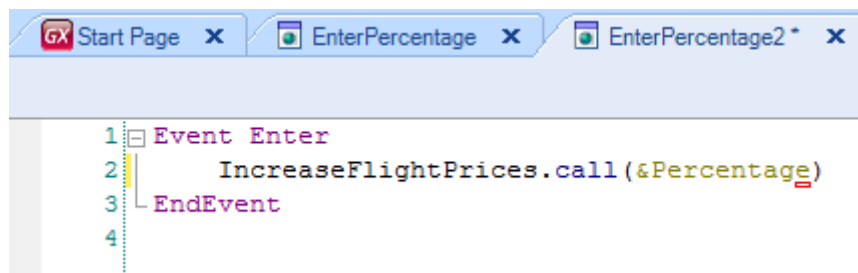


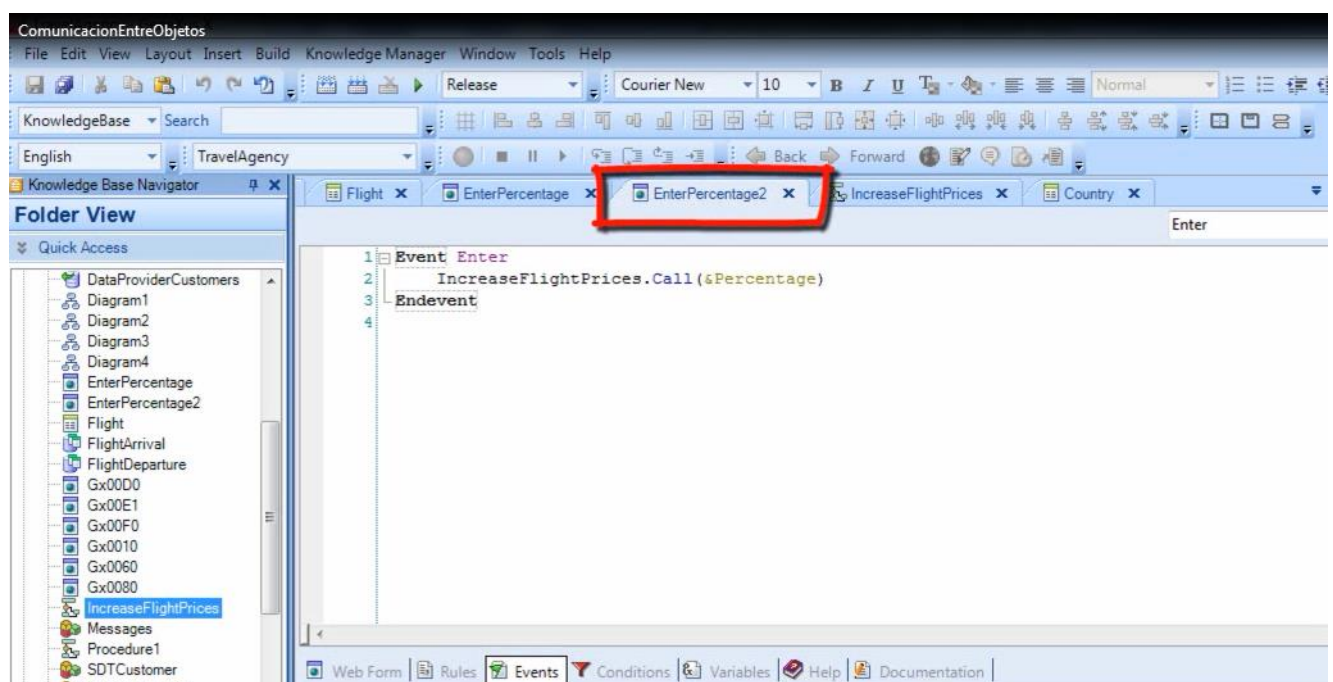
GeneXus course - Communication Between Objects



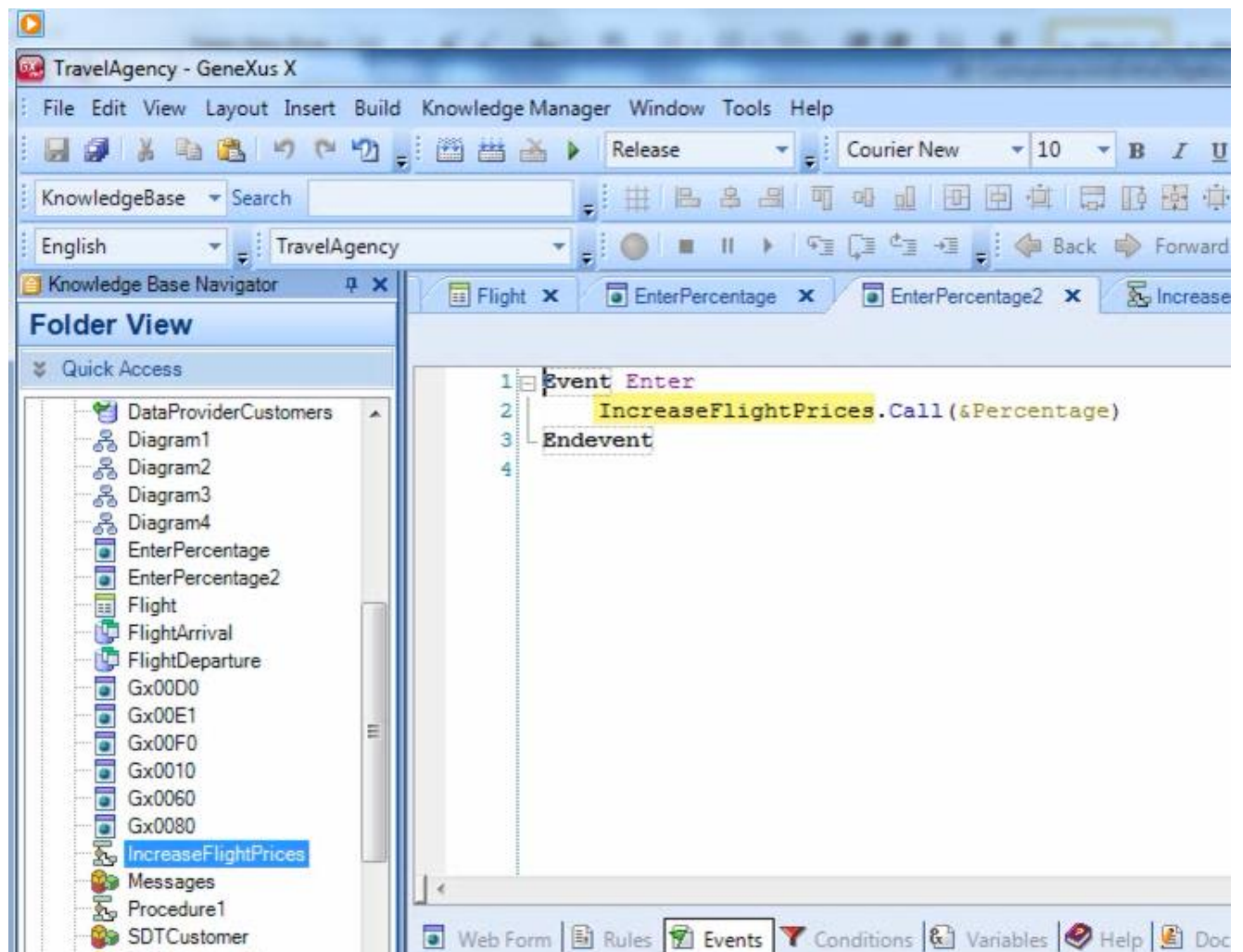
```
1 Event Enter
2     IncreaseFlightPrices.call(%Percentage)
3 EndEvent
4
```

In previous situations we have been faced with the need to call an object from another object.

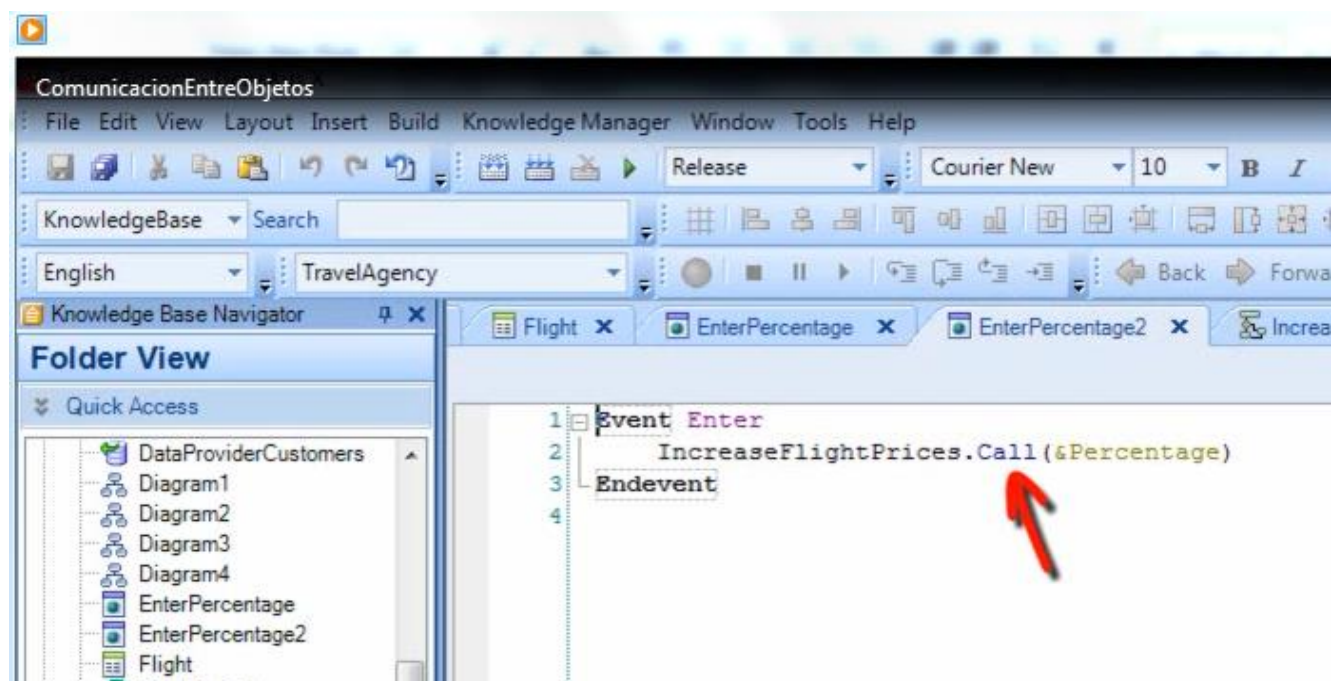
For example, in the Enter event of the “EnterPercentage2” web panel,



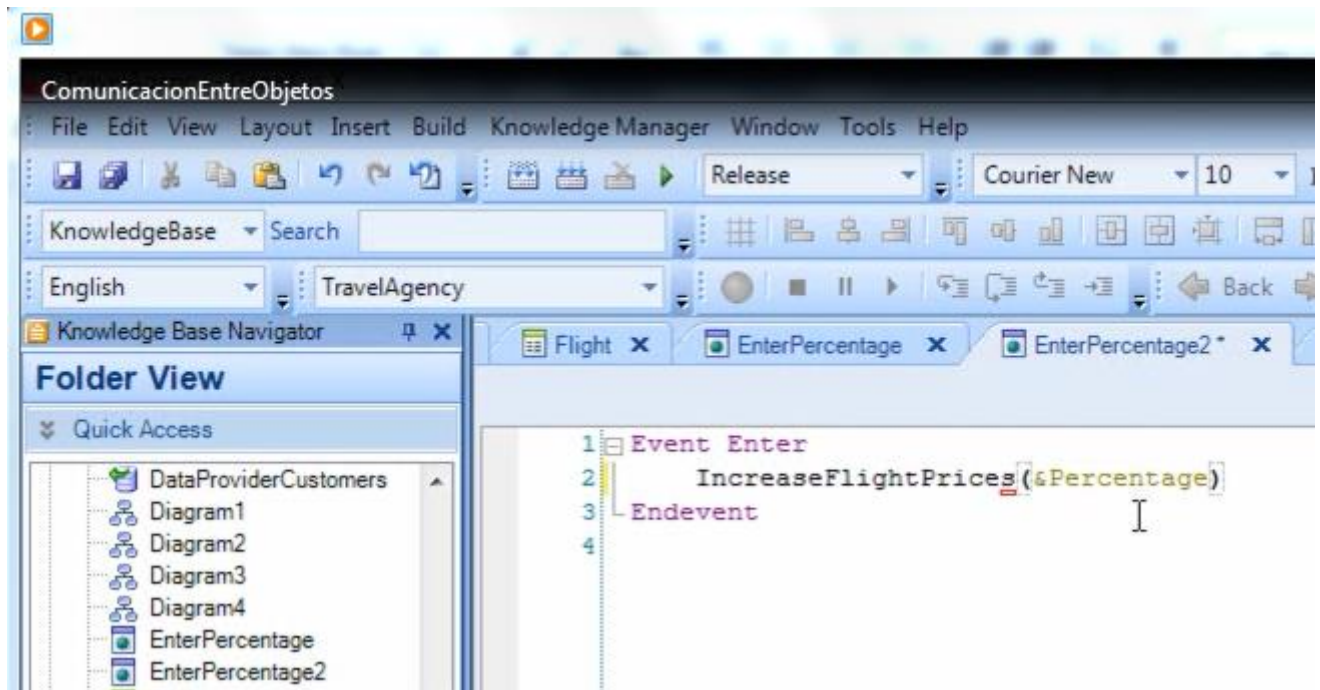
we are calling the “IncreaseFlightPrices” procedure



with the call method:

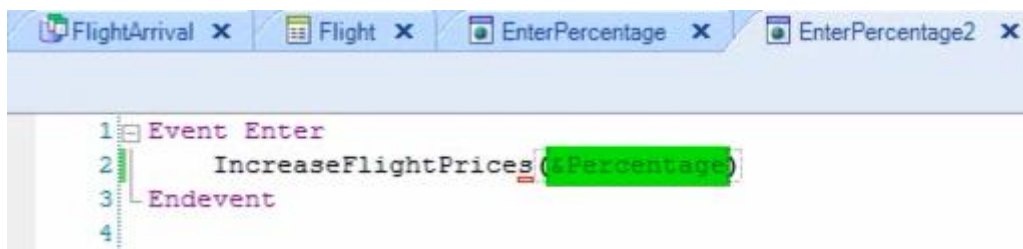


The syntax of calls allows us to omit the call and the operation will be exactly the same, because GeneXus has the intelligence to detect that we are calling an object, so we will delete the dot and the call.



Remember that when we call an object we can include data that we have in the calling object, for the object called to know them and use them.

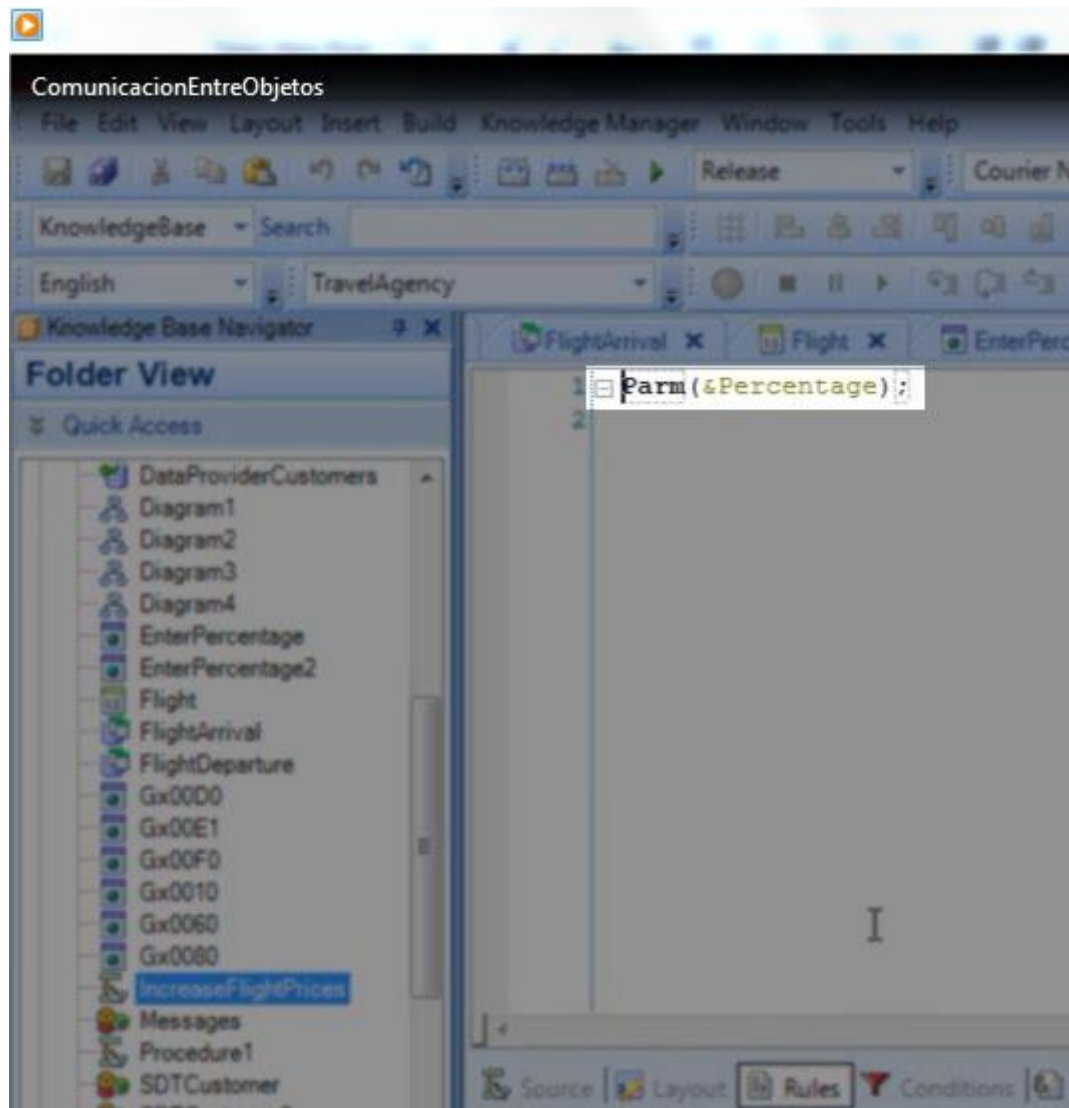
In this example we are passing on to the procedure



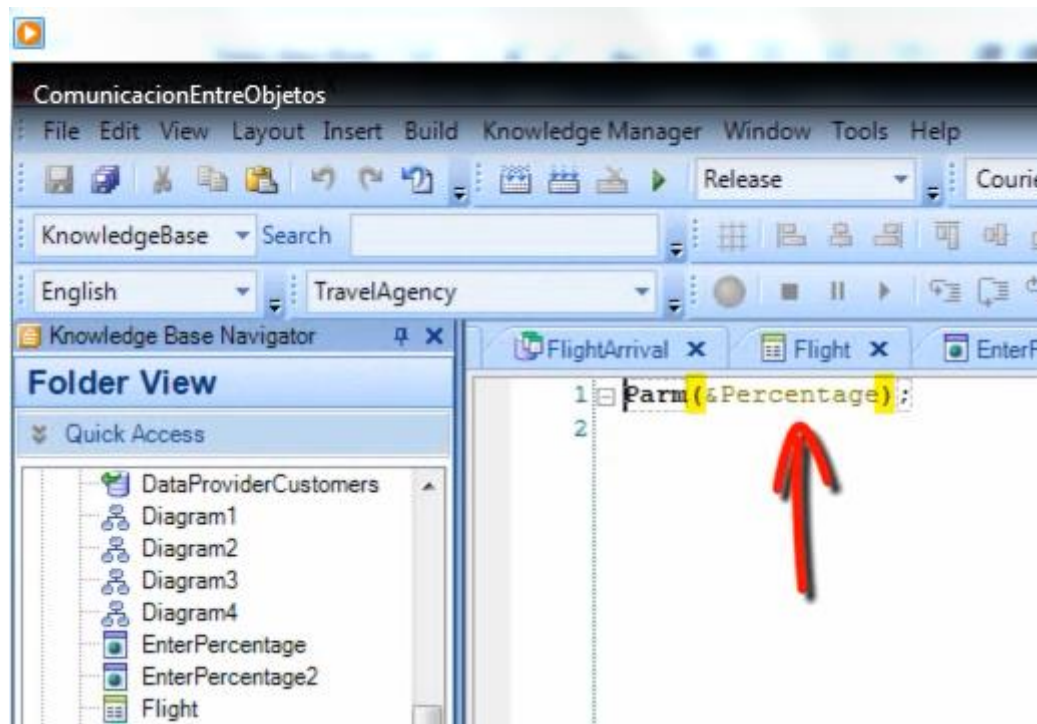
the value of the percentage entered by the user in the web panel, that is stored in the &percentage variable.

In this case, we have the value in a variable. If it were in an attribute we would include the corresponding attribute inside the parenthesis, or if we had to pass on two or more values we would send attributes and/or variables separated with commas.

As we have seen, in the object called we declare the data received in the parm rule.



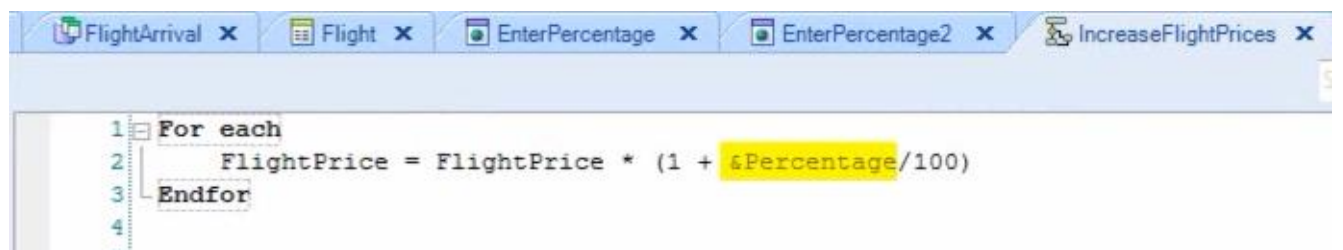
In this rule we can see the &percentage variable between parentheses.



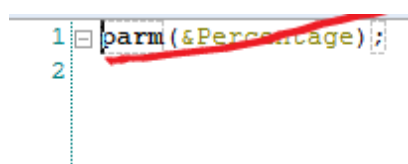
In this object we have defined the variable with the same name and data type as it is known with in the web panel. The name could have been different but the data type must be the same.

| Name | Type | Is Collection | Description |
|--------------------|--------------|--------------------------|-------------|
| Variables | | | |
| Standard Variables | | | |
| Percentage | Numeric(3.0) | <input type="checkbox"/> | Percentage |

We should remember that the variable received with the percentage value is used in the source of the procedure.



If we do not include data when we call an object, then we **don't have to declare a parm rule** in the object called.

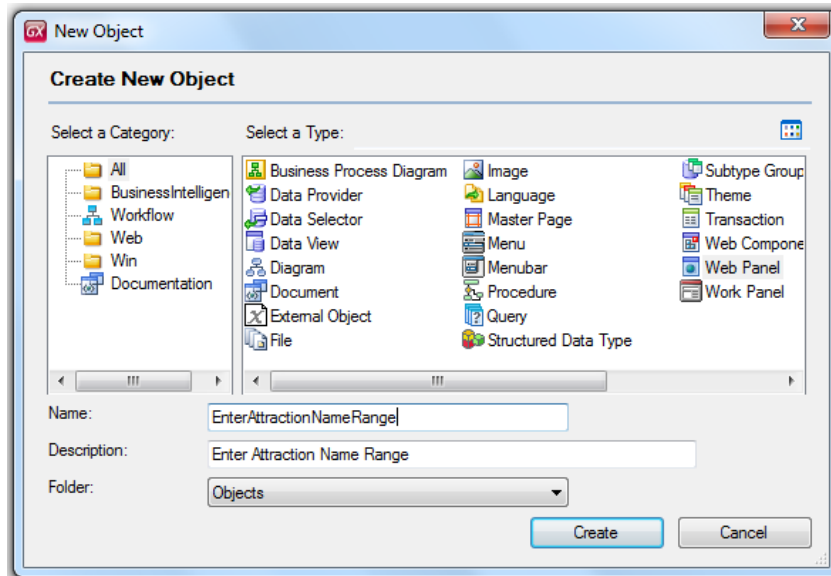


If we include more than one piece of data, they will have to be received within the parm rule in the same order and separated by commas.

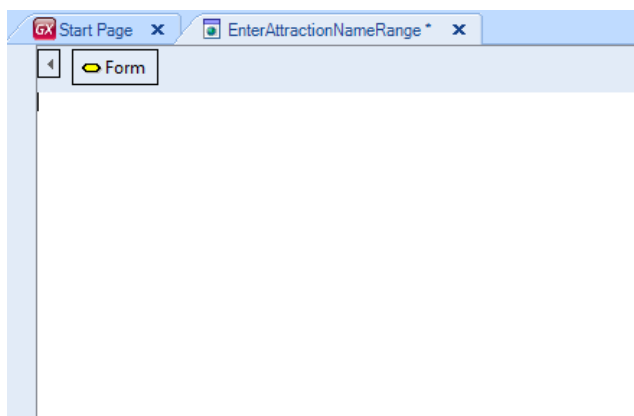
Let's see an example of this.

We will define a web panel for the user to enter a start and end range of names of attractions to be listed. From the web panel we call a procedure to list all the attractions with names included in the range.

New/Object, select web panel, and name it: EnterAttractionNameRange

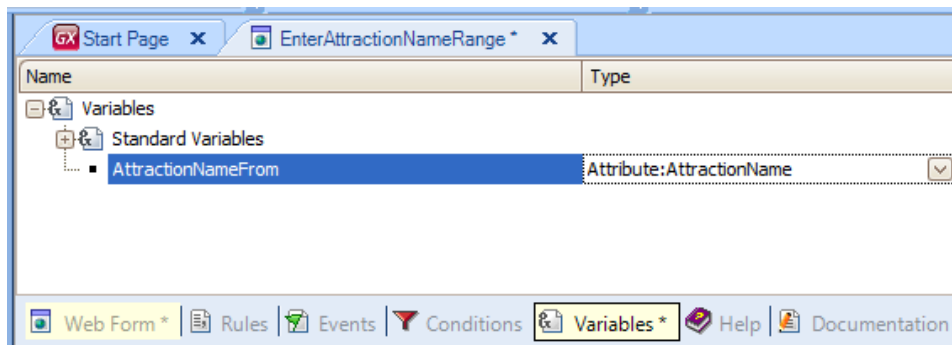


Press Create

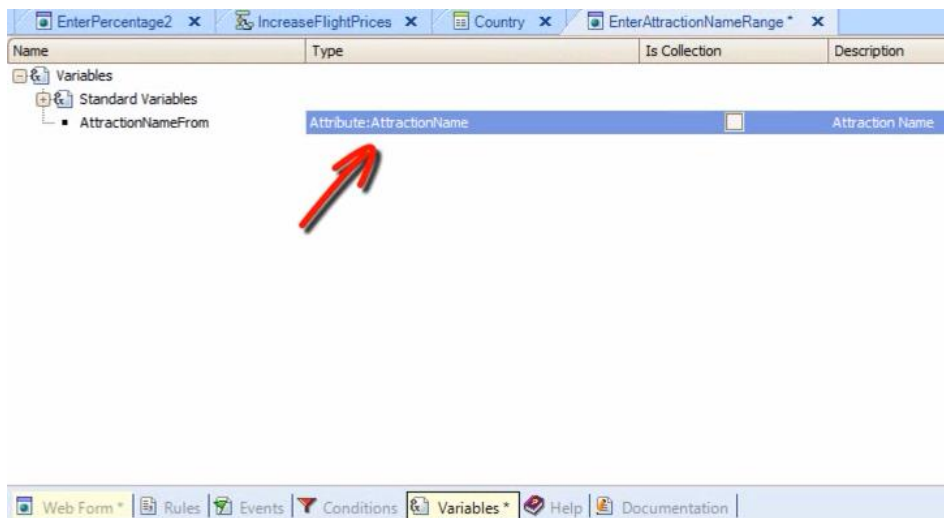


Then we go to the variables section to create the two variables we need.

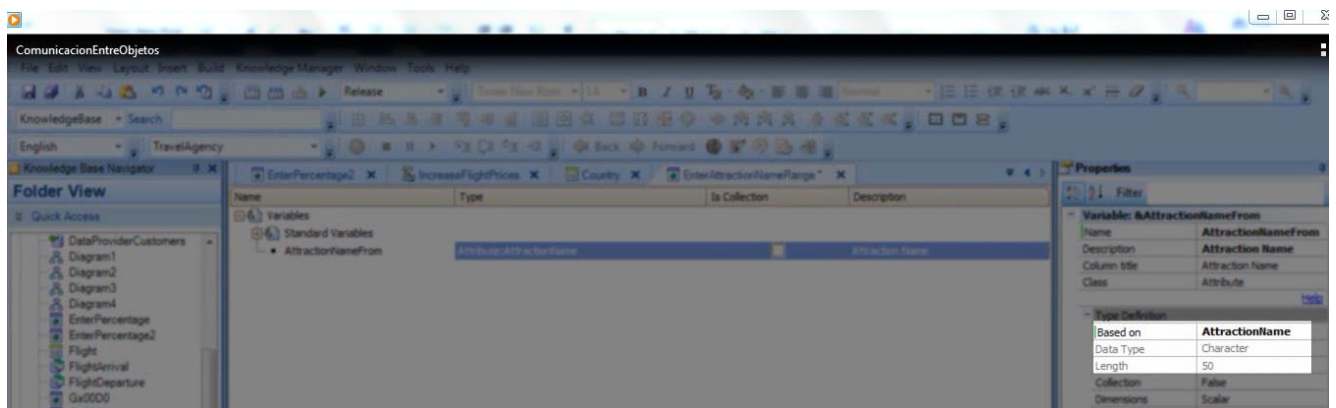
We define the name of the variable: AttractionNameFrom



And the suggestion is to base the definition of the variable on that of the AttractionName attribute.

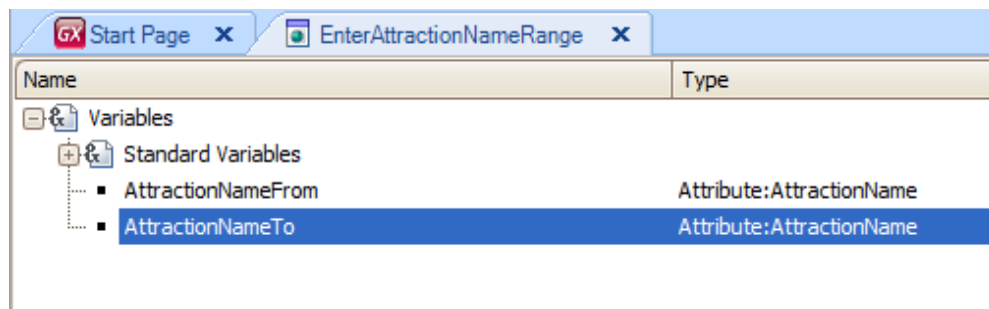


This means that the definition of the variable is related to the definition of the attribute



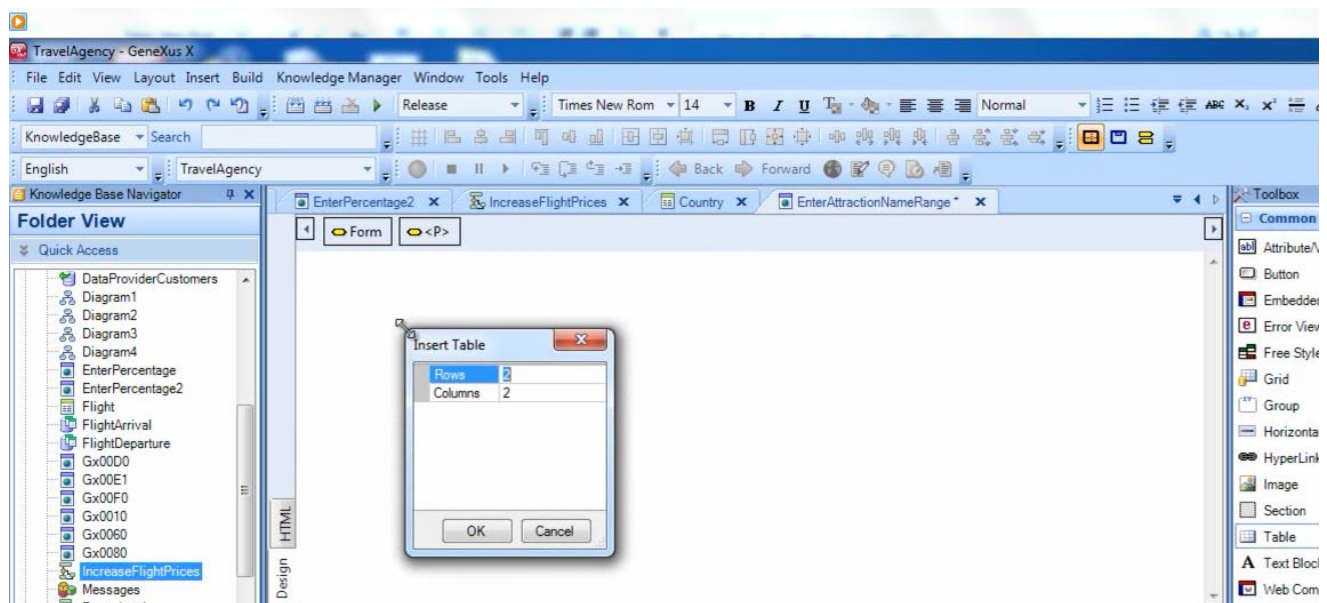
and, if in the future we change the data type of the attribute, the data type of the variable will change accordingly and automatically.

We will define one more variable named: AttractionNameTo, of the same type:



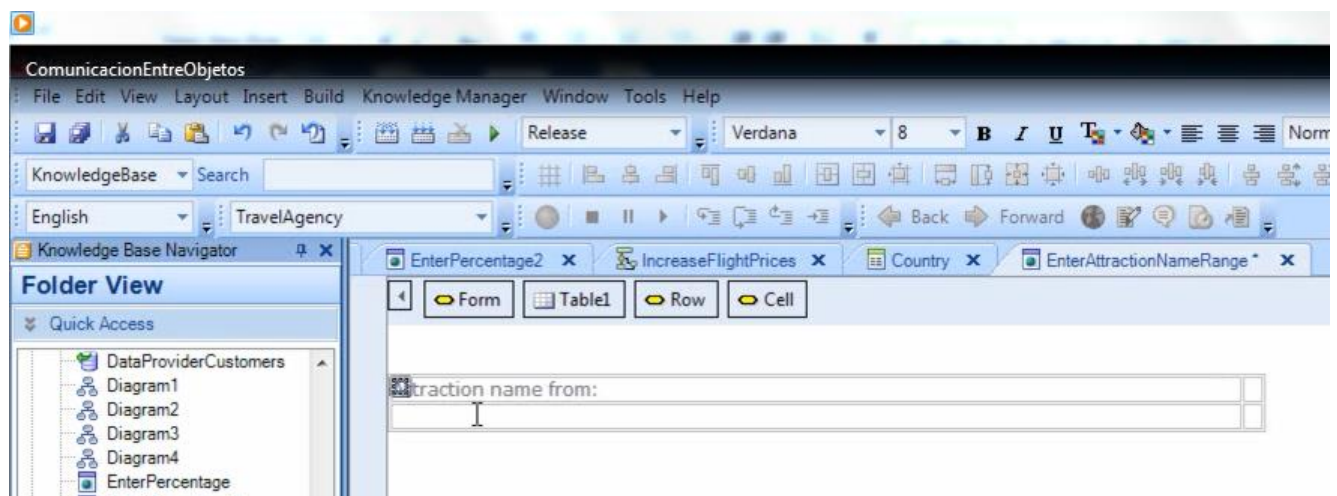
Now we go to the form.

From the ToolBox we insert a table and maintain the default size.

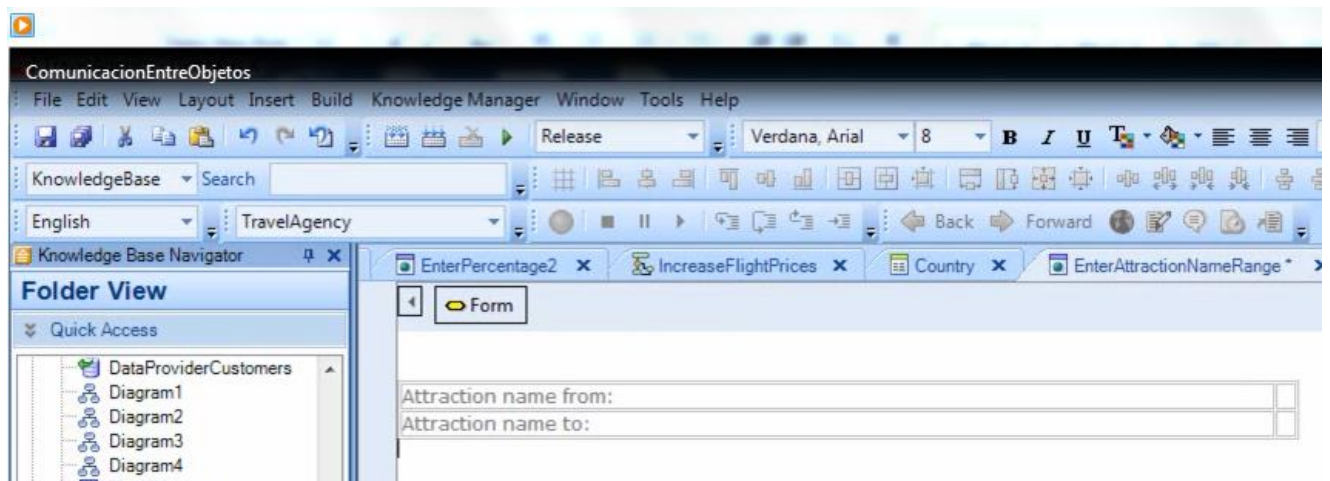


In web design it is important to use tables so that what we write or insert in the screen is duly aligned.

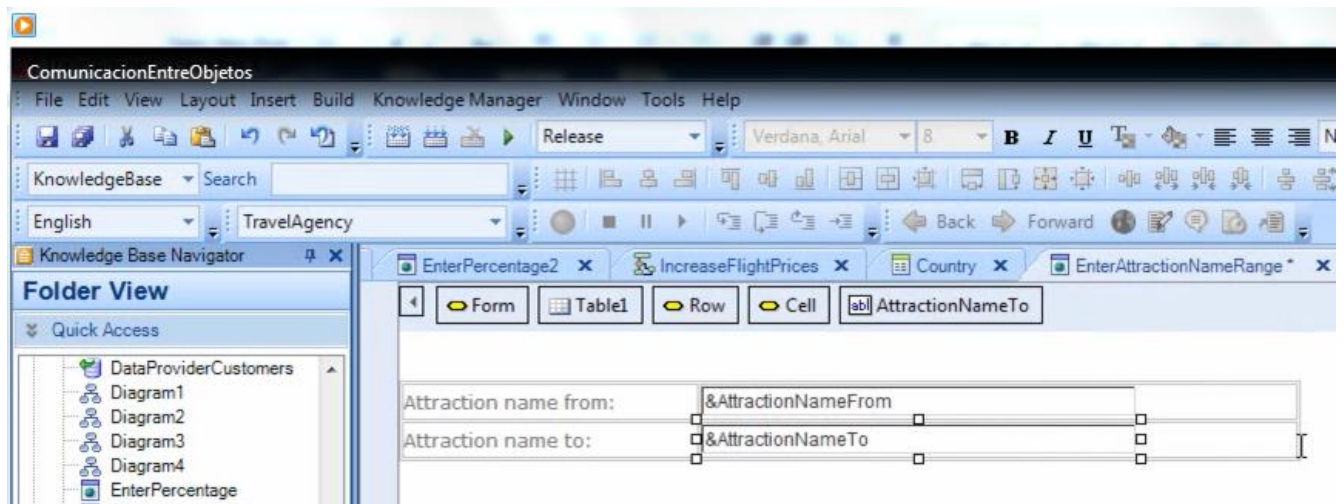
In this cell we type “Attraction name from:”



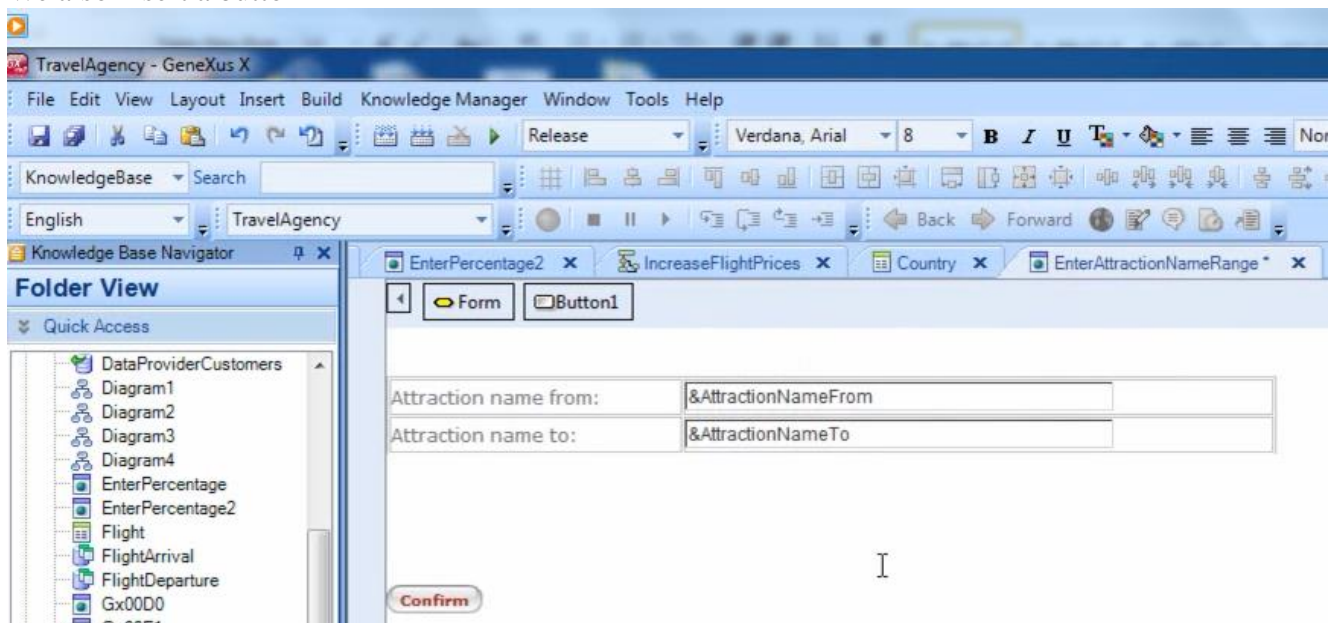
And “Attraction name to:” in this other cell



And we enter the variables we have just defined.

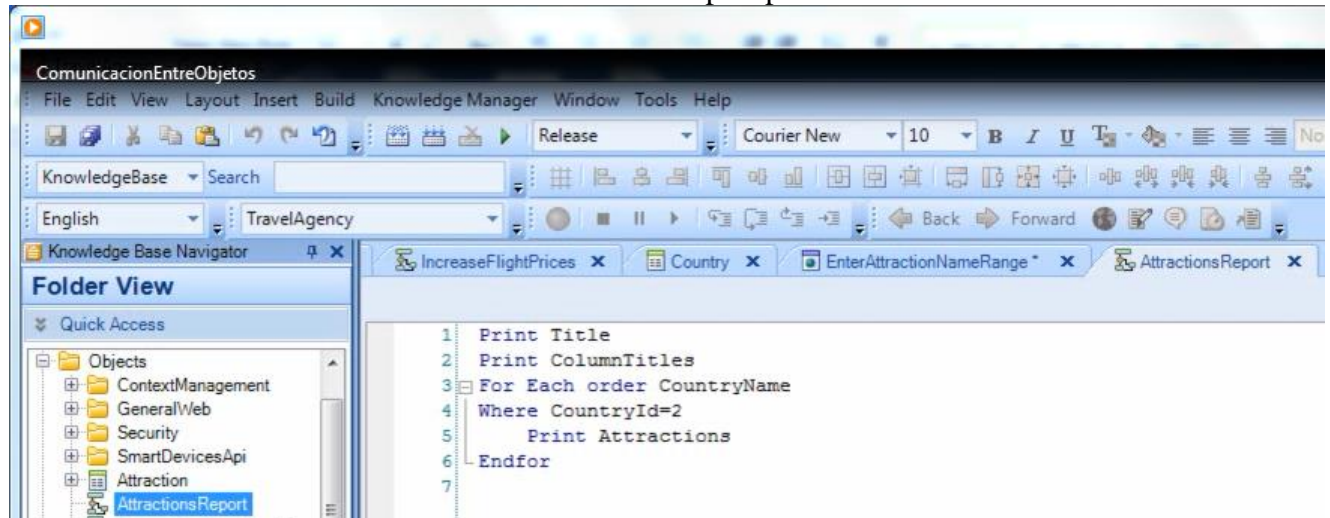


We also insert a button

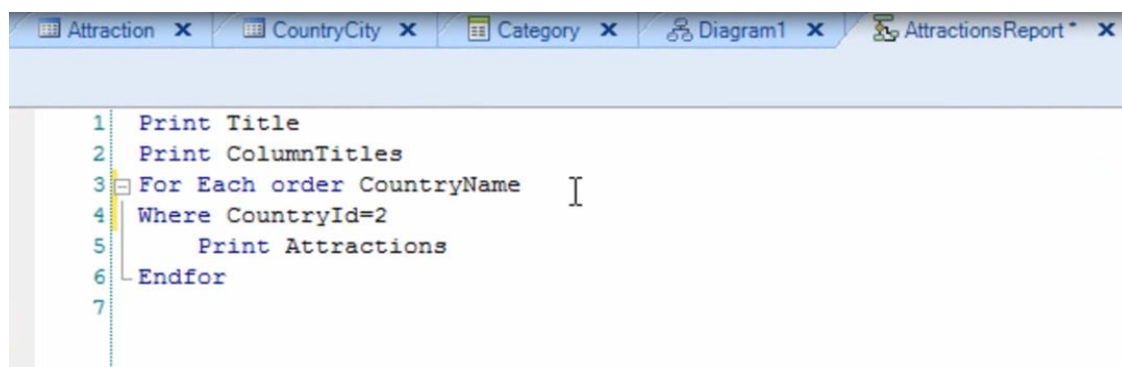


and for a moment we leave the web panel as it is, to now solve the procedure we will be calling in the event associated with the button.

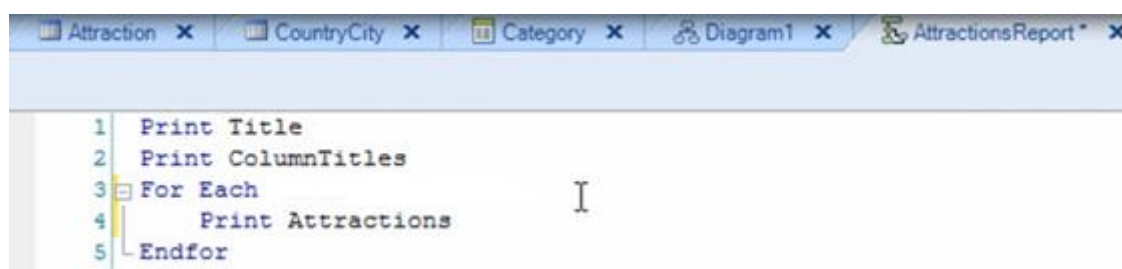
In the Folder View window we have the AttractionsReport procedure



So we right-click and select **Save As ...** we call it “AttractionsReport2” and get a copy where we will do some changes.

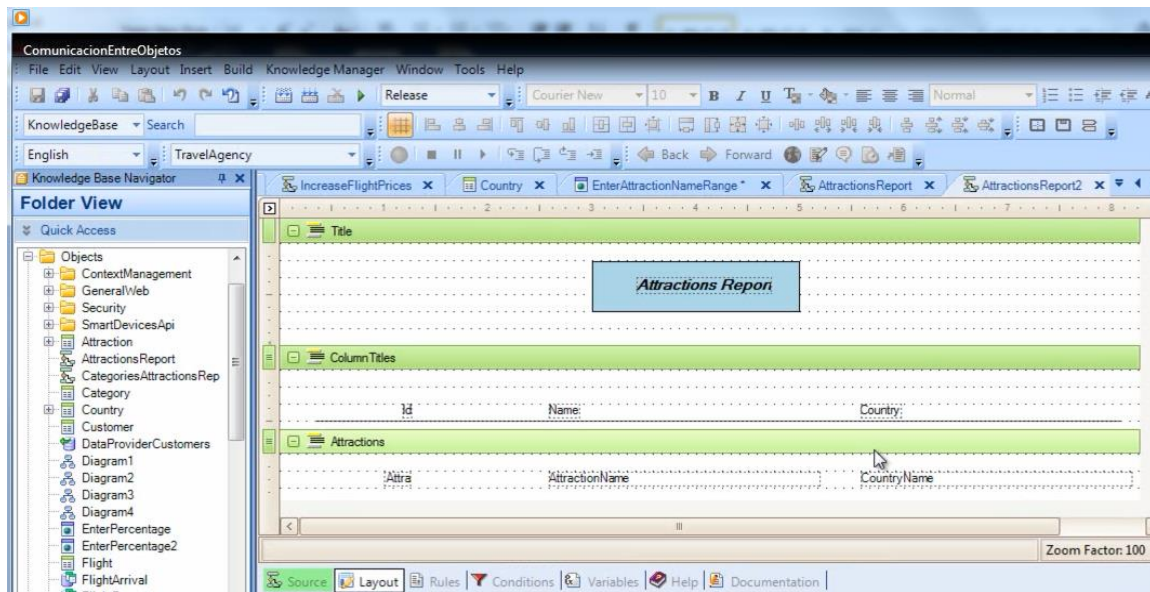


We delete the order and the where.

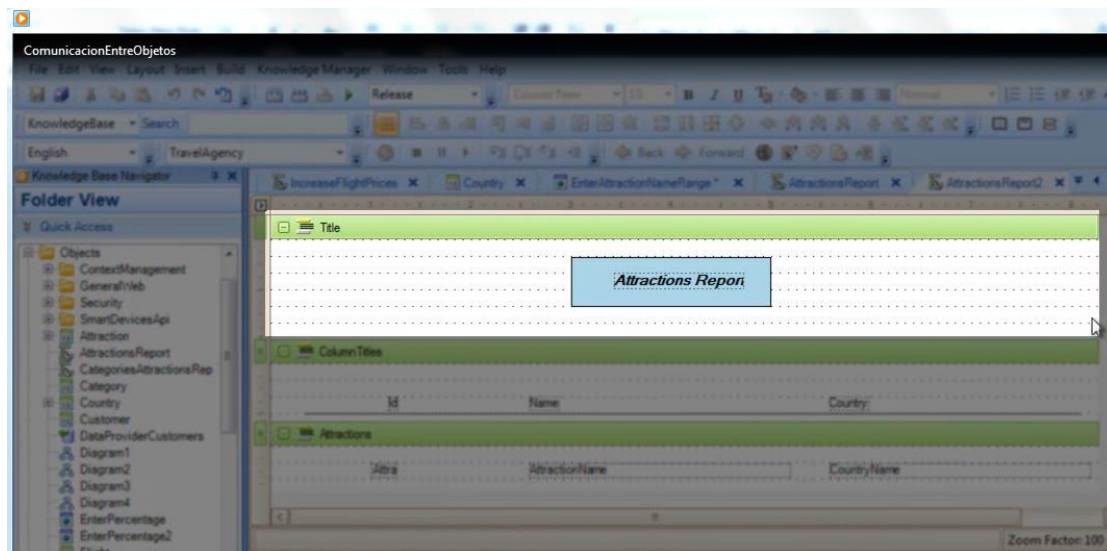


This way the list will print all the attractions.

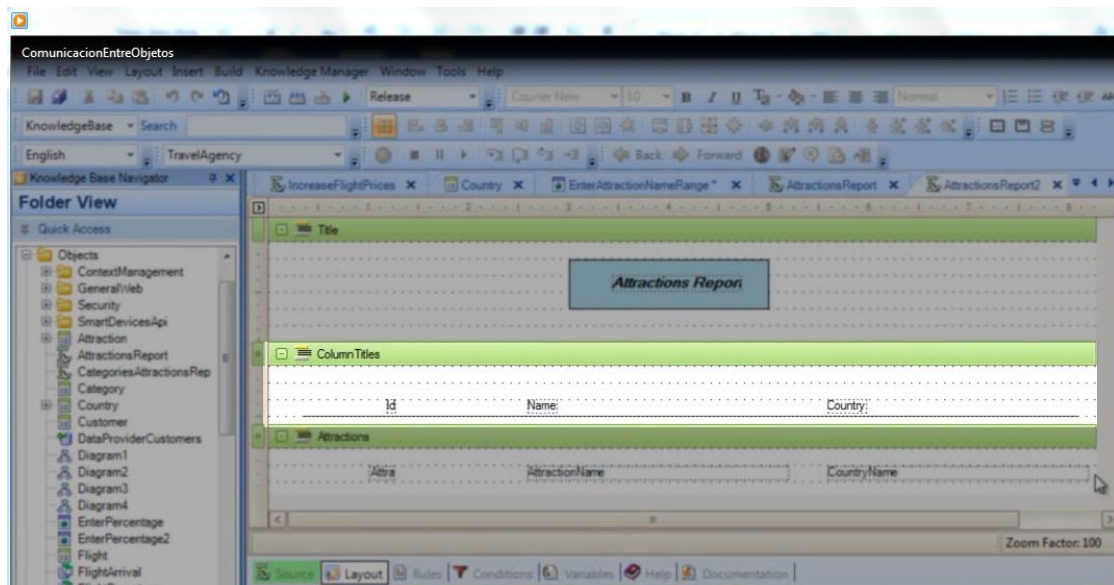
Let's now see the layout



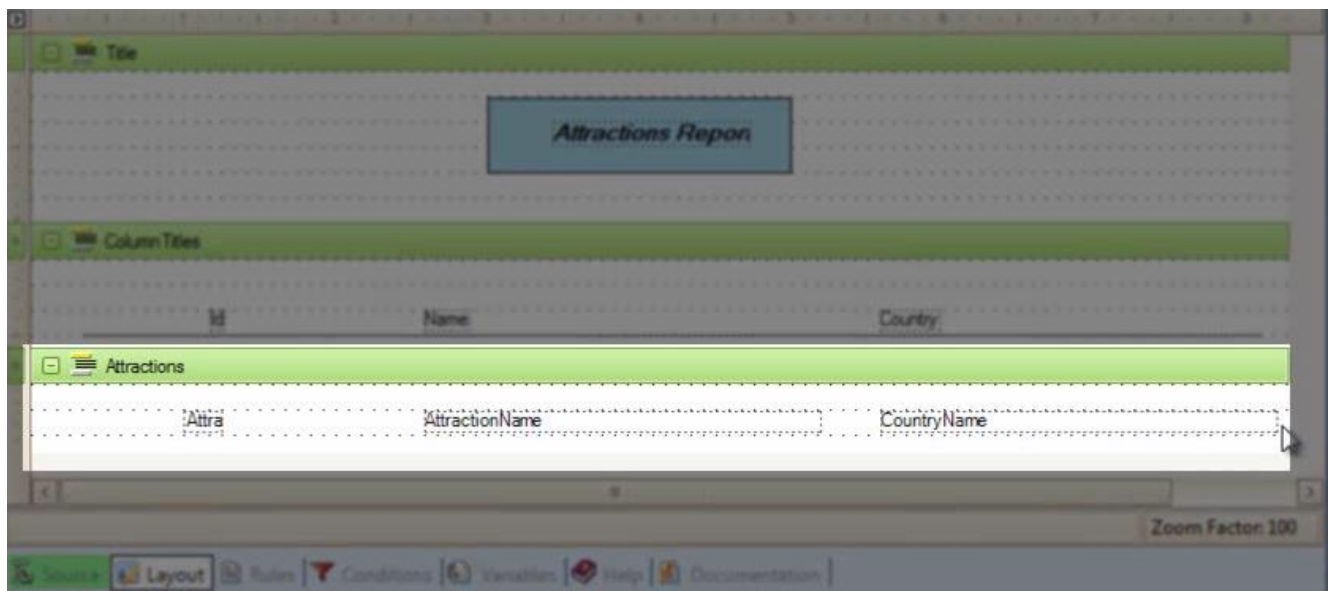
It contains 3 printblocks... this one prints the title of the report



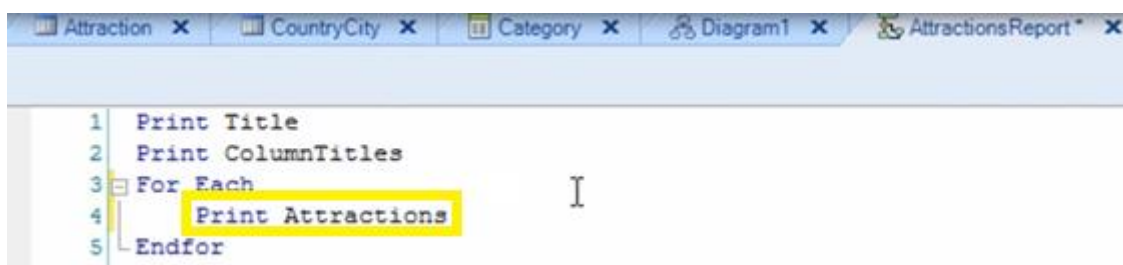
This one prints the titles of columns



And this is the only one containing attributes, which will print inside the For each.



Let's go back to the source, where we will see that the For each command contains only the invocation to the Attractions printblock



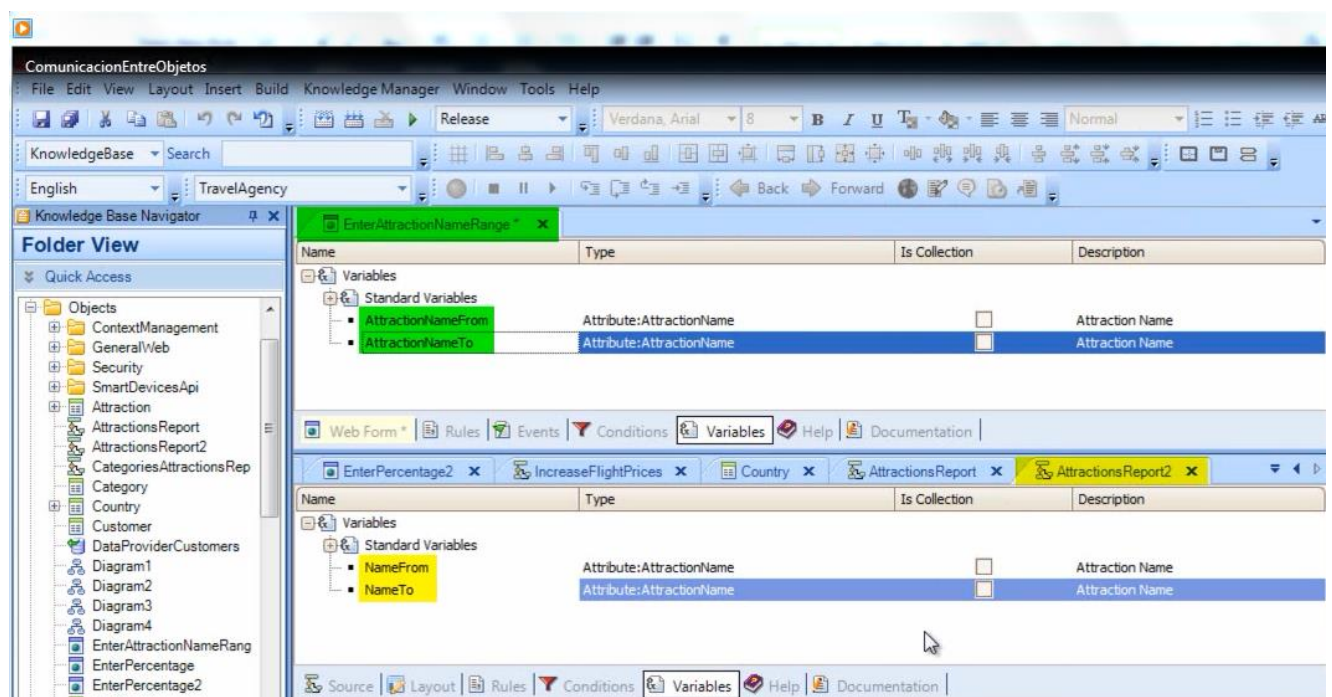
We now define that this procedure will receive the start and end range of names of attractions, and we will use the data received to filter in the For each.

We go now to the variables section of the procedure and define two variables:

- NameFrom, based on the attribute AttractionName
- And NameTo, also based on the AttractionName attribute

| Name | Type |
|--------------------|--------------------------|
| Variables | |
| Standard Variables | |
| NameFrom | Attribute:AttractionName |
| NameTo | Attribute:AttractionName |

Note that we have named the variables differently regarding the names defined in the web panel.



What is important is that the **data types** sent and received match!!

We now go to the Rules section in the procedure and write: `parm(&NameFrom,&NameTo);`

```

1 parm(&NameFrom,&NameTo);
2 output_file('AttractionsReport2.pdf','pdf');
3

```

These variables we receive in the procedure will be used to filter in the For each. So we go back to the source and write ... where AttractionName>=&NameFrom.. enter.. and where AttractionName>=&NameTo


```

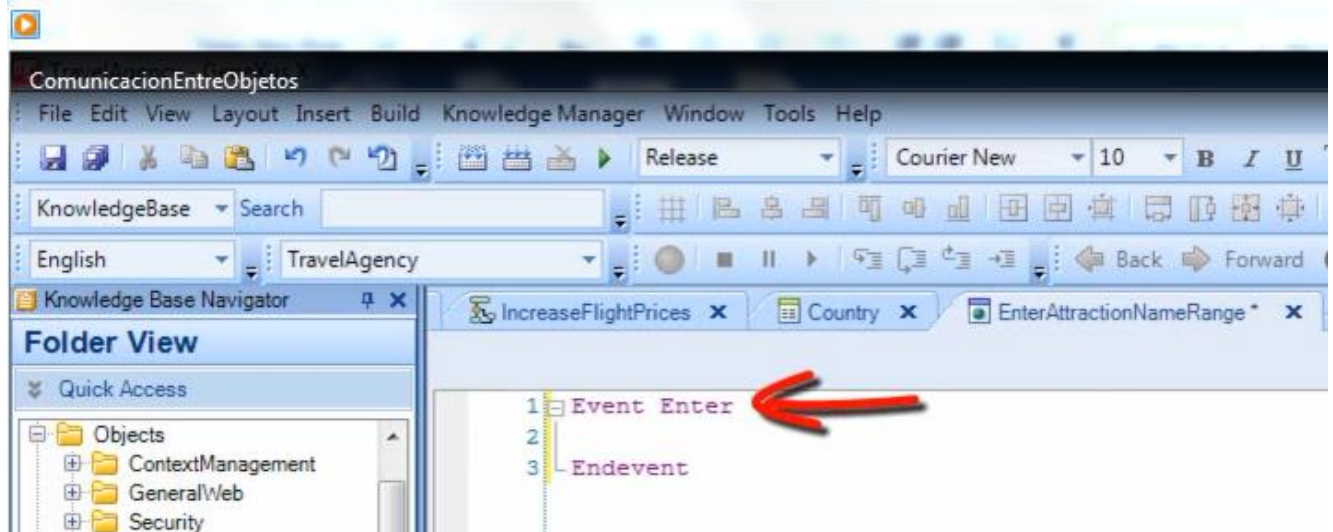
1 Print Title
2 Print ColumnTitle
3 for each
4     where AttractionName >= &NameFrom
5     where AttractionName <= &NameTo
6     print Attractions
7 -endfor

```

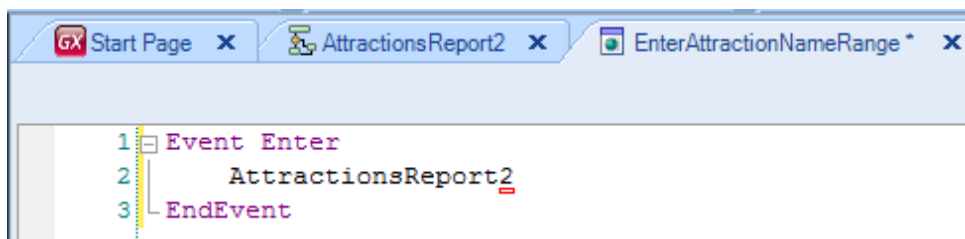
Because the procedure was saved upon a previous one, the properties and the rule necessary to print in PDF format are already set.

With this, the procedure is ready, and all we need is to call it from the web panel.

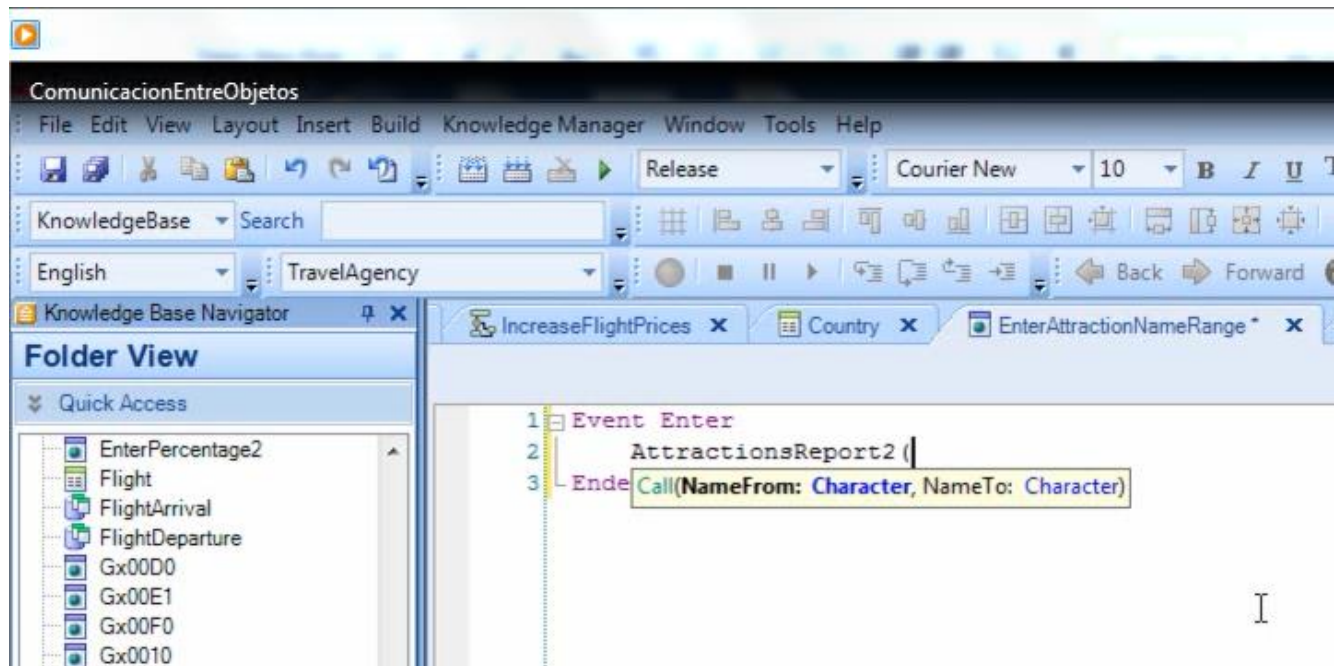
We go to the web panel and double-click on the button to see the associated event.



We then insert the name of the procedure, AttractionsReport2

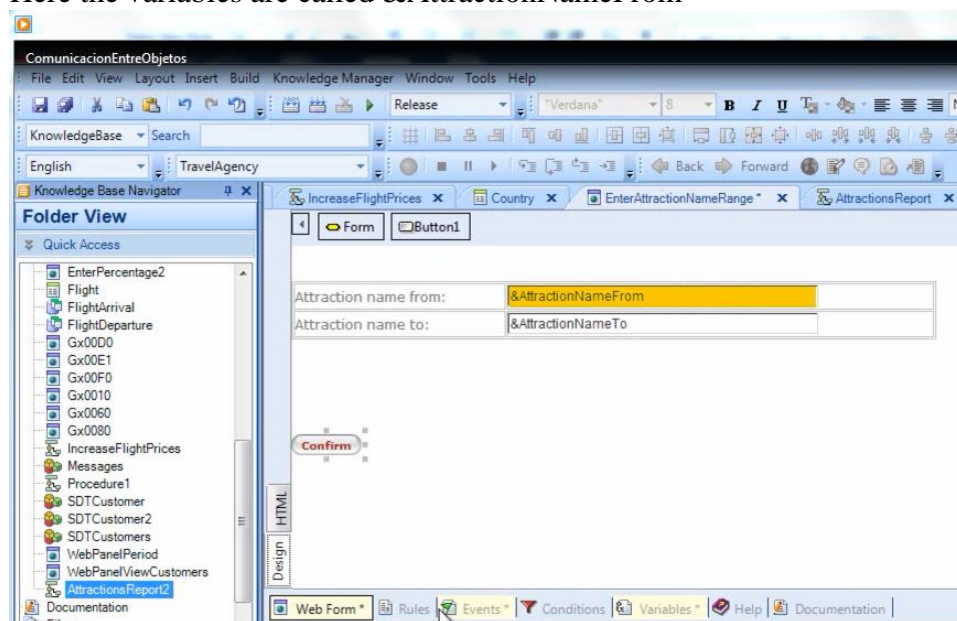


And next to the procedure name we add parentheses,

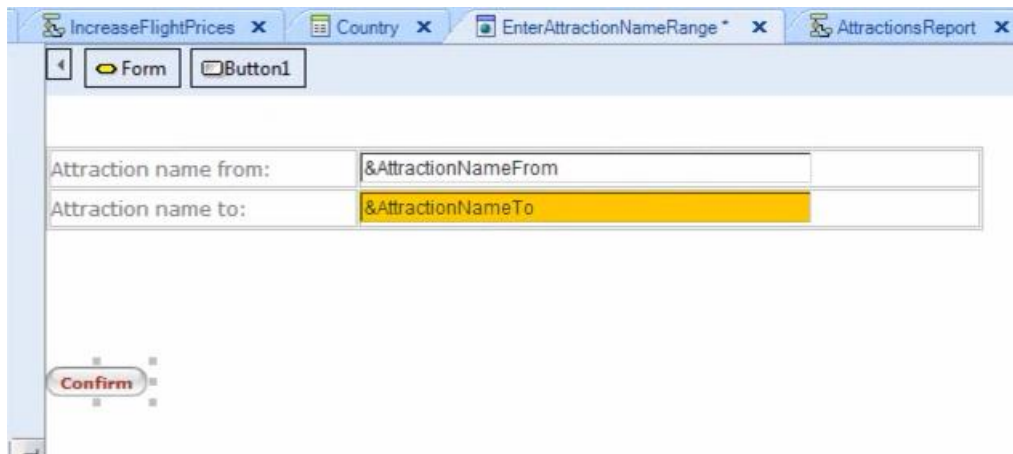


inside which we will include the data we want to send, separated by a comma.

Here the variables are called &AttractionNameFrom



and &AttractionNameTo



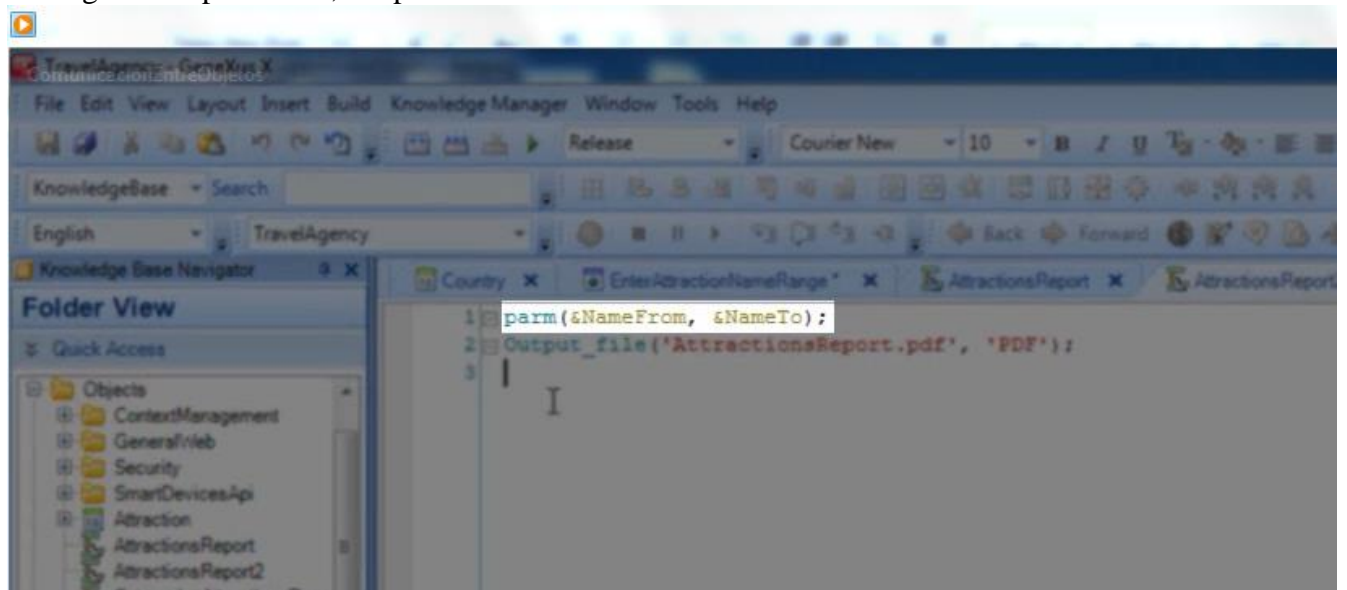
So we will send them with these names.

Inside the parentheses we type ampersand and select &AttractionNameFrom... then type a comma and ampersand again and select &AttractionNameTo.

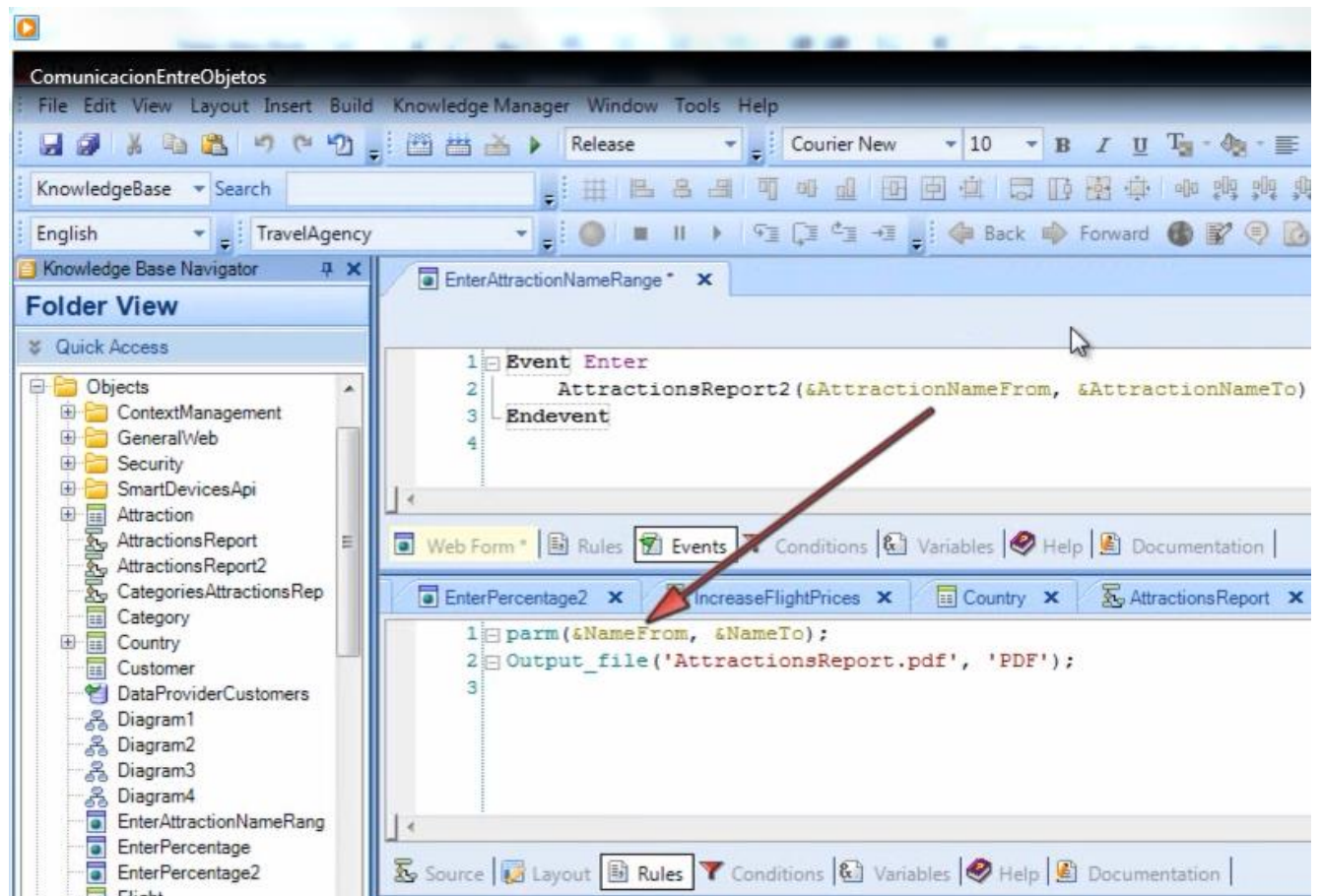


The call to the procedure is now complete, where we have sent the range of names of attractions that the user typed in the form of these two variables.

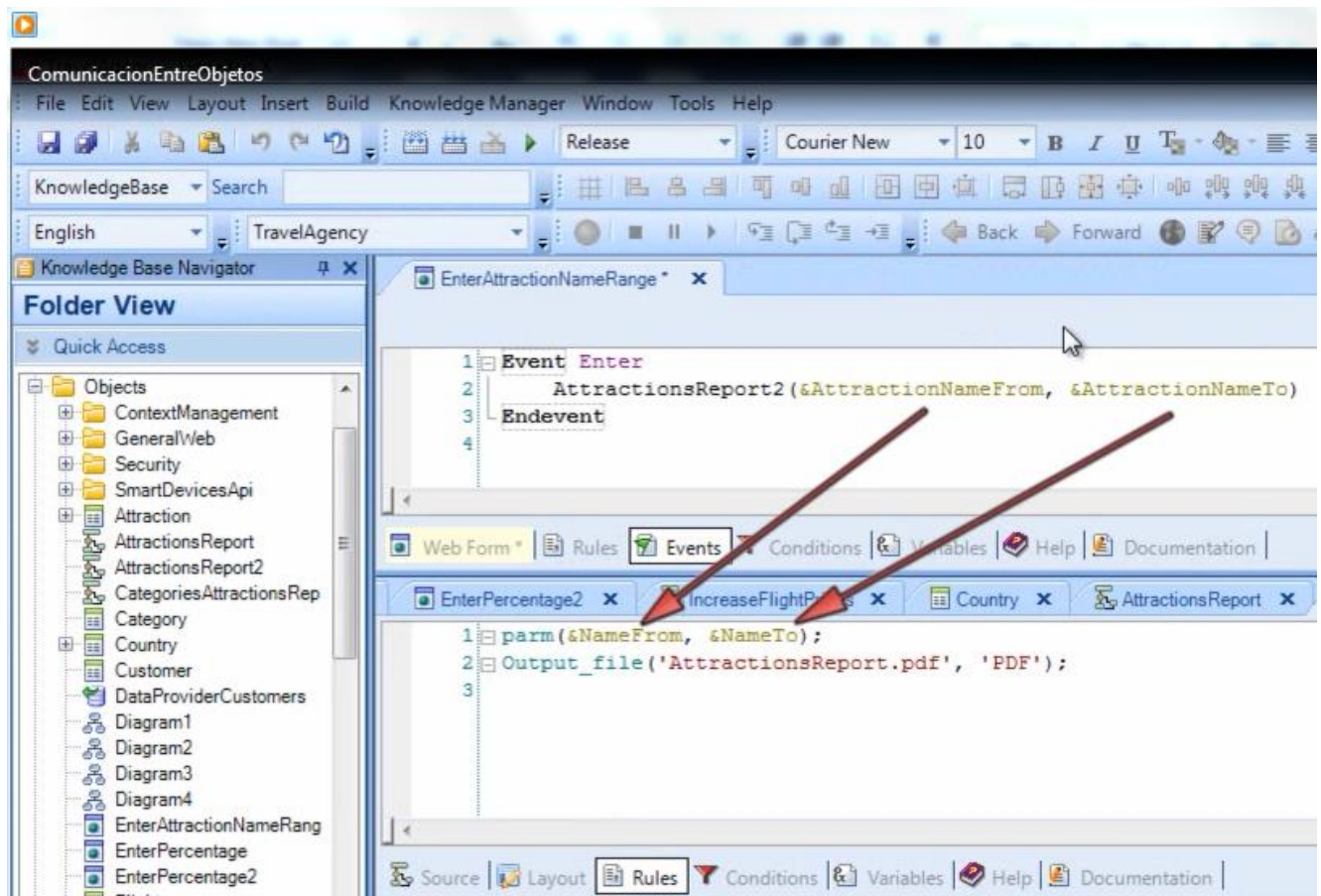
If we go to the procedure, the parm rule receives two variables:



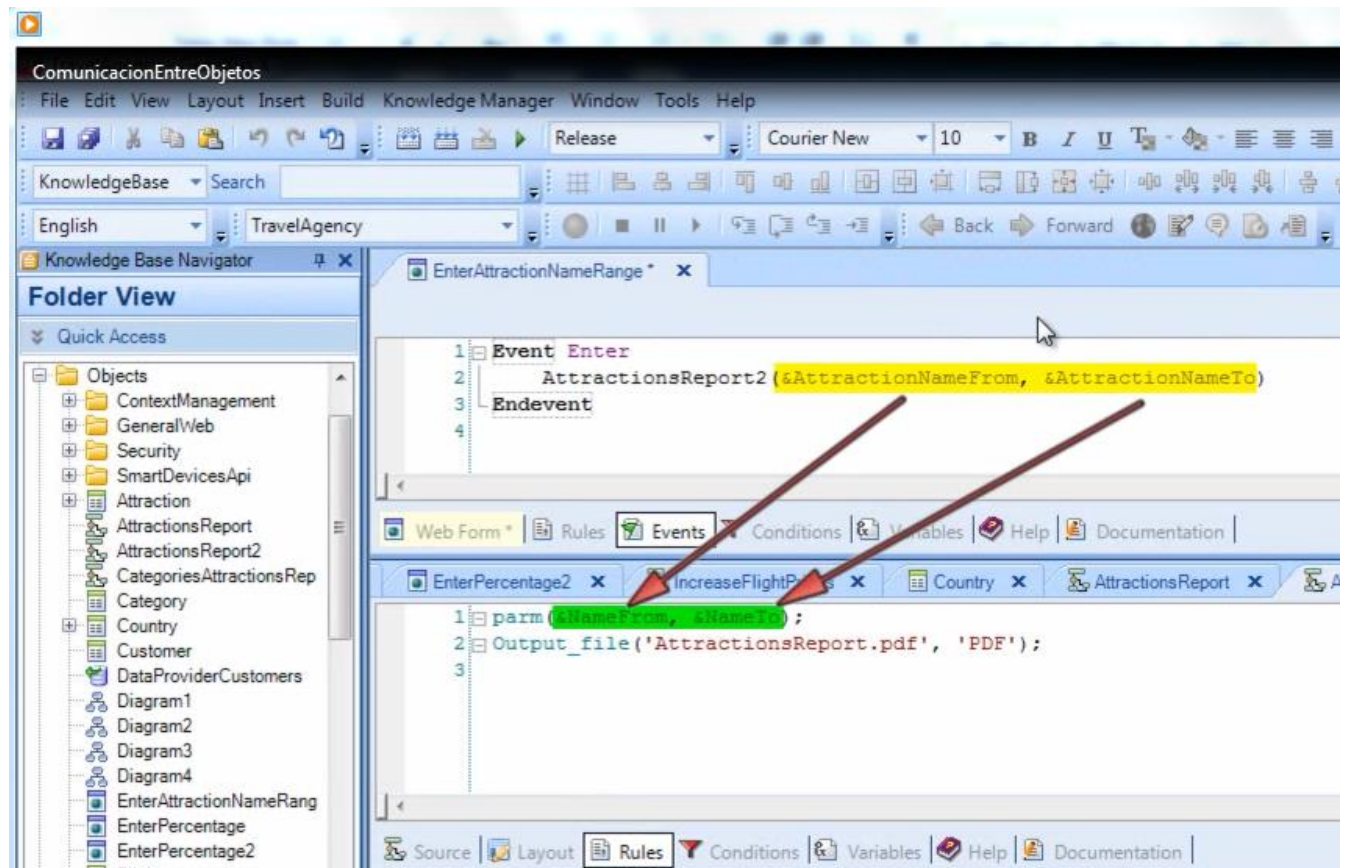
The data sent in the first place comes first



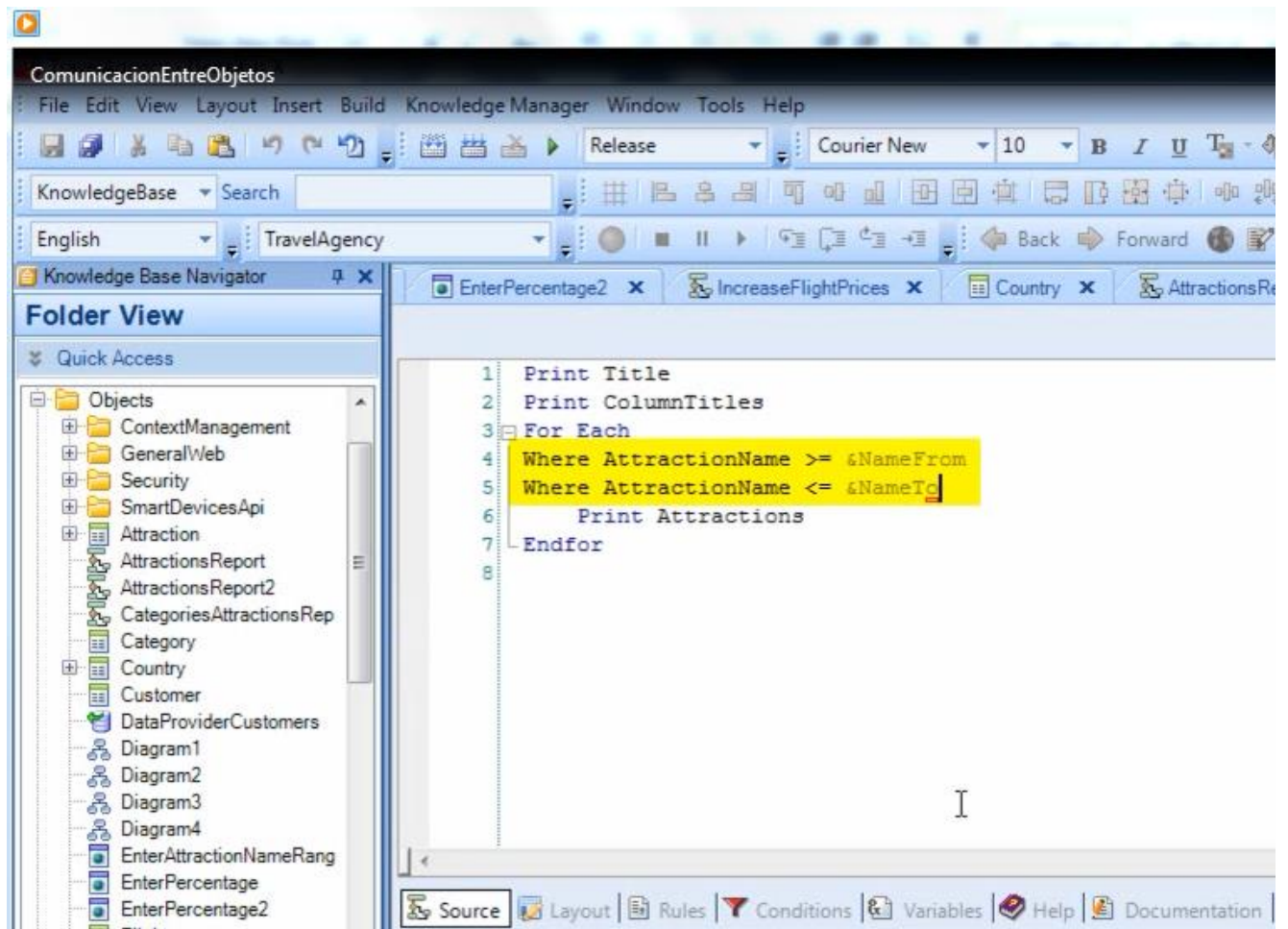
... and then follows the other data as it arrives in the object called, in the order it was sent.



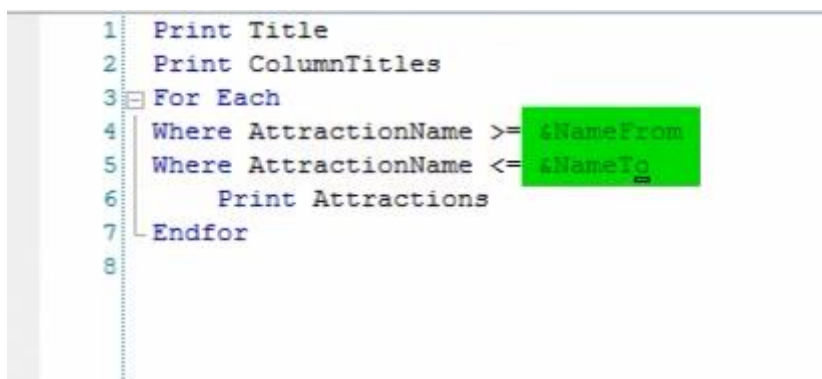
We must insist on the fact that it isn't the name of variables that matters, but rather their order. It is always convenient to use related names as we did here so as to better understand the code.



We use the values received in the variables to filter in the For Each.



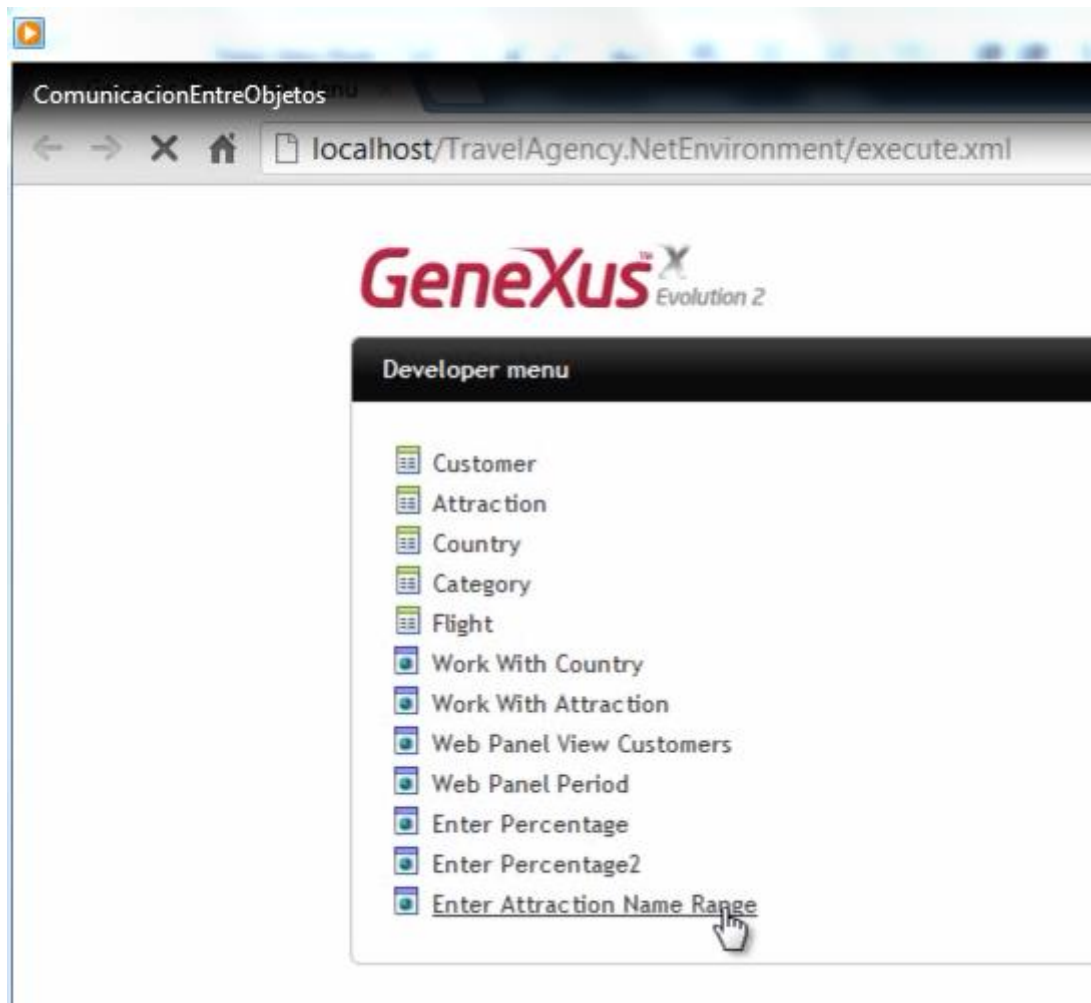
Note that we use the names of the variables defined **in this procedure**



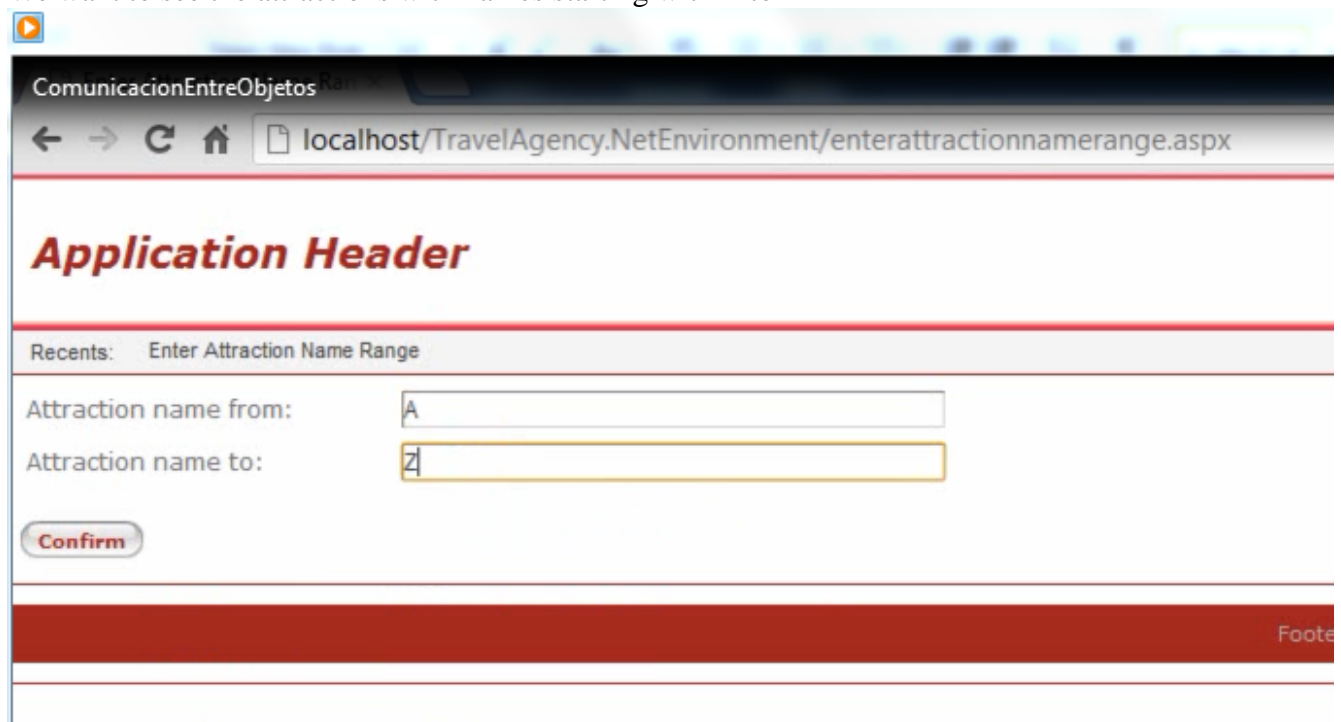
and not those used in the calling object.

Let's see the functioning of all this we have done and explained. Press F5

We execute the web panel "EnterAttractionNameRange".



We want to see the attractions with names starting with A to Z



We press the button ... and we see that all attractions have been listed.

Attractions Report

| Id | Name | Country |
|----|---------------------------|---------------|
| 1 | Louvre Museum | France |
| 2 | Great Wall | China |
| 3 | Eiffel Tower | France |
| 4 | The Christ Redeemer | Brazil |
| 5 | The Smithsonian Institute | United States |
| 6 | Egypt Pyramides | Egypt |

We now reduce the range a little, from A to F

ComunicacionEntreObjetos

localhost/TravelAgency.NetEnvironment/enterattractionnamerange.aspx

Application Header

Recents: Enter Attraction Name Range

Attraction name from:

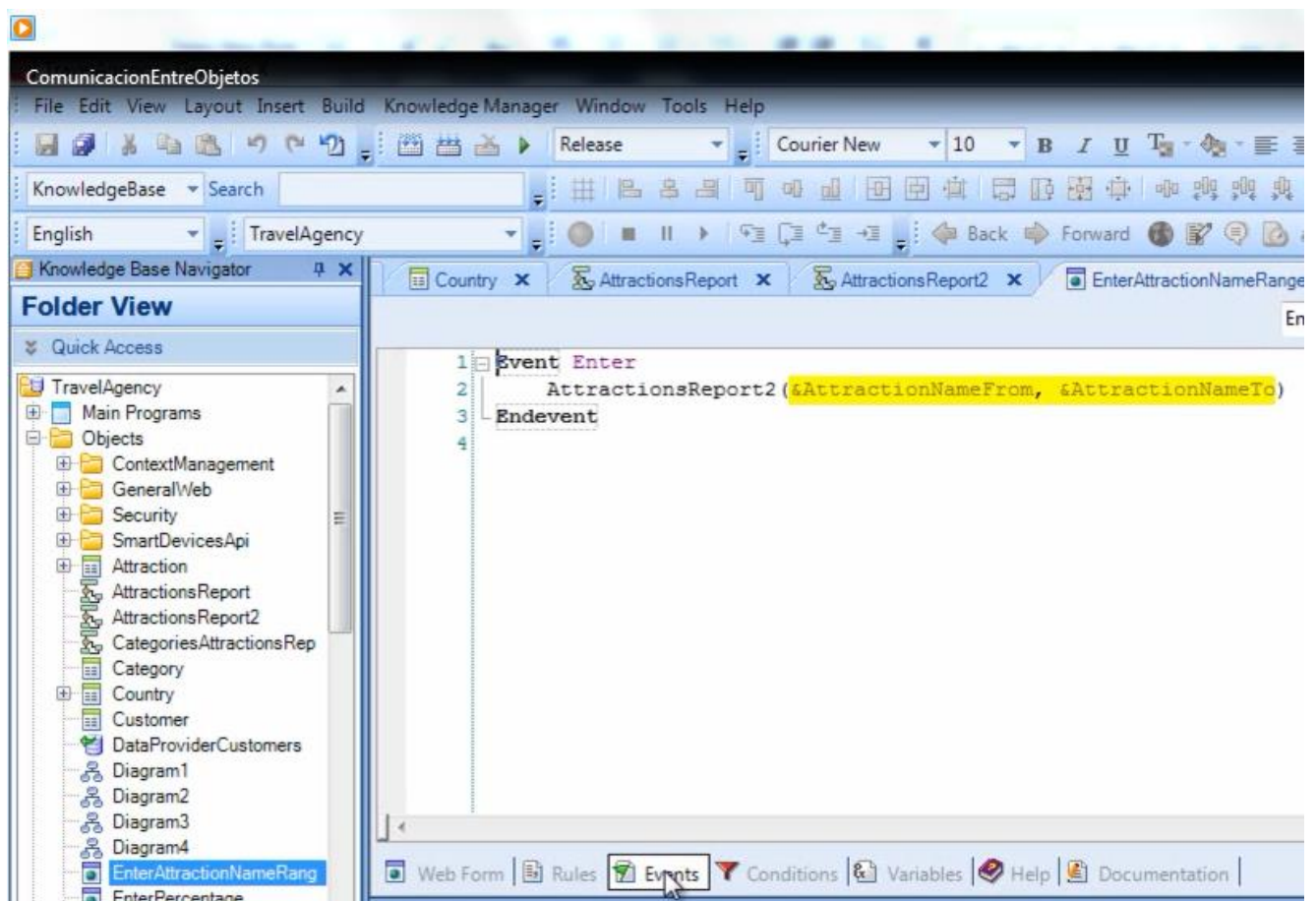
Attraction name to:

and we see that only the Eiffel Tower and the Egyptian Pyramids come up.

Attractions Report

| Id | Name | Country |
|----|----------------|---------|
| 3 | Eiffel Tower | France |
| 6 | Egypt Pyramids | Egypt |

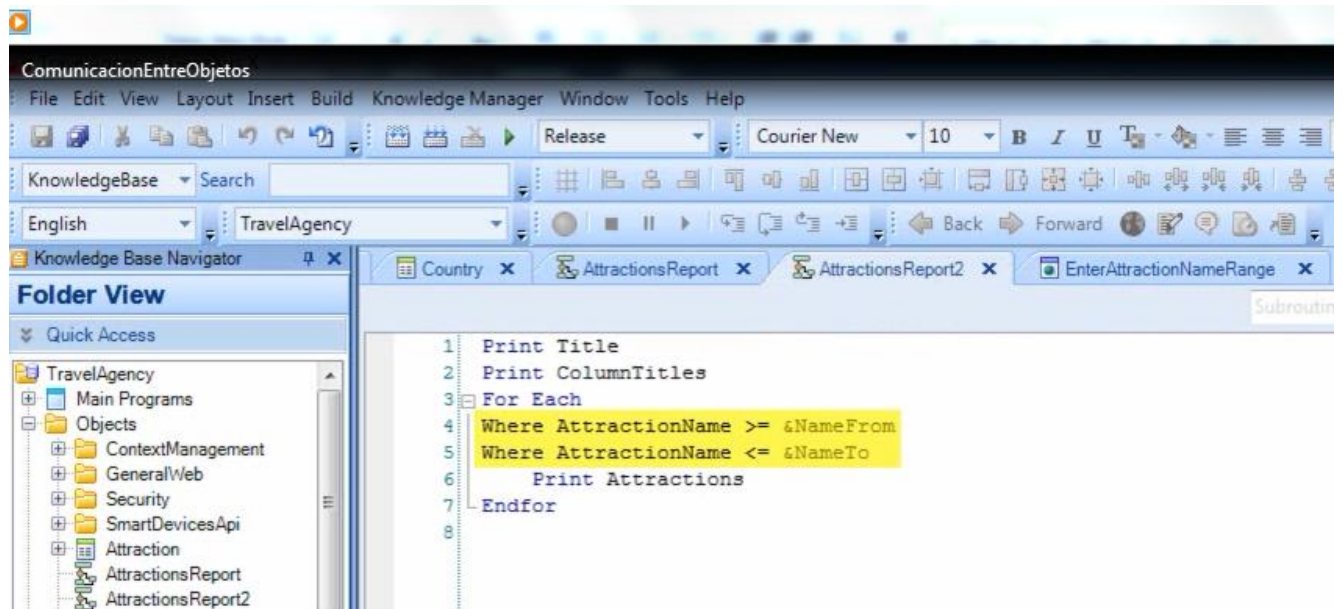
At runtime we have seen a call from one object to another sending two parameters to the object called.



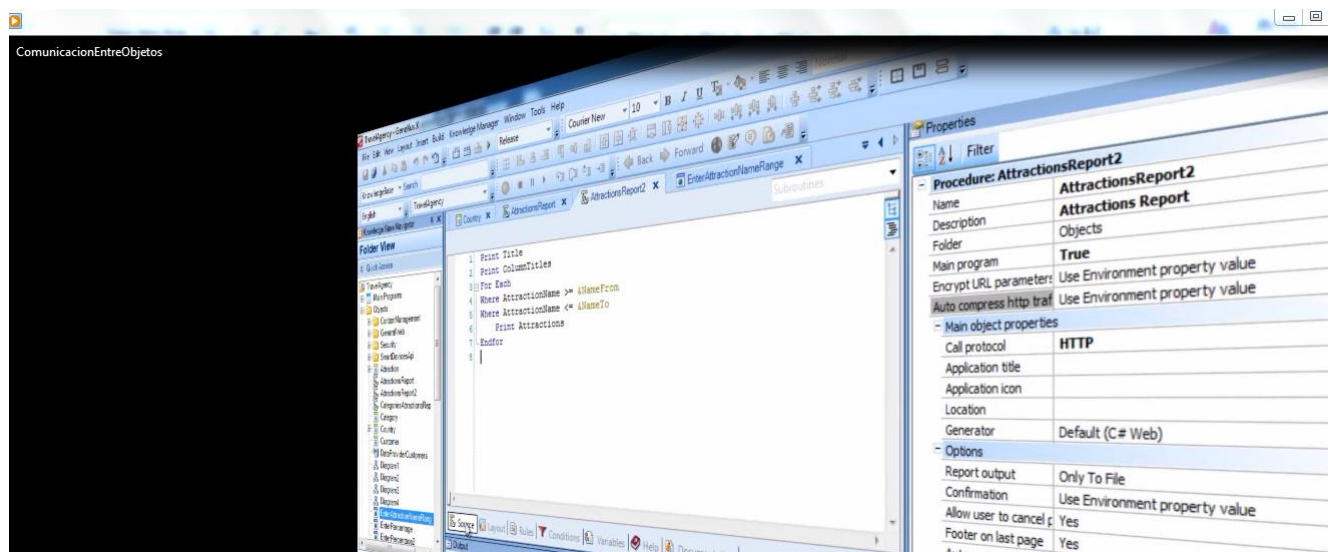
The data received

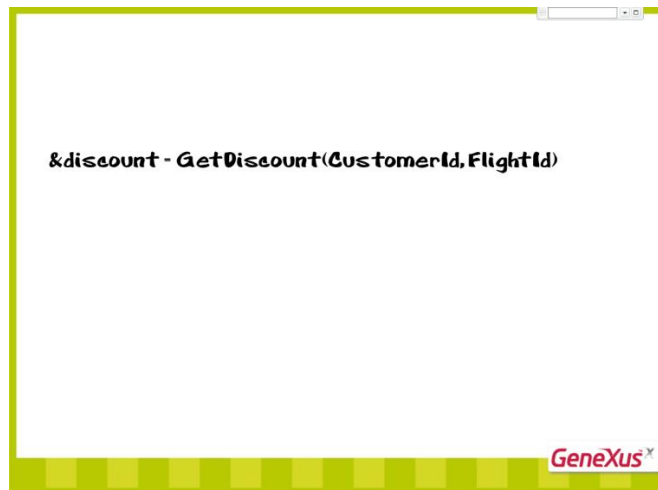

```
Country x AttractionsReport x AttractionsReport2 x EnterAttractionNameRange x
1 parm (&NameFrom, &NameTo);
2 Output_file('AttractionsReport.pdf', 'PDF');
3
```

was used to filter in the For Each command



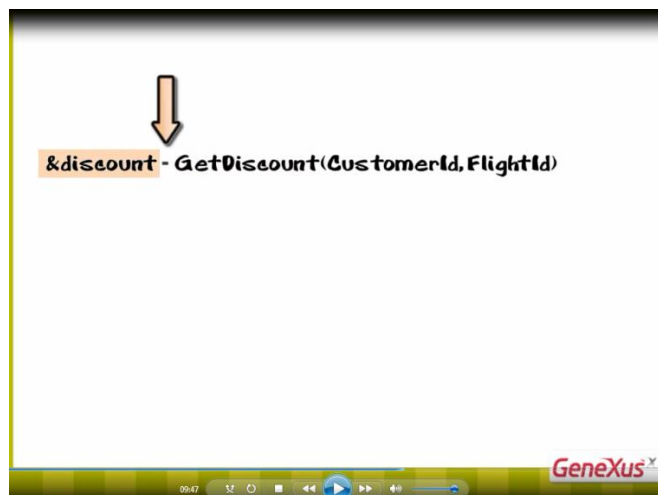
We will now see another possible way to call procedure objects or data providers in particular.



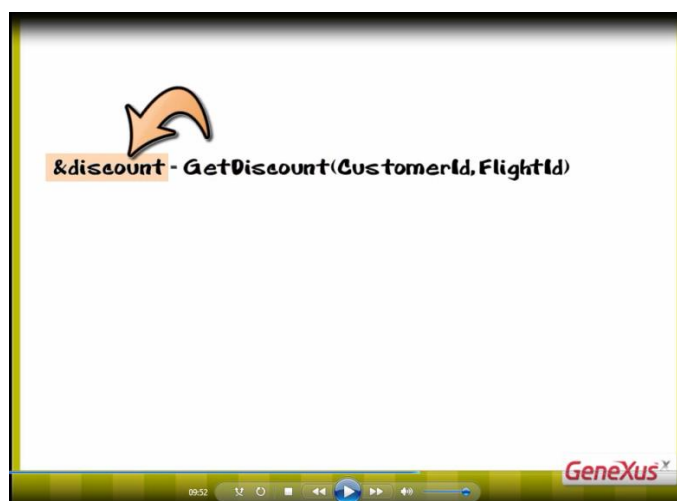


In this example, we are calling a procedure called GetDiscount.

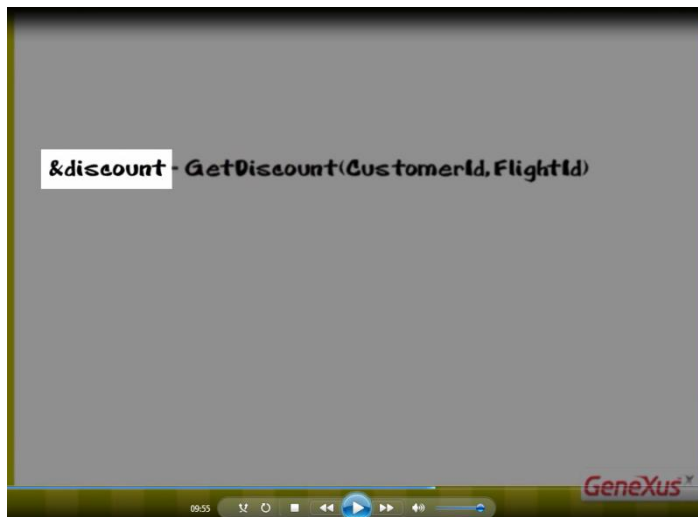
Note that prior to the call to the procedure there is a variable and an equal sign.



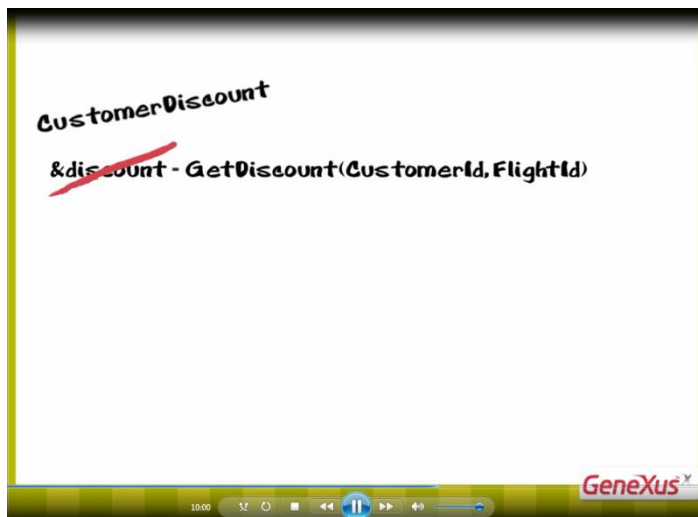
This is because the object called **returns a value**.



To the left of the equal sign

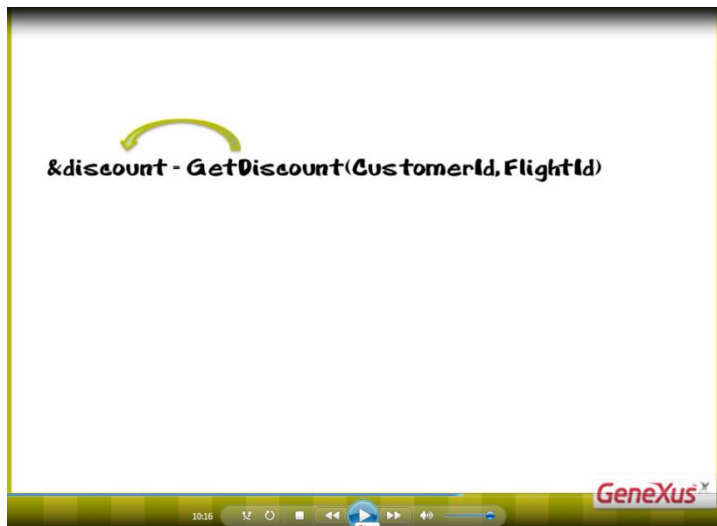


there may be a variable to receive such value or depending on its object and section, there could be an attribute receiving the value.



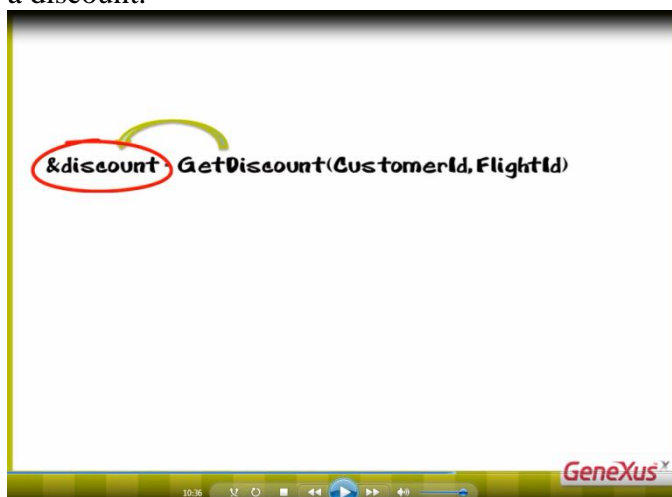
In this case, we are not interested in seeing details of what the GetDiscount procedure does.

Given the name of the procedure, the fact that it returns a value and the name of the variable that receives the value returned by the procedure,



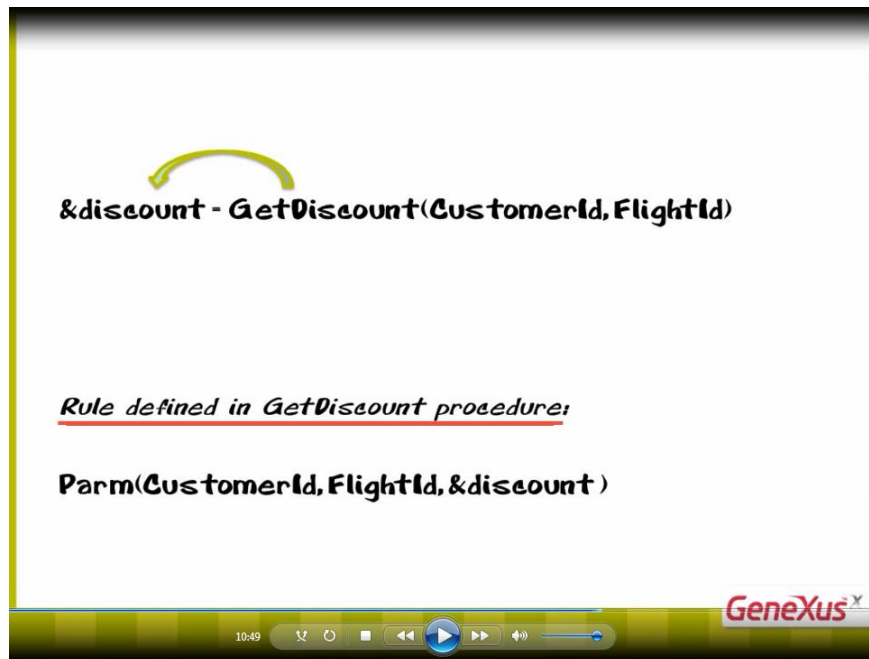
we can conclude that:

- the procedure will return a discount...
- two pieces of data are sent to the procedure: a customer identifier and a flight identifier...
- and the procedure will use that data and assess and calculate whatever is necessary to return a discount.

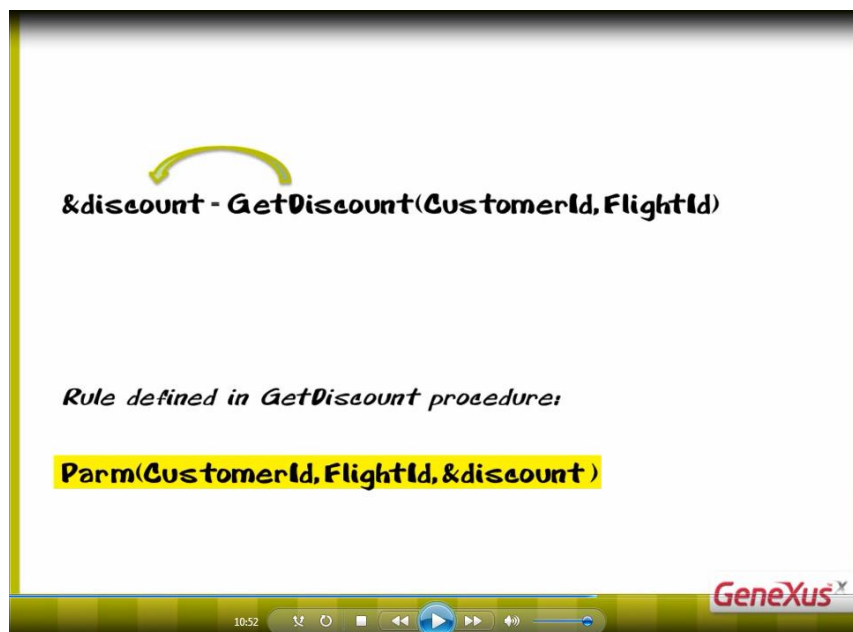


It is now important that we see how the **parm rule** is declared in the object called when the object returns a value in the syntax of the call.

In the rules section of the GetDiscount procedure



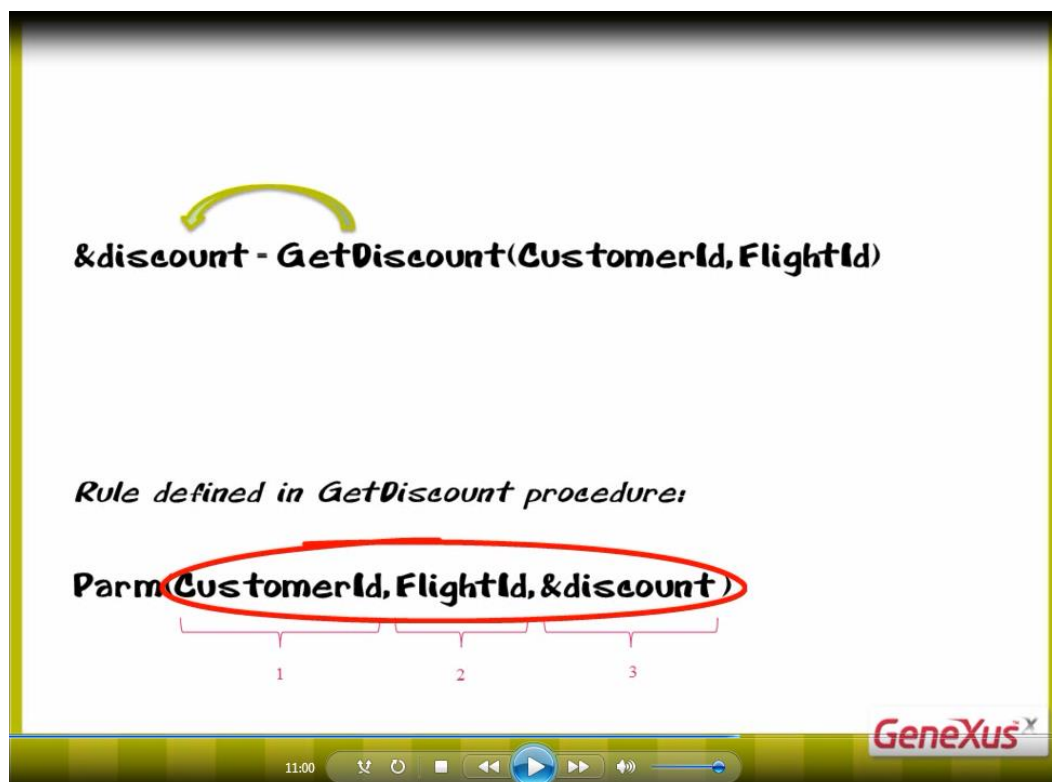
we declare this parm rule



with three parameters.

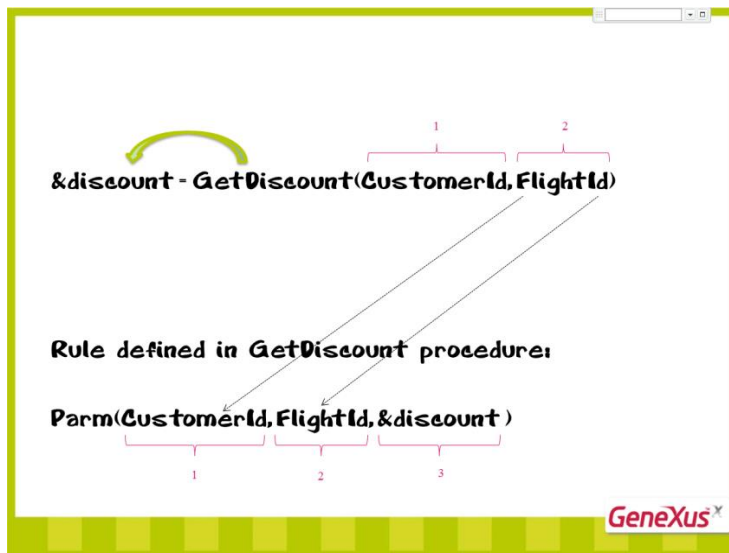


We use the term “parameters” to make reference to data sent and received between two objects where one object calls the other one.



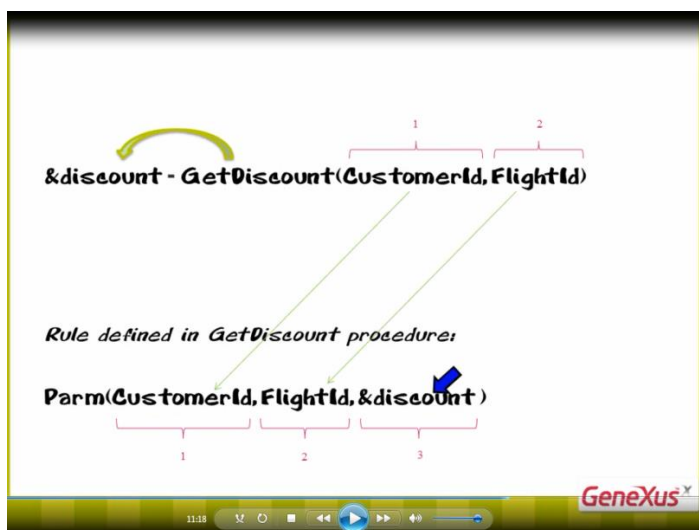
This way we will be able to talk in general, regardless of whether we send or receive variables, attributes or fixed values.

Let's now see how this works: the two parameters sent,

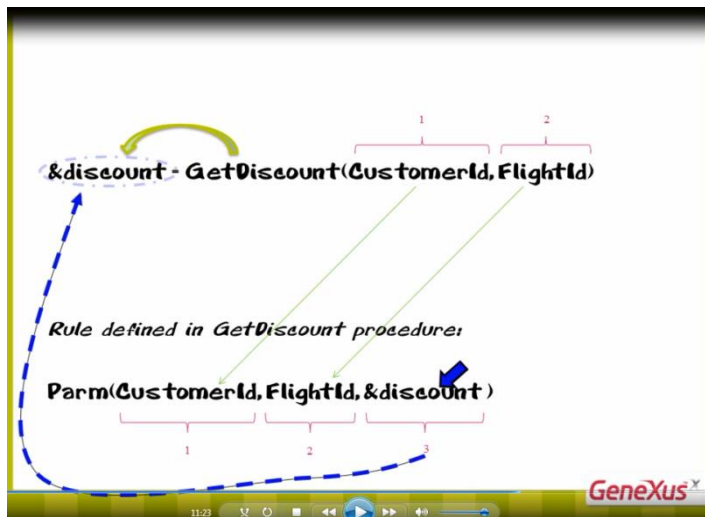


are received in order.

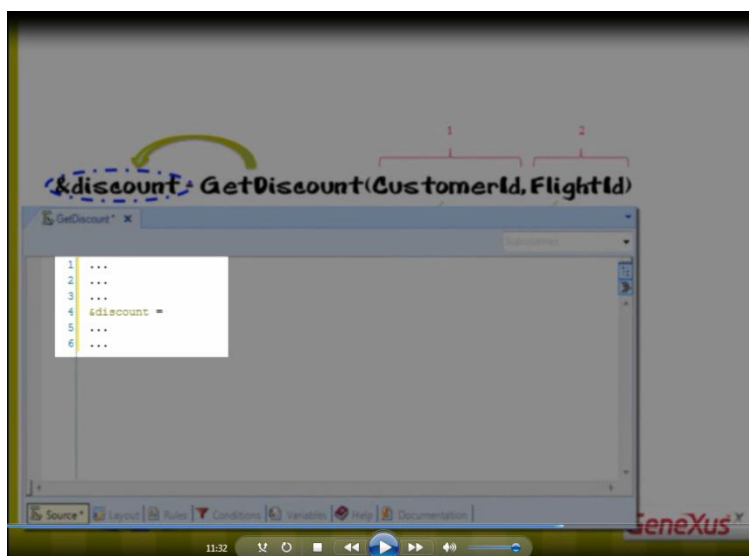
And the third parameter defined in the parm rule



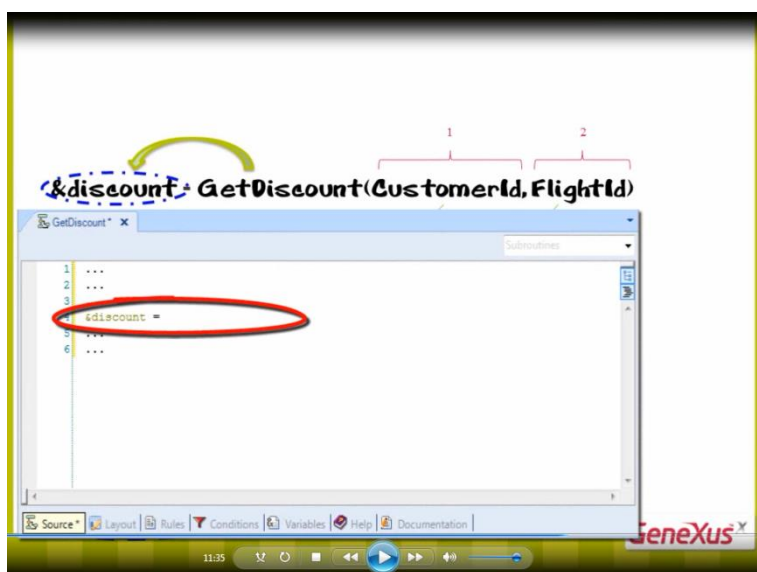
corresponds to the one that stores the value returned, in the `&discount` variable of the invocation.



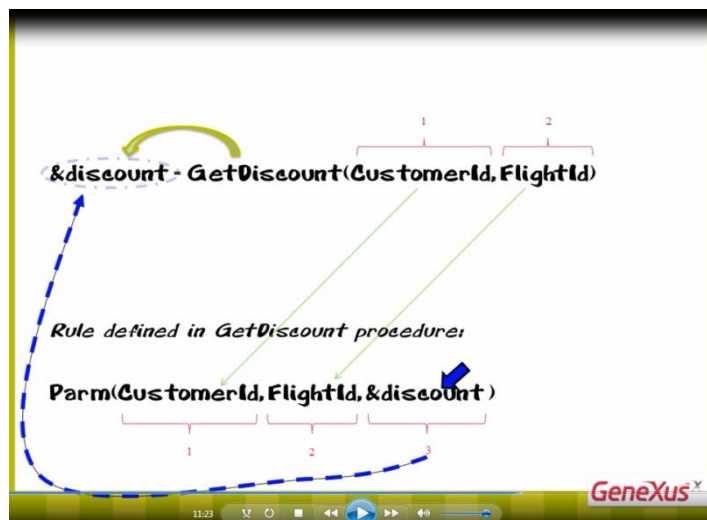
So, in the source of the GetDiscount object, in some place in the group of instructions



we will have to assign a value to the &discount variable,



for that value to be returned and assigned to the variable located to the left of the equal sign.



We have previously called a Data Provider object like this.

```
SDTCustomer x Procedure1 x SDTCustomer2 x DataProviderCustomers x WebPanelViewCustomers
Start
1 Event Start
2 &Customers = DataProviderCustomers()
3 EndEvent
```

This statement

ComunicacionEntreObjetos

File Edit View Layout Insert Build Knowledge Manager Window Tools Help

Release Courier New 10 B I U T

KnowledgeBase Search

English TravelAgency

Knowledge Base Navigator

Folder View

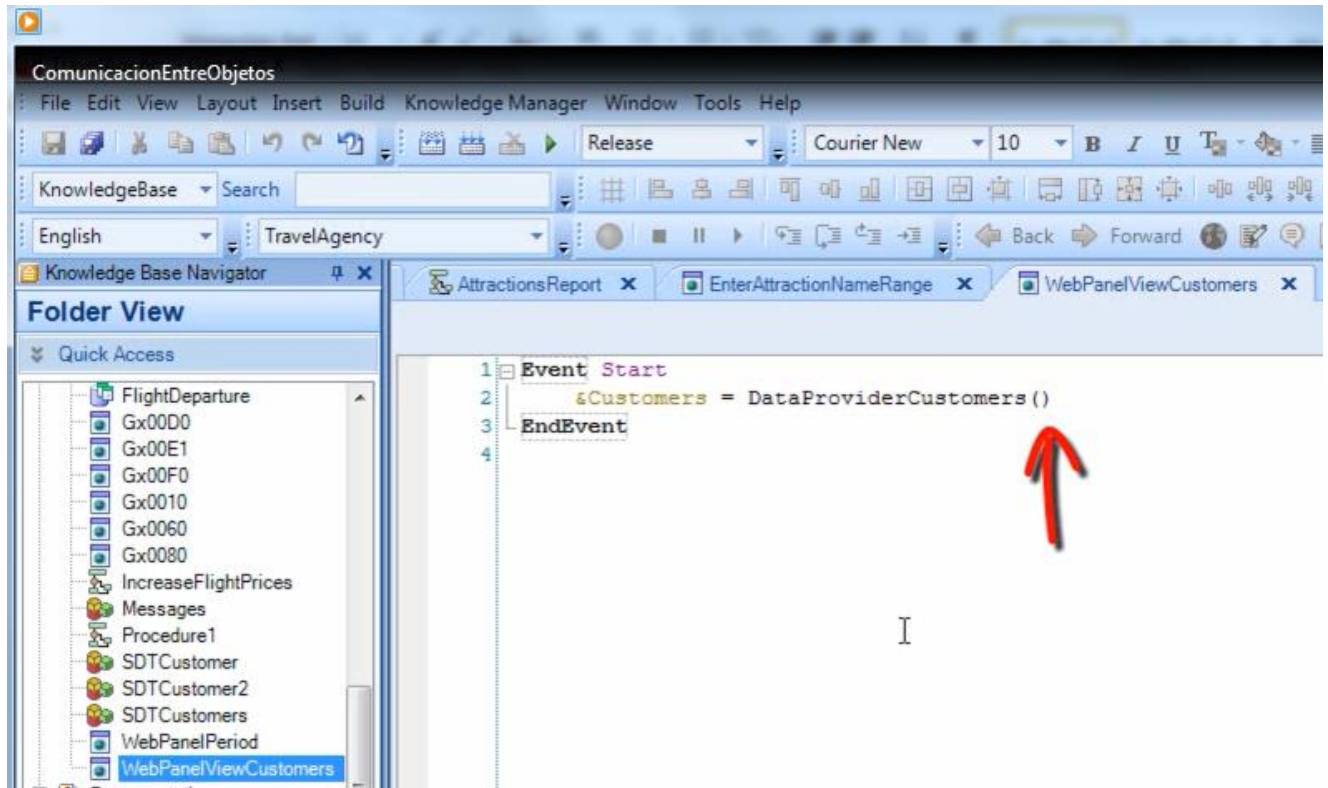
Quick Access

- FlightDeparture
- Gx00D0
- Gx00E1
- Gx00F0
- Gx0010
- Gx0060
- Gx0080
- IncreaseFlightPrices
- Messages
- Procedure1
- SDTCustomer
- SDTCustomer2
- SDTCustomers
- WebPanelPeriod
- WebPanelViewCustomers

```
1 Event Start
2 &Customers = DataProviderCustomers()
3 EndEvent
4
```

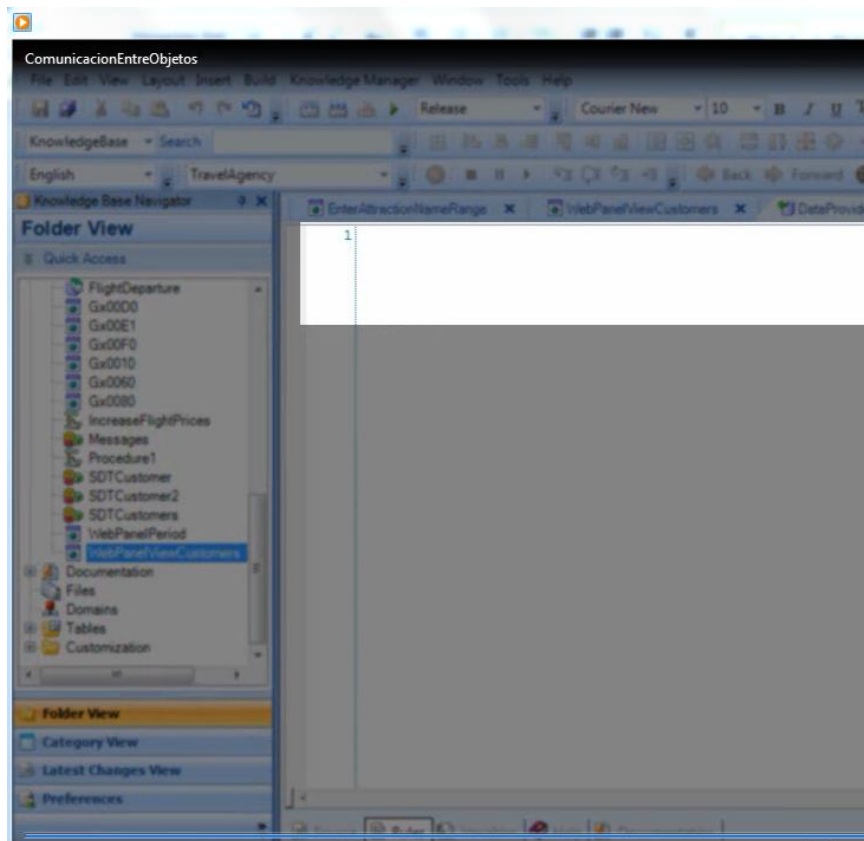

Assigns to the &Customers variable, defined as a collection of customers, what the DataProviderCustomers returns.

In this case, no parameters are sent to the object called

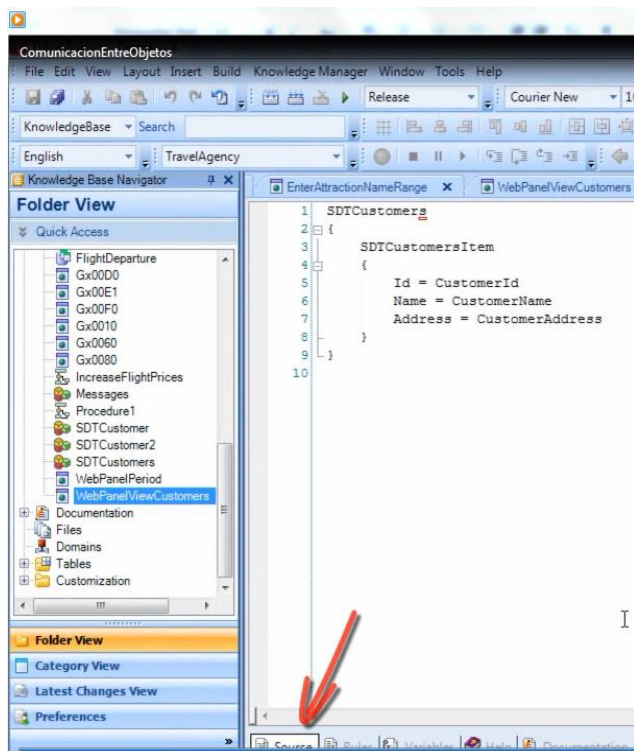


because the parentheses contain nothing inside.

Therefore, the Data Provider called has no parm rule declared.



Remember that when we drag the source of this Data Provider,



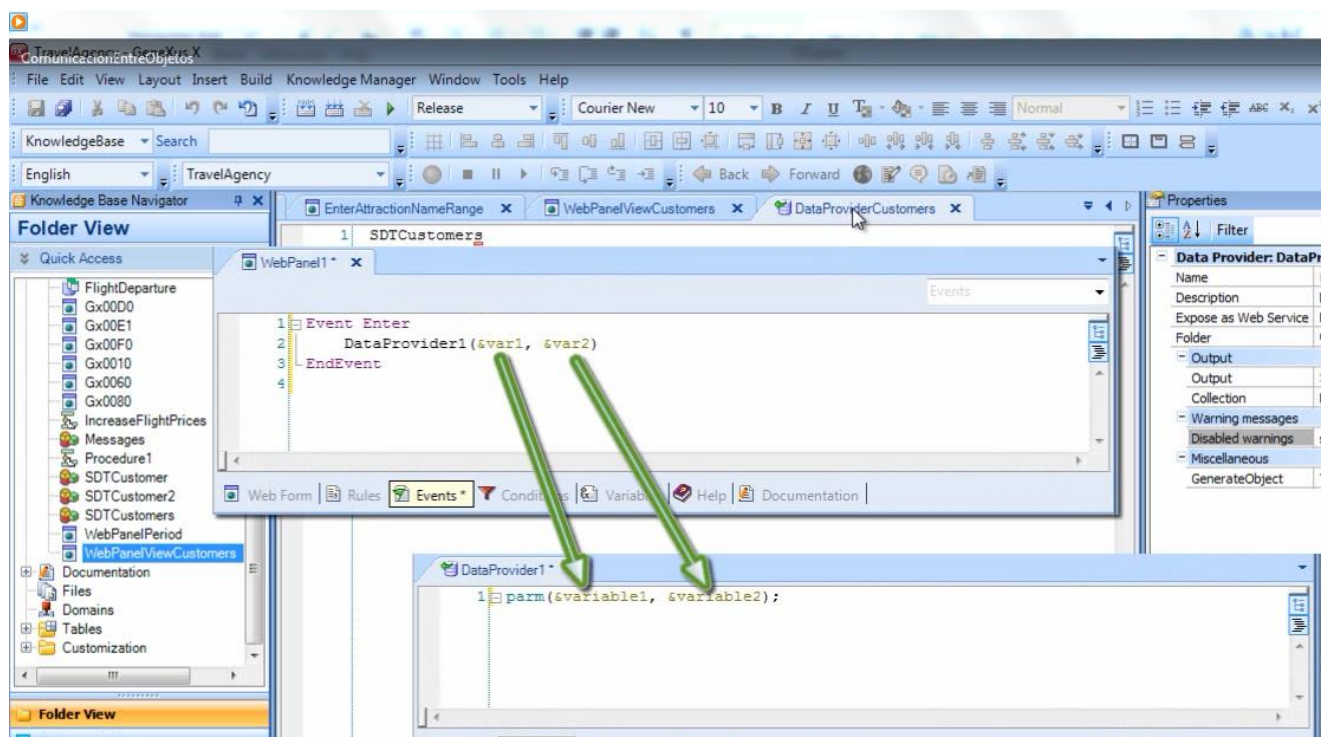
the SDT we wanted to load, the Output property of the Data Provider was automatically completed

| Properties | |
|--------------------------------------|-------------------------|
| Filter | |
| Data Provider: DataProviderCustomers | |
| Name | DataProviderCustomers |
| Description | Data Provider Customers |
| Expose as Web Service | False |
| Folder | Objects |
| Output | |
| Output | SDTCustomers |
| Collection | False |
| Warning messages | |
| Disabled warnings | spc0096 spc0107 spc0142 |

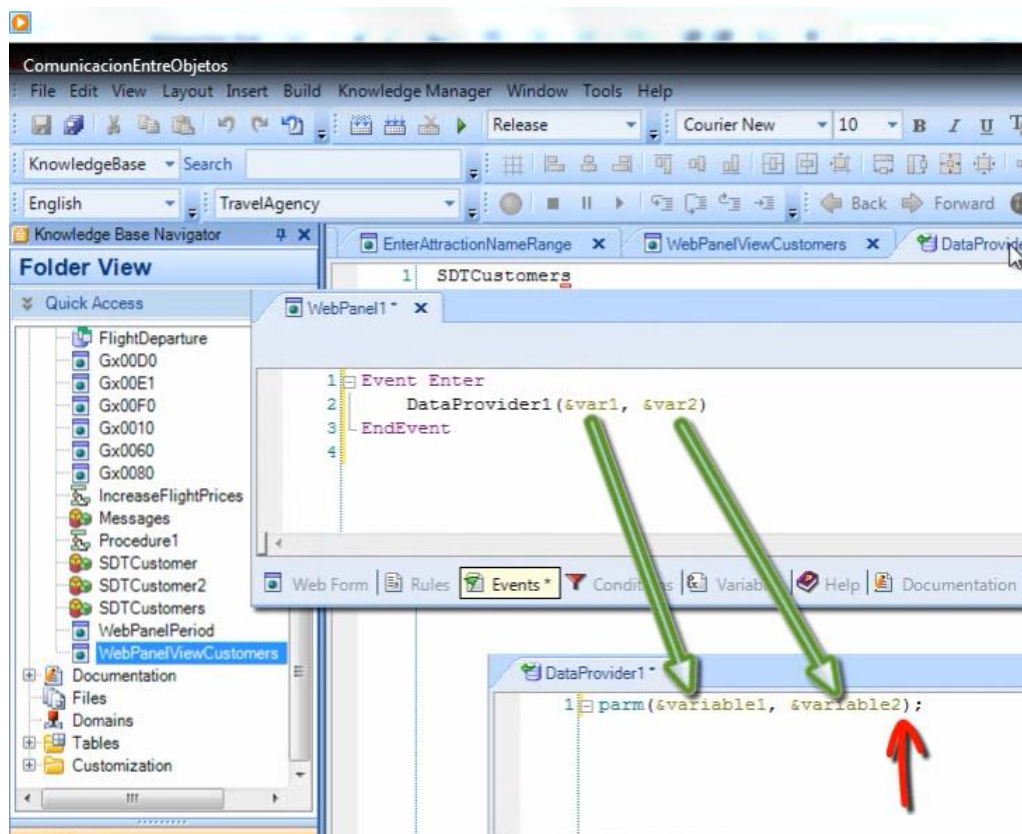
with the name of the SDT we dragged.

As described by the name of the Output property, what is returned is now configured in the object, that is: the output of the Data Provider.

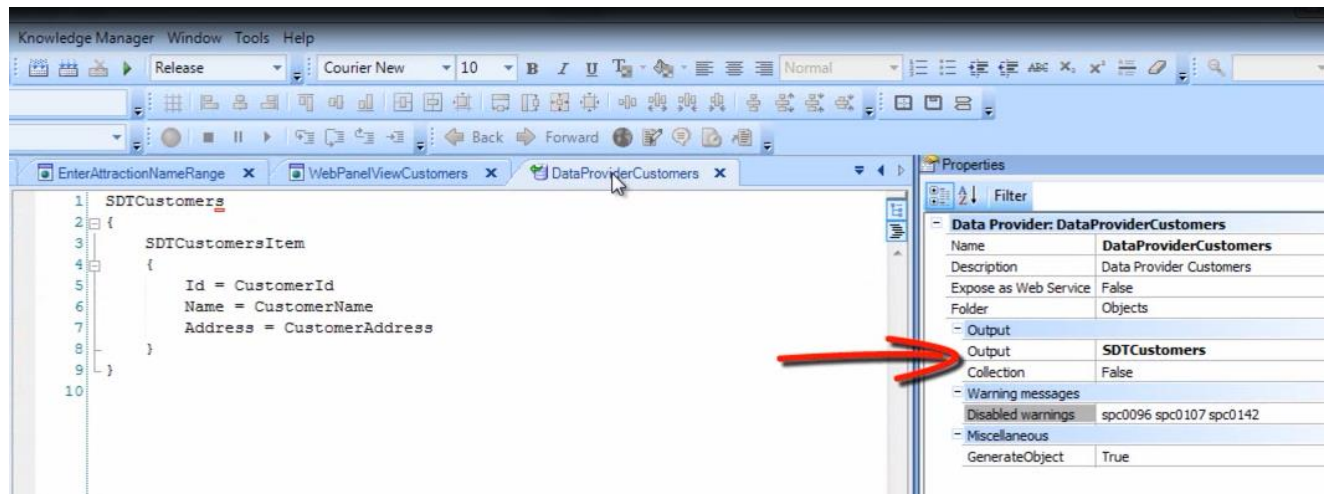
This is why, in Data Provider objects, we have to receive in the parm rule the same number of parameters sent in the call

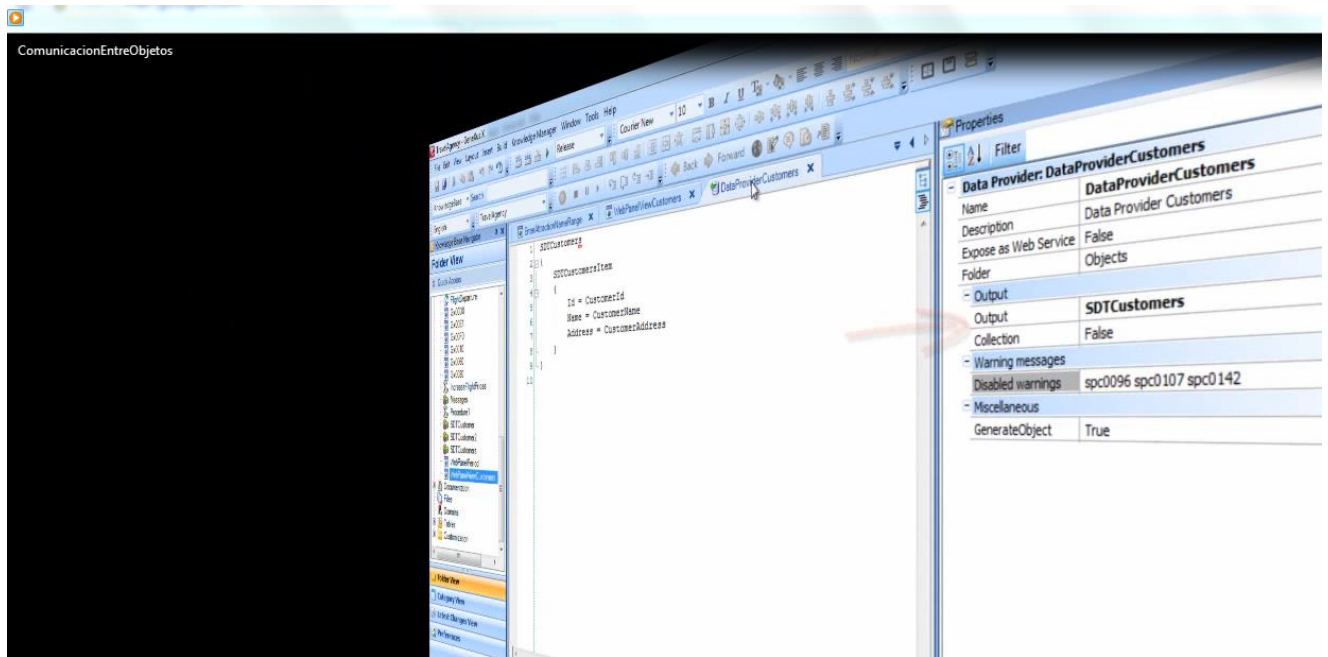


without adding an extra return parameter at the end

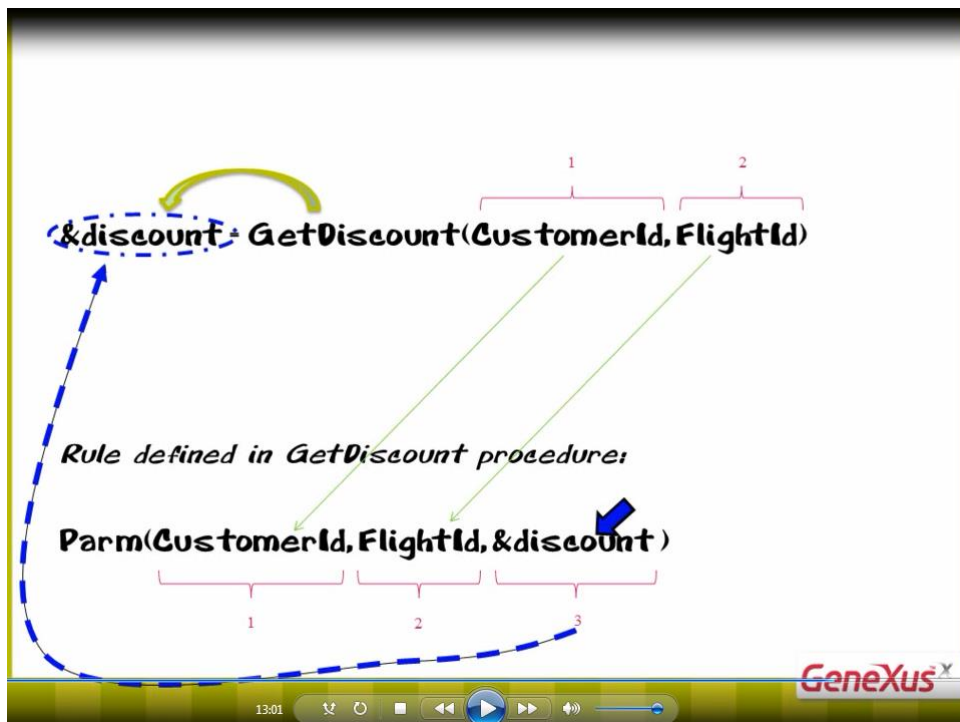


because the output is defined by the Output property of the Data Provider



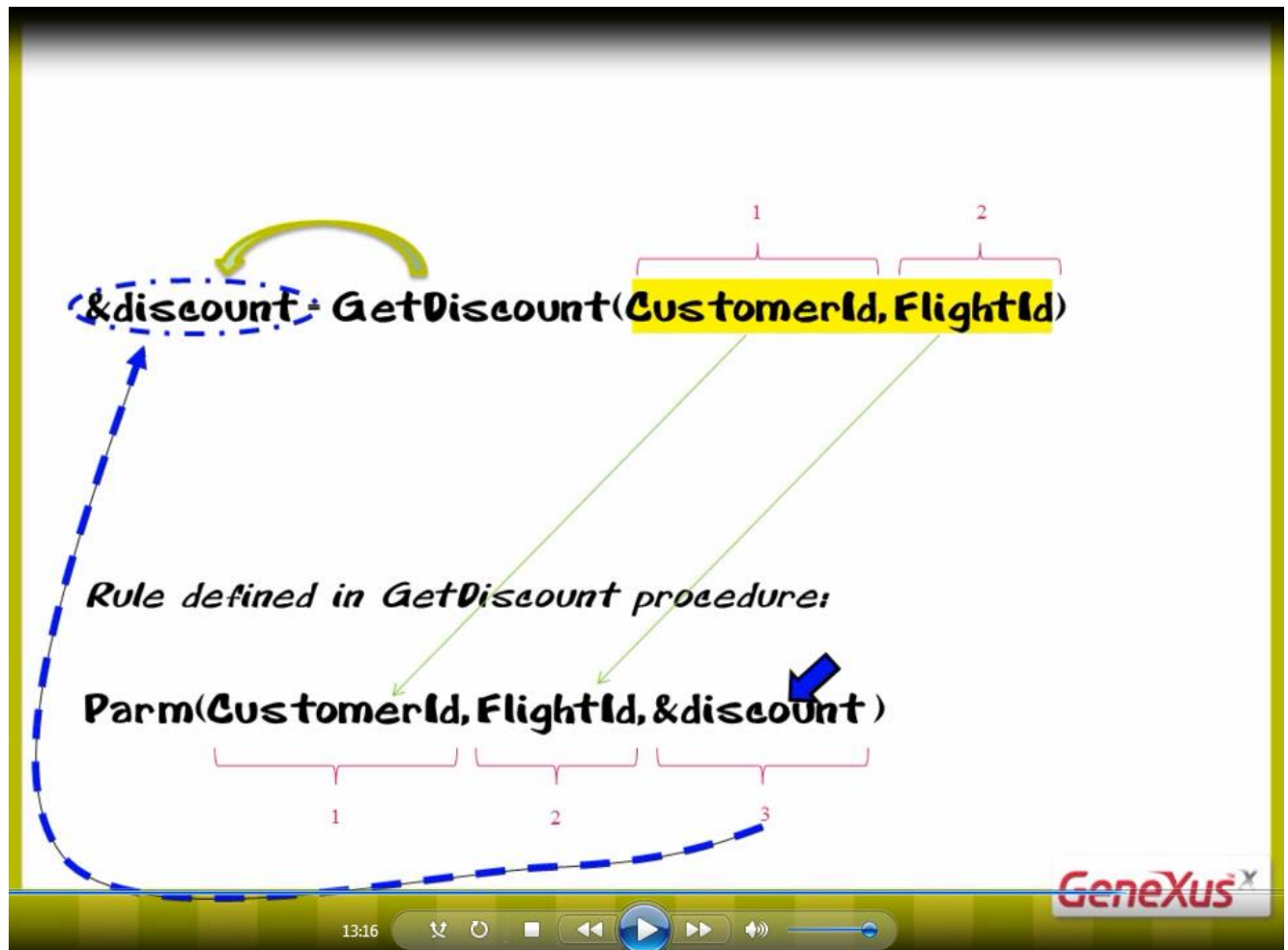


To round off, we will see one last thing,



in this example, some parameters are variables and other are attributes. How do we select what to send and what to receive?

When sending data to an object that is called, there are no doubts possible: if the data is in an attribute



the attribute is included as parameter, and if it is in a variable, the variable is used.

An example where we have the data in attributes is the case where the call to the GetDiscount procedure is defined in a rule of a transaction

| Name | Type | Description |
|------------------|---------------|--------------------|
| Invoice | Invoice | Invoice |
| InvoiceId | Id | Invoice Id |
| InvoiceDate | Date | Invoice Date |
| CustomerId | Id | Customer Id |
| CustomerName | Character(20) | Customer Name |
| CustomerLastName | Character(20) | Customer Last Name |
| Flight | Flight | Flight |
| FlightId | Id | Flight Id |
| FlightPrice | Price | Flight Price |

GeneXus[®]

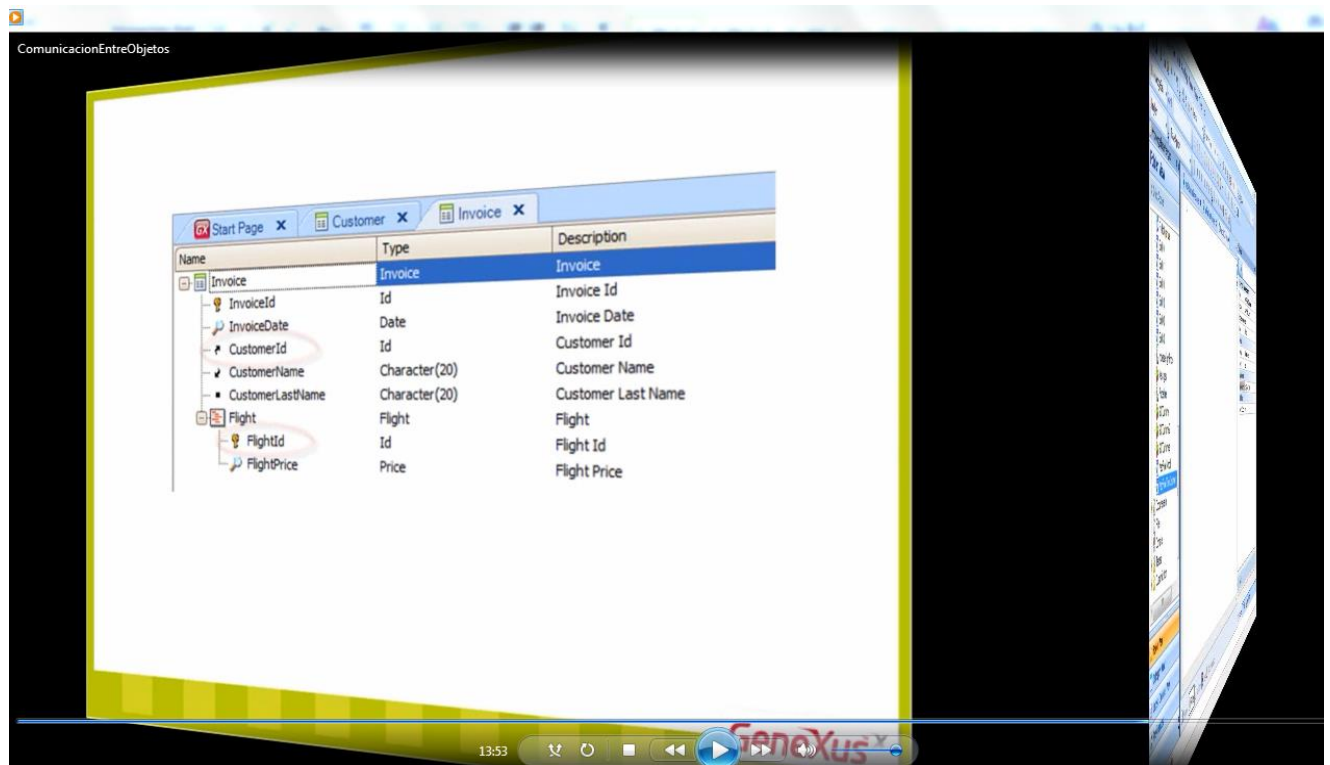
where the user has entered a customer identifier and a flight identifier

| Name | Type | Description |
|------------------|---------------|--------------------|
| Invoice | Invoice | Invoice |
| InvoiceId | Id | Invoice Id |
| InvoiceDate | Date | Invoice Date |
| CustomerId | Id | Customer Id |
| CustomerName | Character(20) | Customer Name |
| CustomerLastName | Character(20) | Customer Last Name |
| Flight | Flight | Flight |
| FlightId | Id | Flight Id |
| FlightPrice | Price | Flight Price |

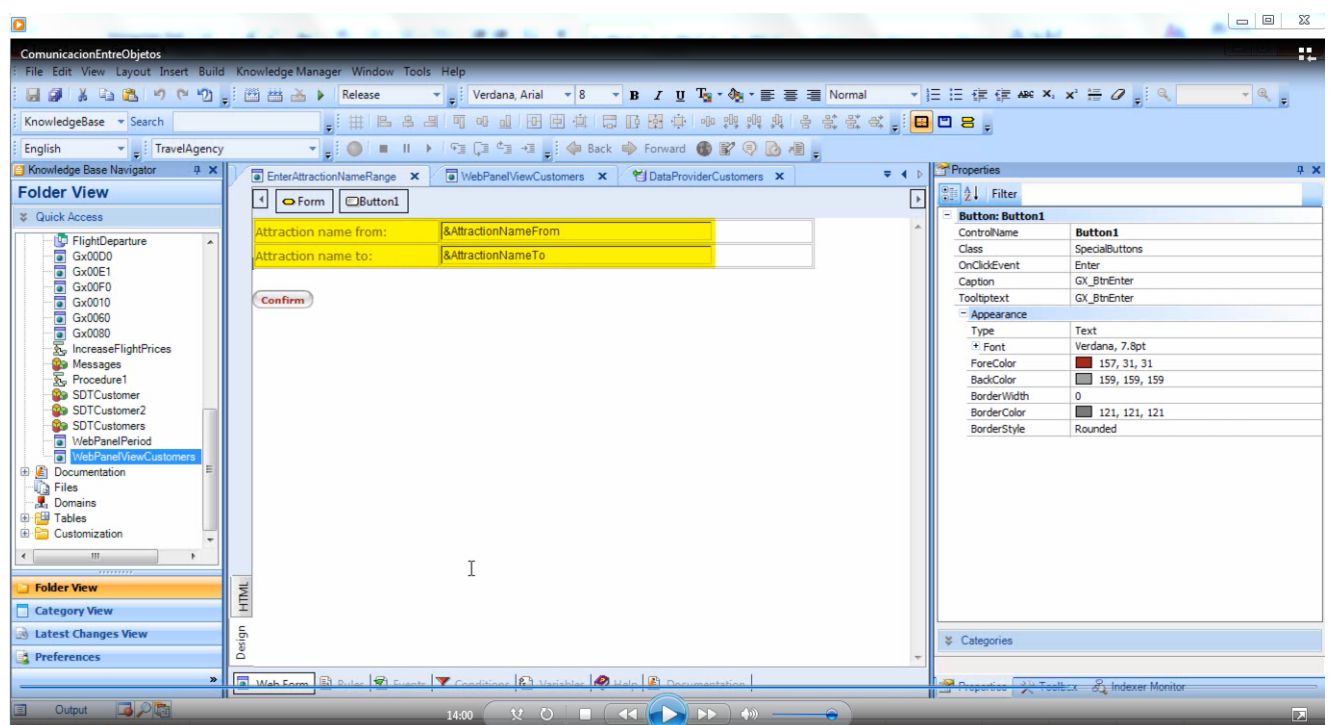
GeneXus[®]

and the values to be sent to the procedure are in attributes.

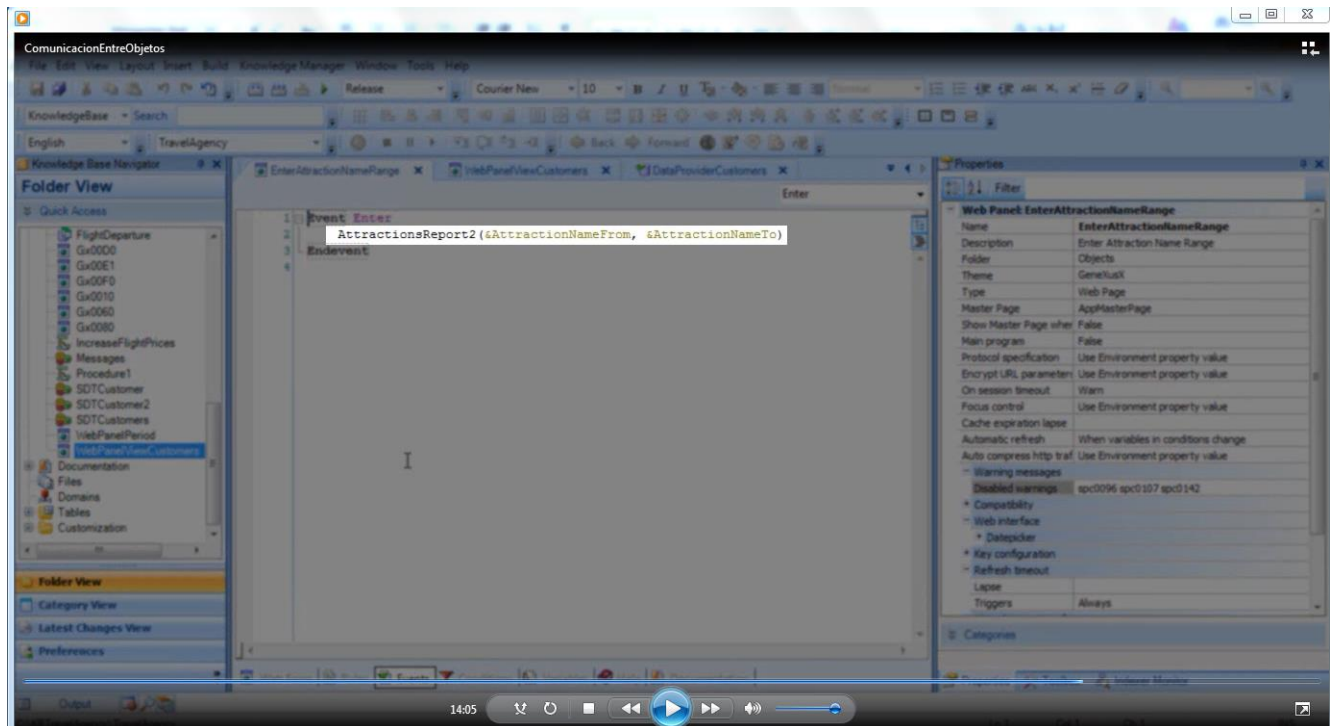
Another example of using attributes could be when the call itself is defined in a For each that navigates a given base table, has in its scope the corresponding extended table and has those attributes available.



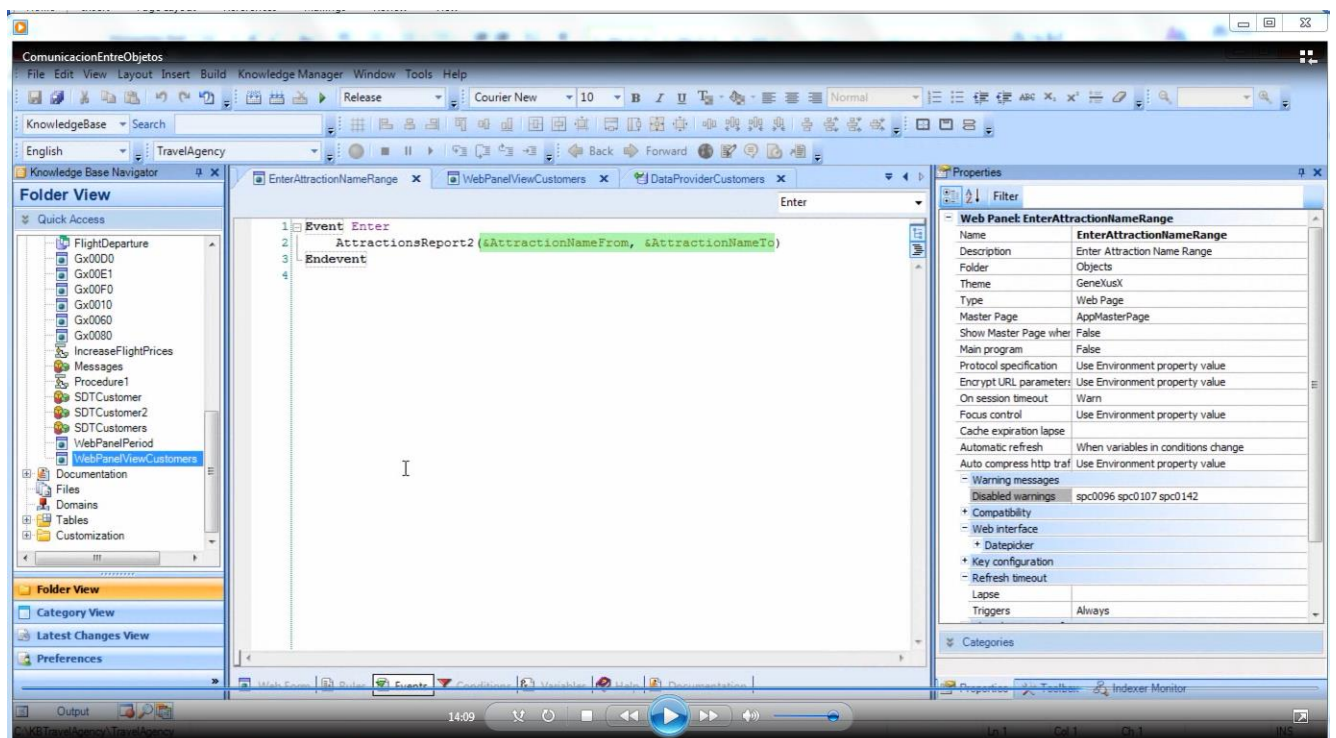
An example where we had the data in variables, was in the EnterAttractionNameRange web panel



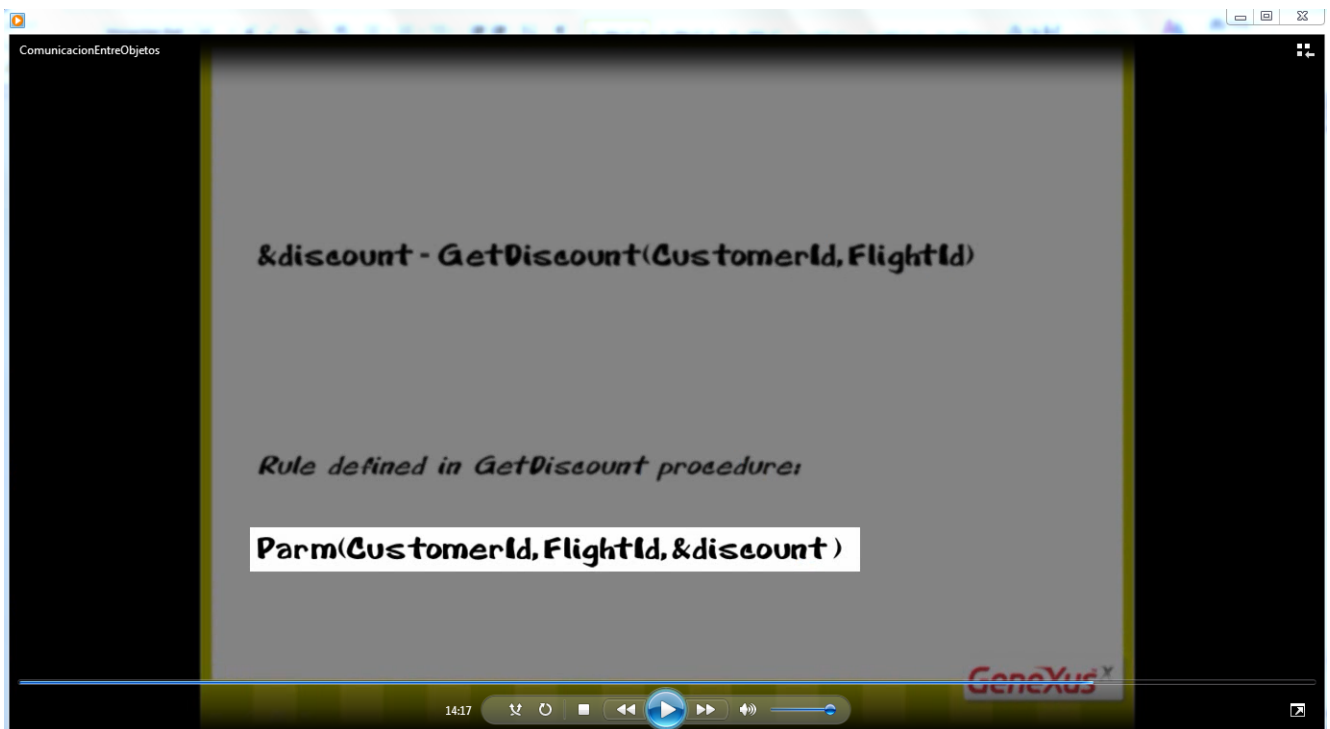
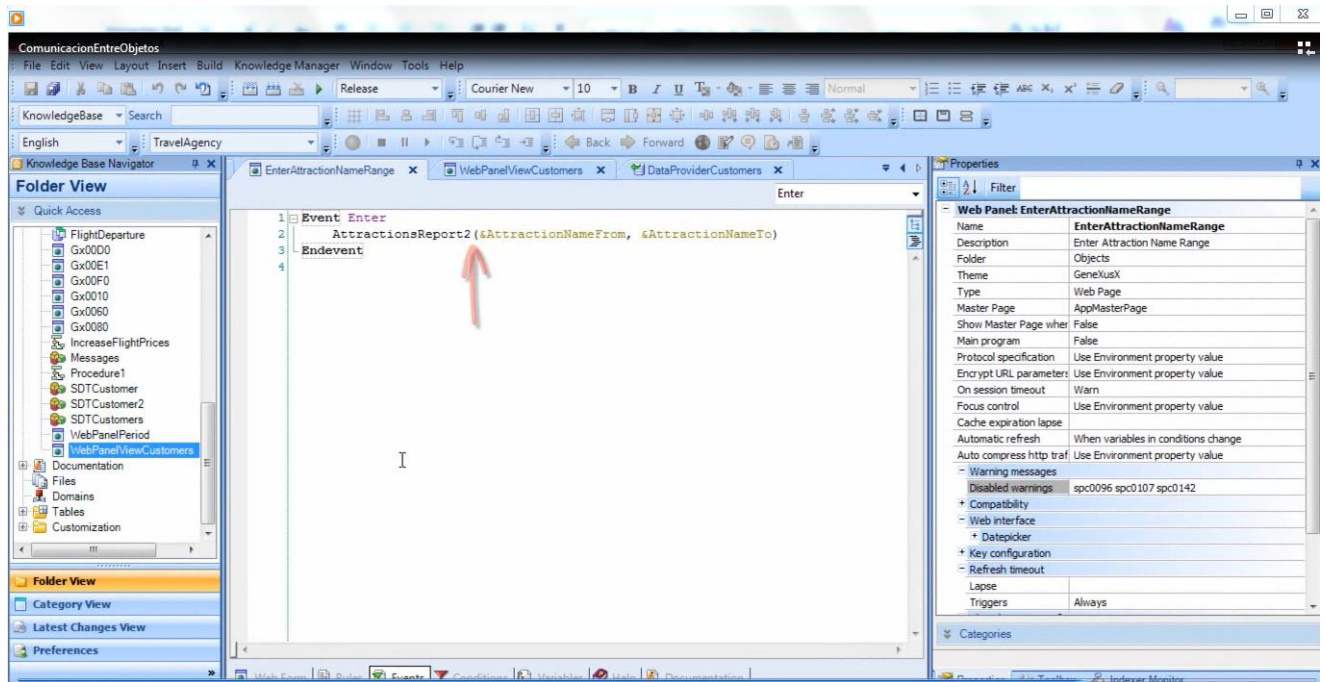
where we entered a range of attractions, and upon confirming



we invoked a procedure sending that range which is included in two variables.



Remember that here we can omit the call.

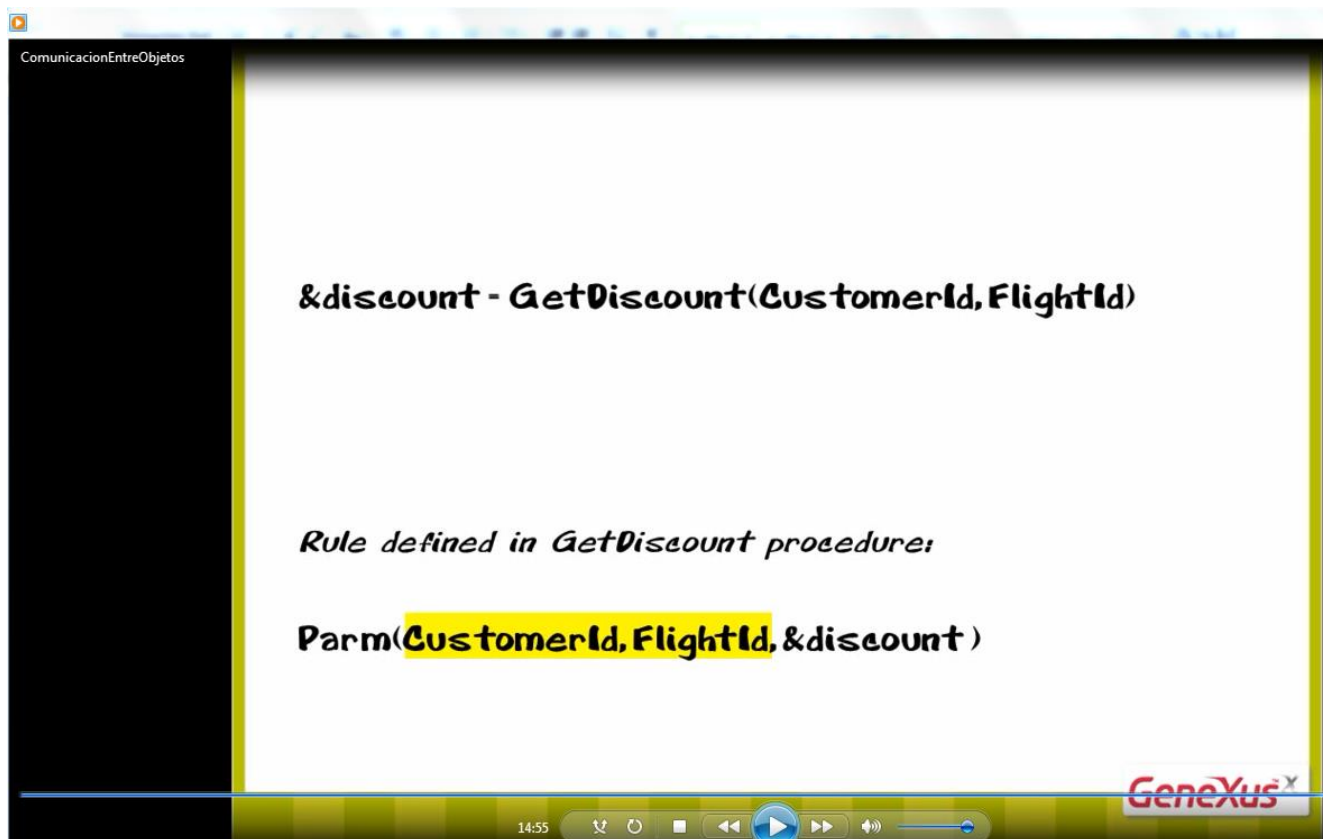


As for stating the parm rule in the object invoked, we can decide **for each parameter received**, whether we will state an attribute or a variable, regardless of how it was sent.

What is the difference between using a variable or an attribute in the **parm** rule of the object invoked?

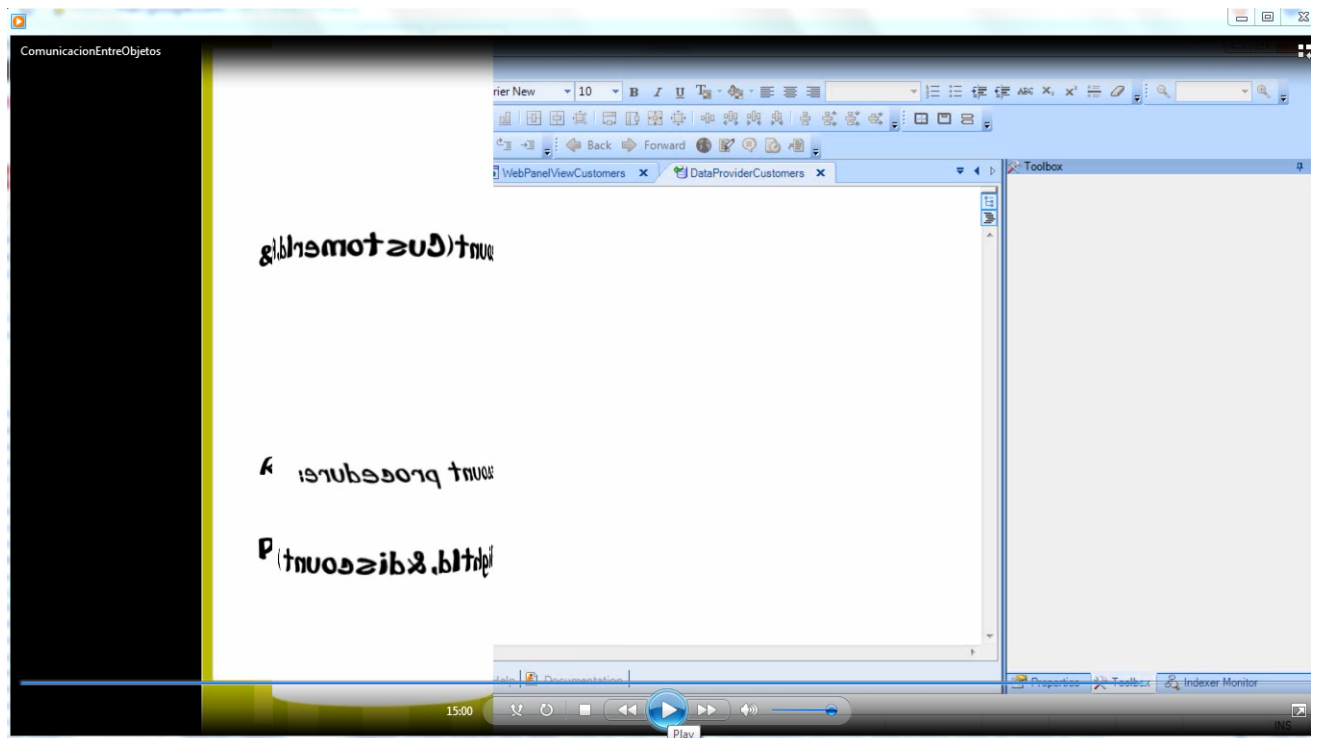
If we receive the value in a variable, it may be used freely in programming; as a filter condition for equality, greater than, greater than or equal to, lesser than, lesser than or equal to... it may be used for some arithmetical operation, or whatever we may need to do with it.

If we receive the value in an attribute,

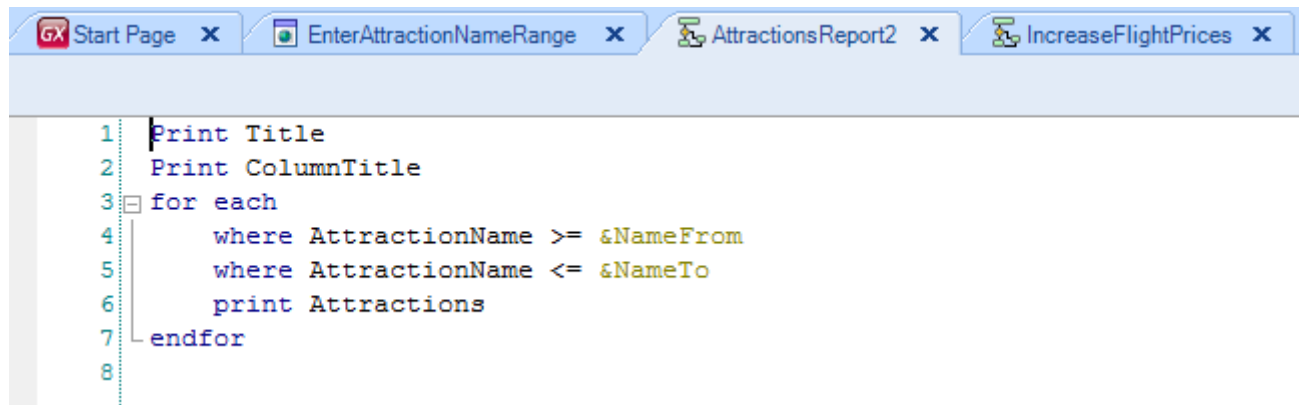


it will automatically act as filter for equality in the object.

Let's see this.

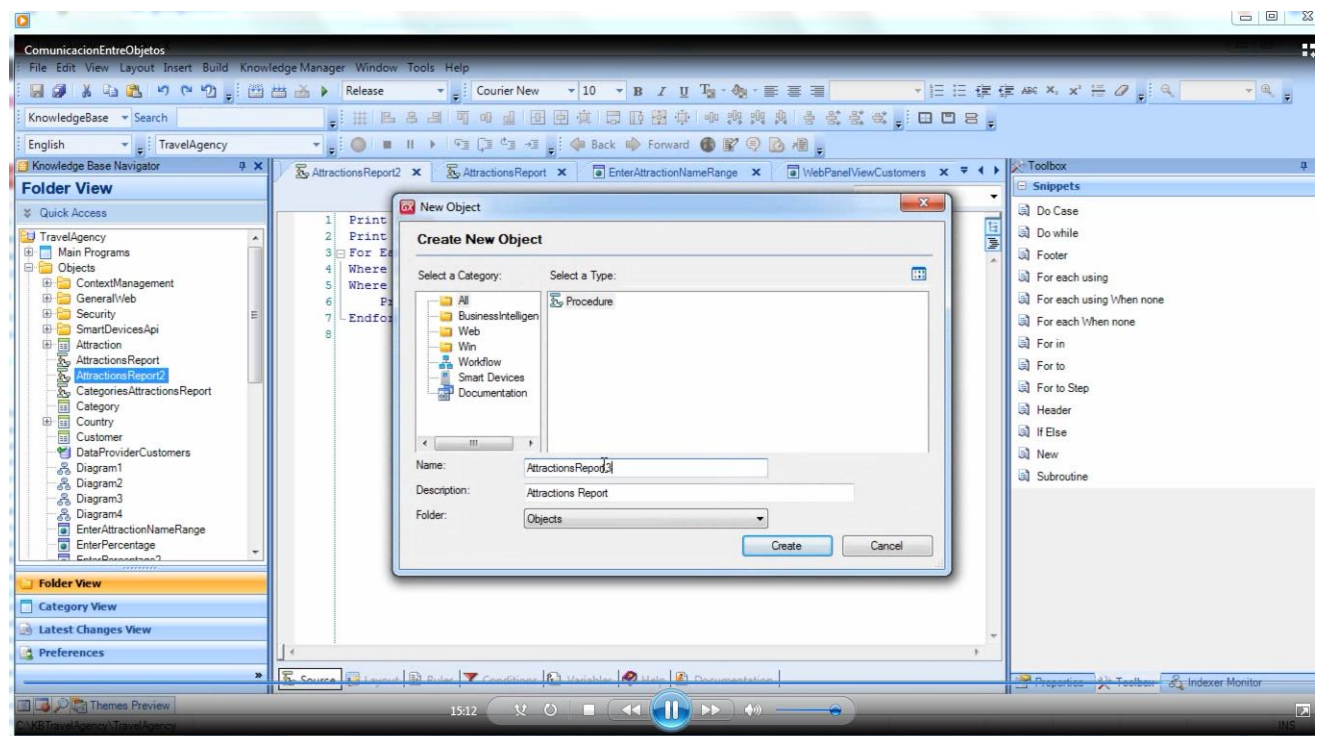


Let's save a copy of this procedure with a different name.



```
1 Print Title
2 Print ColumnTitle
3 for each
4     where AttractionName >= &NameFrom
5     where AttractionName <= &NameTo
6     print Attractions
7 endfor
8
```

We call it AttractionsReport3.



In this new object, instead of receiving two variables containing a start and end value of attraction names... we will receive one single variable with an attraction identifier, whose data we want listed. That means that the idea here is to list the data of a single attraction.

We go to the parm rule, remove these two variables and write &AttractionId.

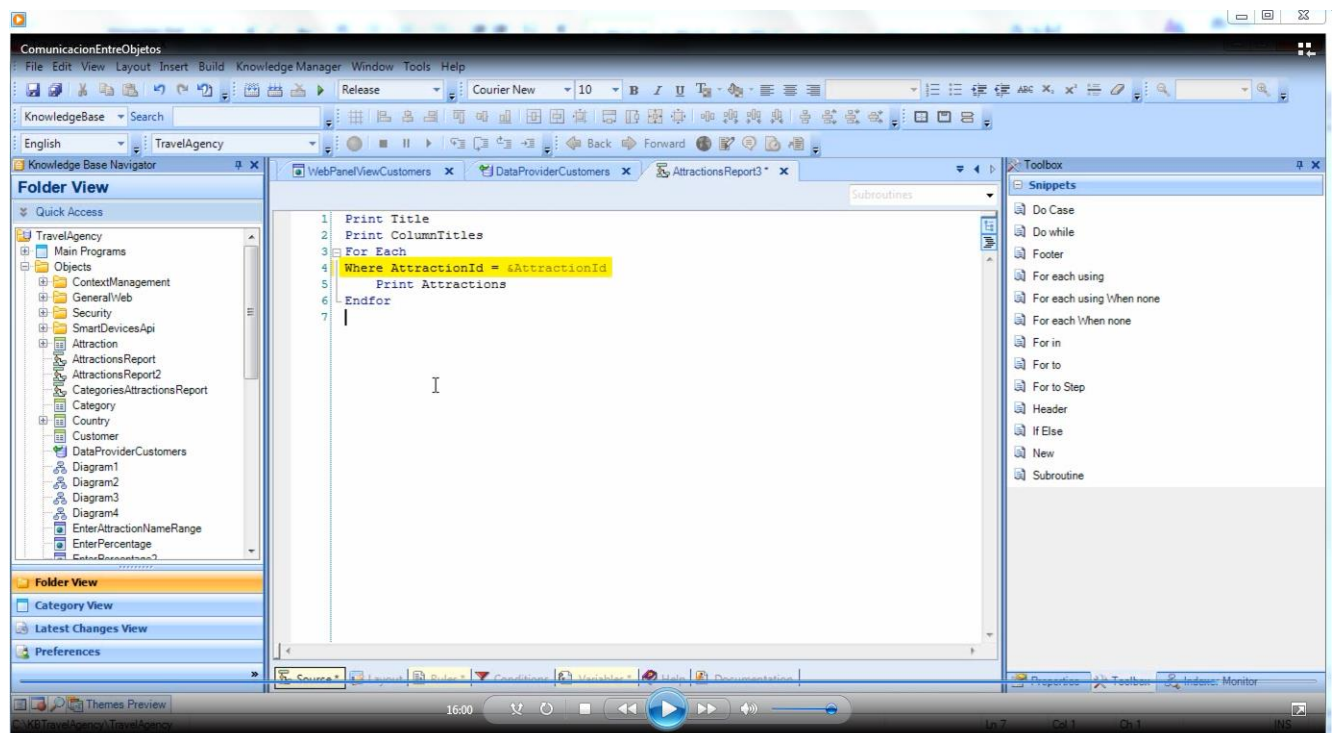
Since we are at it, we change the name of the pdf file to: AttractionsReport3.

```
1 parm(&AttractionId);
2 output_file('AttractionsReport3.pdf','pdf');
3
```

We now go to the variables section and define the variable &AttractionId.

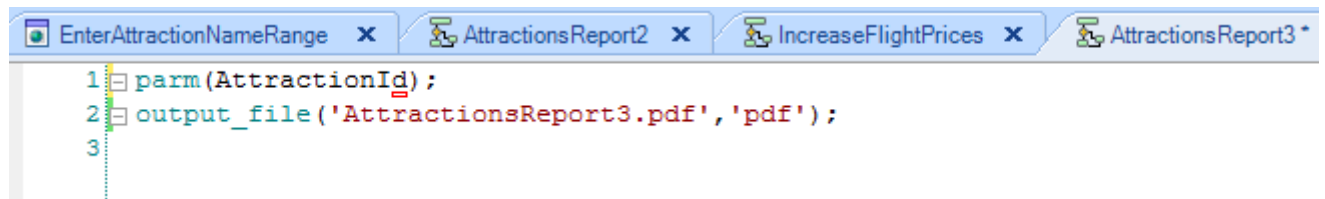
| Name | Type | Is Colle... | Description |
|-----------------------|--------------------------|--------------------------|---------------|
| Variables | | | |
| Standard Variables | | | |
| Autodefined Variables | | | |
| NameFrom | Attribute:AttractionName | <input type="checkbox"/> | Name From |
| NameTo | Attribute:AttractionName | <input type="checkbox"/> | Name To |
| AttractionId | Attribute:AttractionId | <input type="checkbox"/> | Attraction Id |

And then change the For Each filters in the source



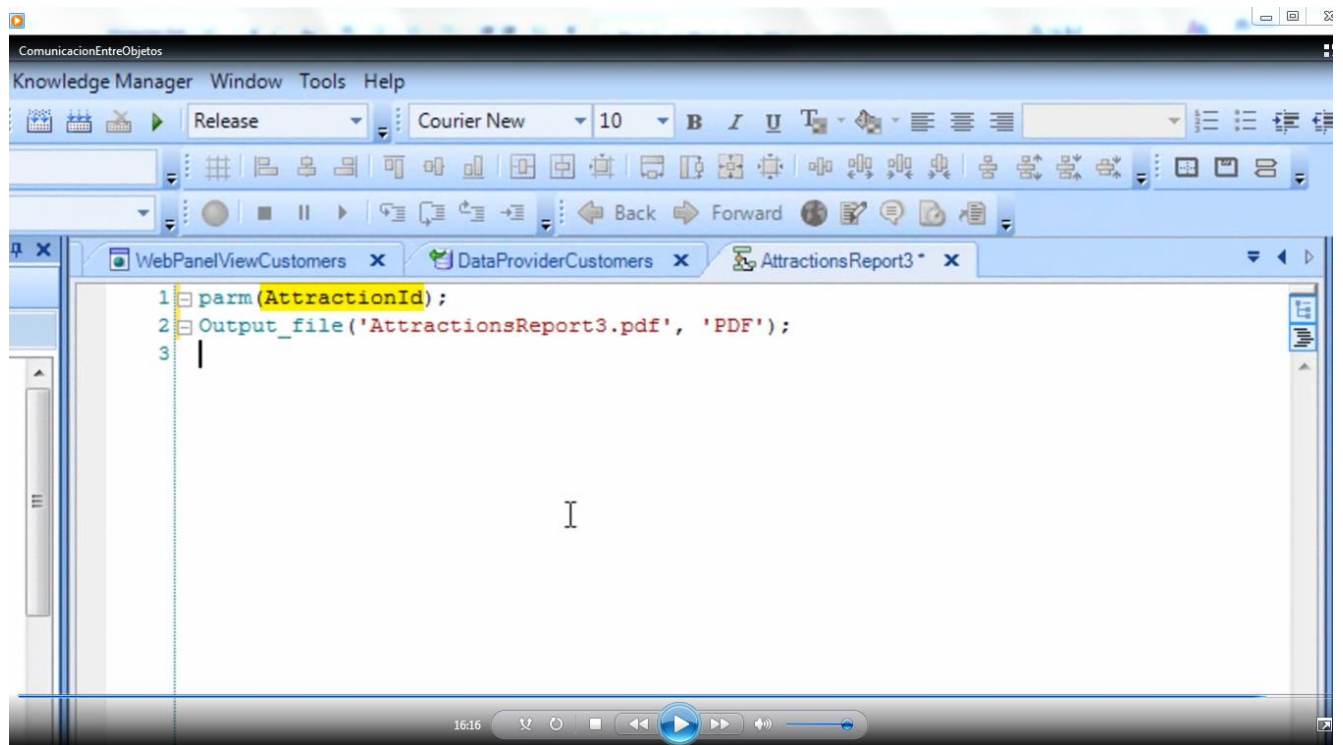
It is clear that with this definition we are listing the data of an attraction received by parameter.

Let's now see what happens if instead of receiving the attraction identifier in a variable we receive it in its attribute name.



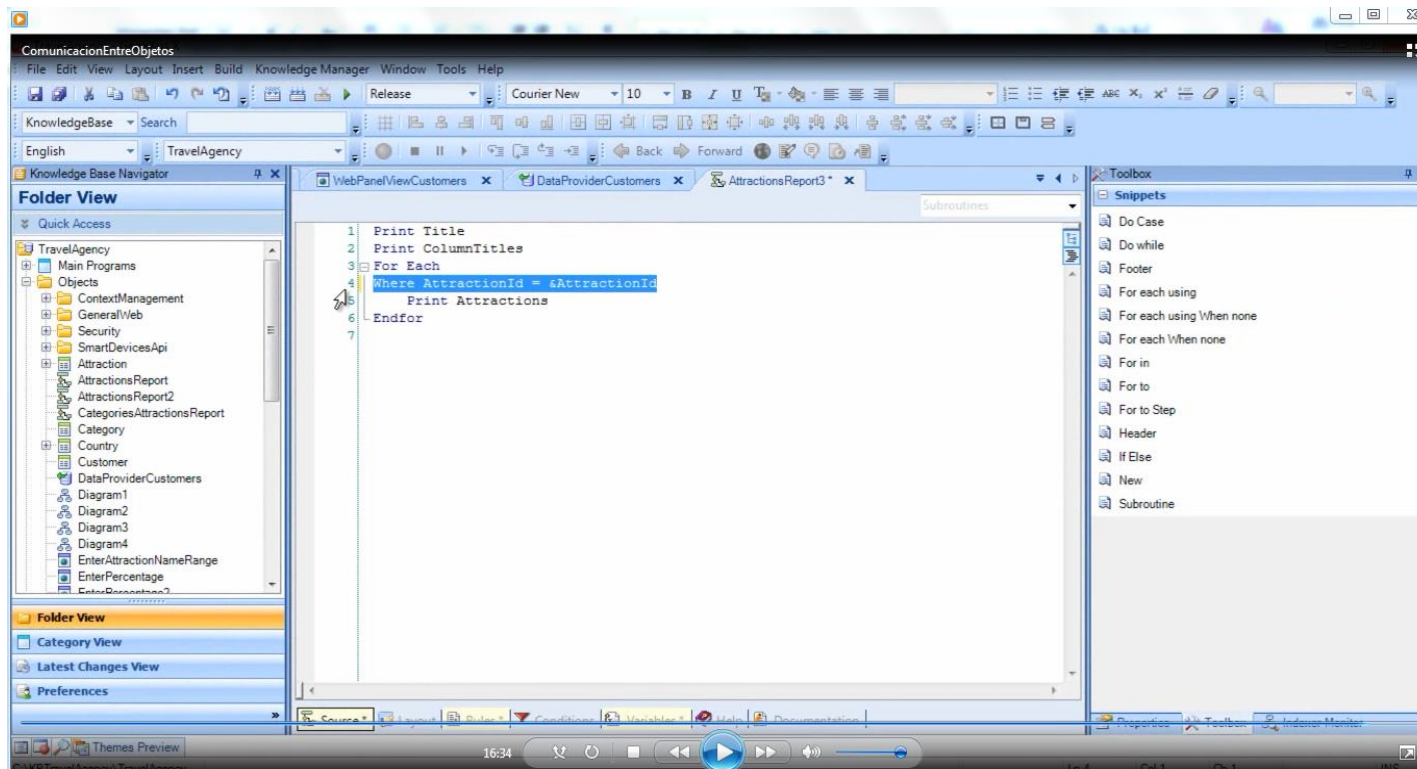
```
1 parm(AttractionId);  
2 output_file('AttractionsReport3.pdf','pdf');  
3
```

When we receive the value in an attribute in the rule,



GeneXus will filter for equality, that is, only the records with that attraction identifier value will be accessed, in all the accesses to the database that take place in the object source.

Therefore, it will not be necessary to write this where clause



Since the effect of filtering by that attribute is achieved when receiving in the attribute in the parm rule.

We then delete this line.

If we want to see the navigation of this object

ComunicacionEntreObjetos

WebPanelViewCustomers x DataProviderCustomers x AttractionsReport3 x Navigation View x

AttractionsReport3

Procedure AttractionsReport3 Navigation Report

| | | | |
|----------------|--------------------|---------------|--------------------|
| Name | AttractionsReport3 | Environment | .net Default (C#) |
| Description | Attractions Report | Spec. Version | 10_2_2-59488 |
| Output Devices | File | Form Class | Graphic |
| Main | Yes | Program Name | AttractionsReport3 |
| | | Call Protocol | HTTP |
| | | Parameters | AttractionId |



Levels

For First Attraction (Line: 11)

Order: AttractionId
Index: IATTRACTION

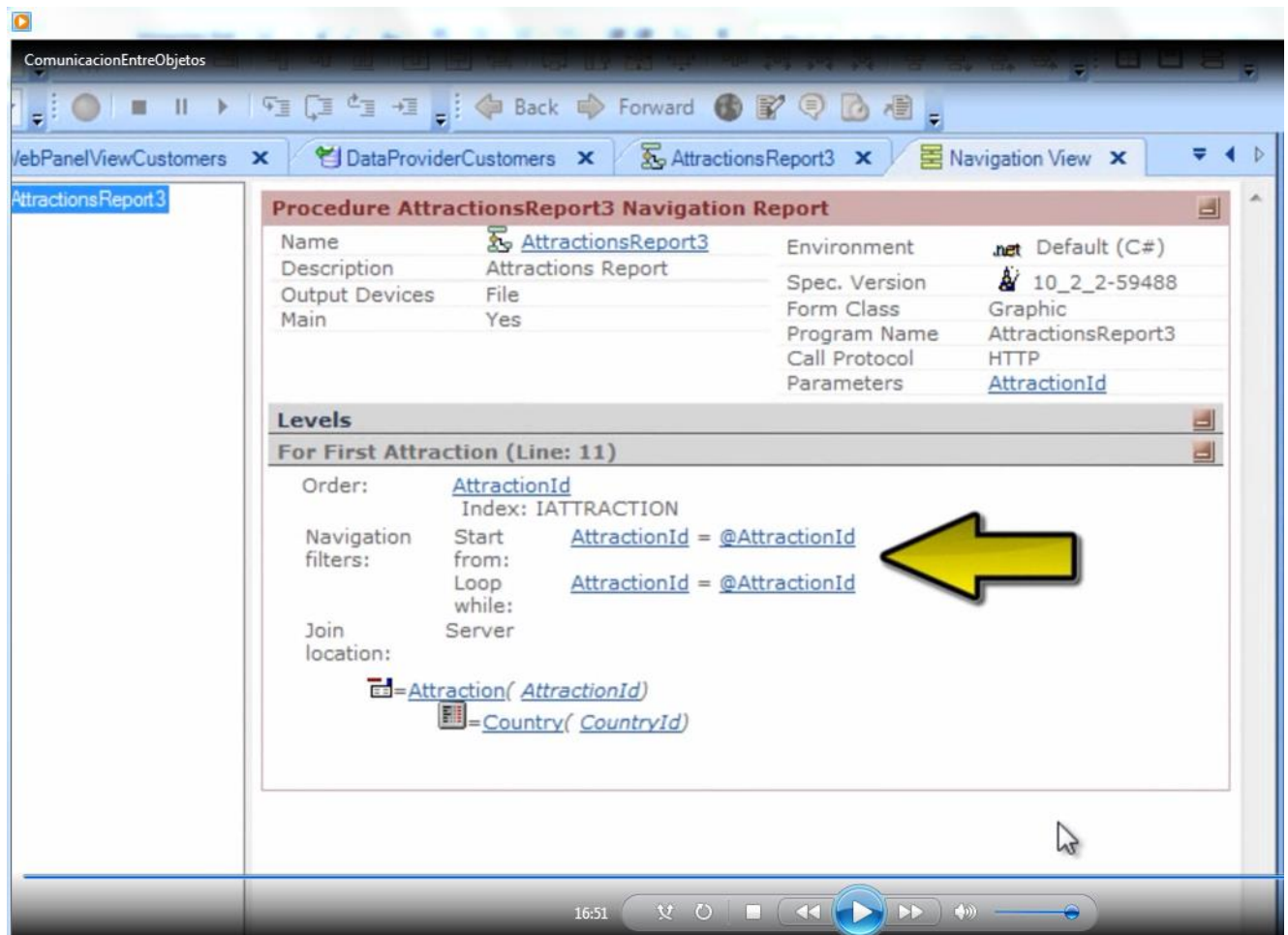
Navigation Start AttractionId = @AttractionId

filters: from:
Loop AttractionId = @AttractionId
while:
Join Server
location:

 = Attraction(AttractionId)
 = Country(CountryId)

16:49

we will see that it is doing the filtering



even when the where clause is not written. If we write it, we will not get an error, but there is no need to write it.

We have seen two ways of filtering for equality the information accessed. If our objective is not to use a value received to filter for equality, then the only solution possible is to receive the values in variables to use them freely.