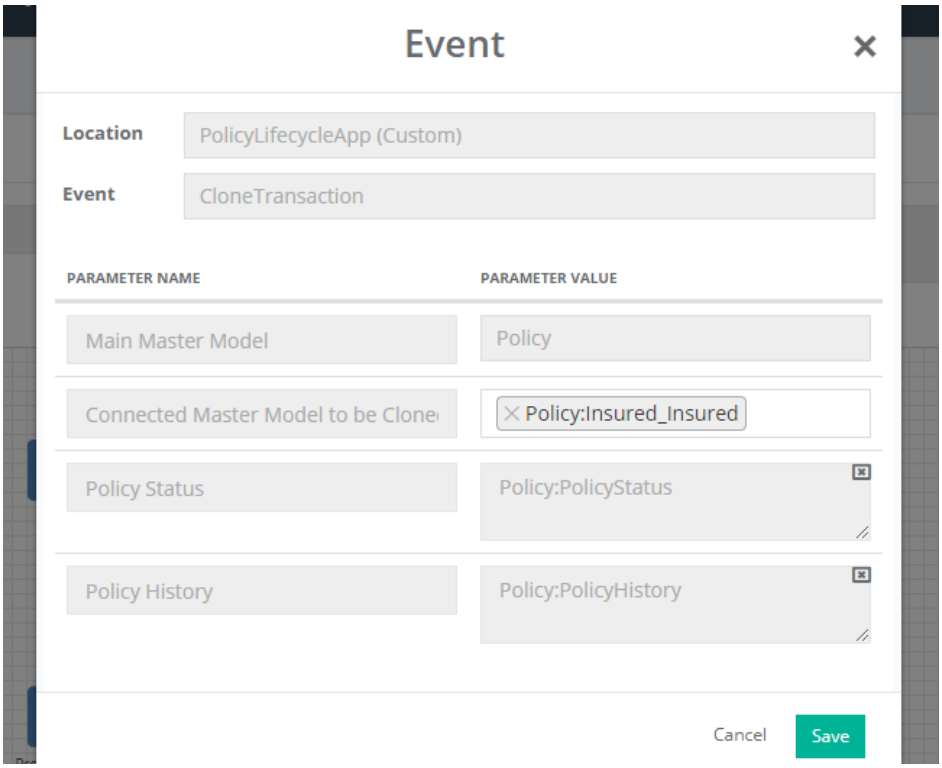


Click next and check changes on PolicyInfo Page



Same logic is to be used with clone transaction, with below change at flow level.





22 Newly added keywords

keywords	Description
sentence	returns case of words(lower or upper case) as it would be in a sentence, when input is string of words with any case.
columns	returns number of columns of a record
rows	returns number of columns of a record
number of records	returns number of records of a column

i] sentence

i/p=> return sentence("this is a Test for Sentence.lets see how it works.")
expected result - This is a test for sentence.Lets see how it works.

ii] columns

if you are getting a value as Record in snowflake (like lookup resultset into variable or MQL resultset into a variable). Then you can use columns to get the number of columns in a record and rows to get number of rows in a result set.

Example :-

```
    if SingleLineAddress <> null then
   def x as split SingleLineAddress separated by ","
        for y as 1 to columns(x) do
        if y = 1 then
               return value from x at y
       end
       end                # return value from (x, 1)
       end
```

iii] rows

If you are getting a value as Record in snowflake (like lookup resultset into variable or MQL resultset into a variable). Then you can use rows to get number of rows in a resultset.

Example :-

```
    if SingleLineAddress <> null then
   def x as split SingleLineAddress separated by ","
        for y as 1 to columns(x) do
        if y = 1 then
               return value from x at y
       end
       end                # return value from (x, 1)
       end
```

iv] number of records

22.1 To be added keywords

Keyword	Description
decrement	(TBD) Only increment functionality is there in Snowflake. 'decrement' should also be a keyword. If functionality is already added, this should be added in the list.
format time	(TBD) format date does not accommodate only formatting time as of now. If functionality is already added, this should be added in the list.
exact	(TBD) 'with optional match' functionality is added in Snowflake and uses 'exact' keyword as well e.g. 'with exact match'. If functionality is already added, this should be added in the list.

23 Compiler behavior

- Syntax Errors will take priority over all other errors - Compiler will throw Syntax error first if both syntax and logical errors are found in the code.
- Syntax Errors will take precedence over all other errors - If at line 1 logical error has occurred and at last line syntax error has occurred, then Compiler will throw Syntax error first if both syntax and logical errors are found.
- Only one error will be thrown at a time - even if code has multiple errors and errors of any type.
- Only one error will be thrown at a time - even if all errors are syntax errors and more than one syntax errors exist on single line.

24 Error types

This area is still needs consensus building over requirement, but there is some broad level understanding on this. The overall idea is that all errors should be-

- User friendly
- User understandable (not complex or abbreviated or use of jargon)
- User must be able to exactly pin point the source of error for example - which widget, which label, which line, which position etc.
- Error title must give idea as to whether the error is of type - Logical / Syntax / Data Type mismatch etc.
- Error description must have field ID/label ID mentioned so that user can easily identify exactly which widget has caused the error. for example-
  If a page has two Display Text widgets both having erroneous codes then the error thrown must have widget name and its label mentioned in the error.
- Error occurred on a flow route must have specific route mentioned in the error message.
- Error description must be explaining the correct error and explaining the error correctly.