Invocations between objects

GeneXus™

We've used fixed values in filters... in this case by Country

```
GX Start Page  X    AttractionsList  X
Source | Layout | Rules | Conditions | Variables
Subroutines
1   print Title
2   print ColumnTitles
3 ⊟ For each Attraction order CountryName
4       where CountryName = 'France'
5       print Attractions
6   └ endfor
7
```

```
GX Start Page  X    AttractionsList *  X
Source * | Layout | Rules | Conditions | Variables
Subroutines
1   print Title
2   print ColumnTitles
3 ⊟ For each Attraction order CountryName
4       where CountryId = 2
5       print Attractions
6   └ endfor
7
```

What if we want to make the list general and "receive" the country to filter by?

In previous situations we've had to call an object from another object.

For example, when we implemented the AttractionsList procedure object, **we needed to filter the attractions whose country name was "France", or,** similarly, whose country identifier was 2 (which corresponded to "France")... **To do so, we used fixed values in the code.**

However, this implies that if we wanted to filter the attractions of a country other than France, we would have to change the procedure code every time!

Object A

Object B
(AttractionsList)

```
AttractionsList * ✕
Source *  Layout  Rules  Conditions  Variables
Subroutines                              ⌄
 1   print Title
 2   print ColumnTitles
 3 ⊟ For each Attraction order CountryName
 4       where CountryId = &parm
 5       print Attractions
 6 └ endfor
```

Value

&parm
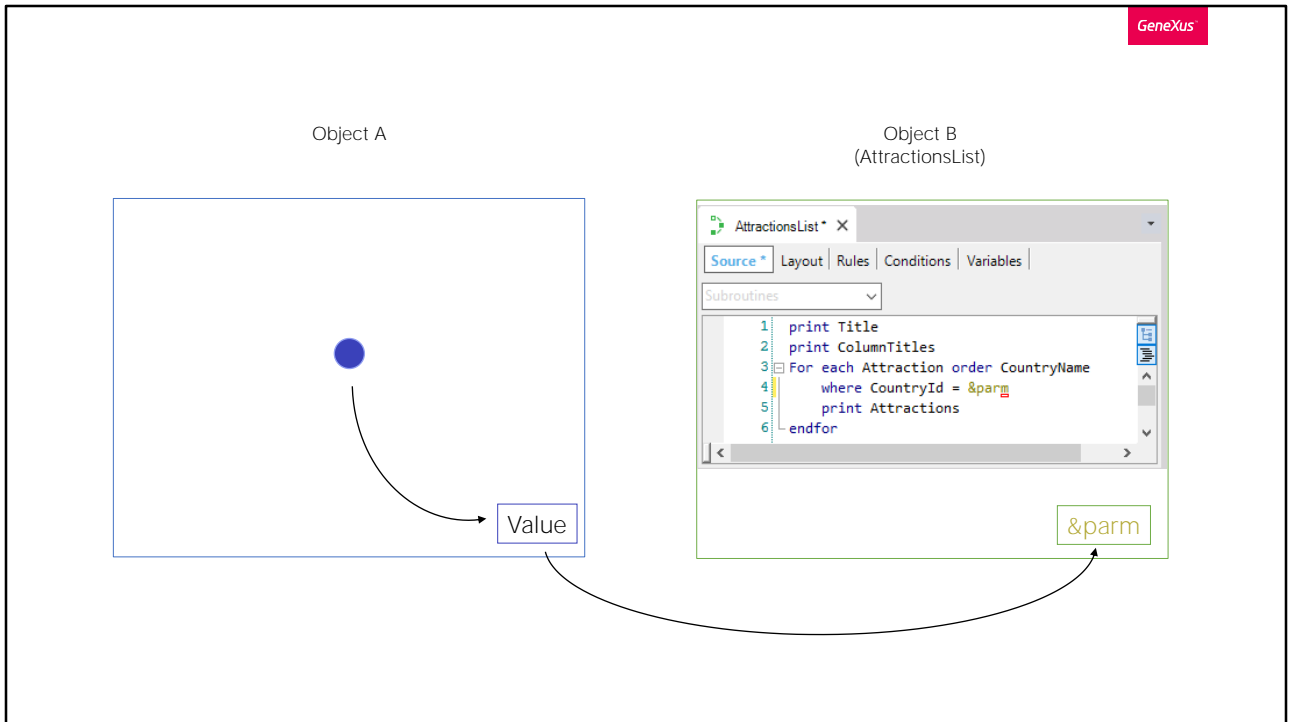
Ideally, we should be able to "receive" in this object the value that we want to filter by. In other words, that another GeneXus object allows the user to **choose that value... and then sends it to this procedure object to have the** attractions listed according to the country received.

Next, we will use this example to see how to implement this communication between GeneXus objects.

Establishing communication between objects.
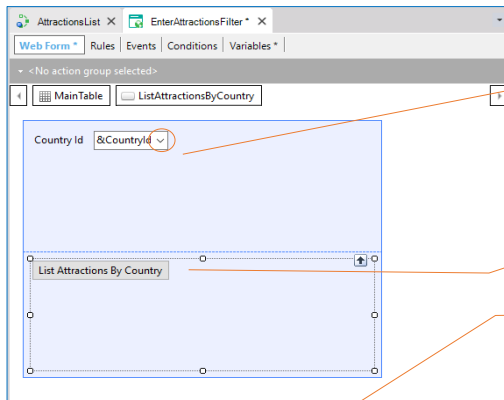
1) We create a web panel that asks for the country to be considered.

Variable with Dynamic Combo control type to show the user the countries in the database.

2) We add a button to call the AttractionsList procedure.

Button that has the ListAttractionsByCountry event associated with it

Variable containing the country indicated in the web panel form

First, we need to create an object that provides a screen to prompt the user for values and do something with those values. One of the objects that enables this is the web panel, which will be studied in detail later. For now, let's say it is a visual, flexible panel that can prompt the user for data, as well as show information from the database or other sources, among other things. For example, the attractions Work With element was automatically implemented by GeneXus as a Web Panel.

So, we will create an object of this type, and call it EnterAttractionsFilter. Note that a Web Form will be created to be the object screen. It contains a single table

We add a CountryId variable... Since its name is the same as that of the attribute, it is based on it and, therefore, takes its same data type. In this way, if we change the attribute's data type, for example, from numeric (10) to numeric (4), the variable will automatically take this new value.
Now we edit the variable properties and see that its Control Type property takes the Edit value. This means that when the web panel is executed, this field will expect the user to enter a numeric value, but it will not provide any help to choose from the values existing in the database or indicate the corresponding country. We will change the control type to Dynamic Combo Box. In this way, the user will be offered a series of values

extracted from the database for him to choose one. What values?

Those of the CountryId attribute. That is to say, the Country table will be run through and the existing CountryIds will be loaded in the combo box. But, since identifiers don't usually provide any details, even though the variable will store a country identifier, the user will be shown the content of the attribute indicated in the Item Descriptions property of the variable... We choose to show the country name. Note that the arrow that indicates the combo is displayed. In sum, at runtime it will offer a combo box with a list of the countries stored in the database to choose the one we're interested in.

Also, we add a button. We're asked to enter the name of the event that will **be associated with that button. We call it: "List Attractions By Country".** We see that the button's text takes the same default name. If we click on it, right-click, and select Go to Event... we see that an event with that name has been created, and automatically changed from the Web Form tab to the Events tab. Also, the cursor is waiting for us to enter the code that will be run when this event is triggered. That is to say, when the user presses the associated button.

What we need to do now is call the AttractionsList procedure object that lists the attractions and send the country that we want to use to filter them.

Note that when we press the button and run this code, the &CountryId variable will contain the country identifier of the country selected by the user in the combo box displayed on the screen. Previously, we saw that a variable is a portion of memory that is given a name and used to temporarily save a data item. Also, that each object has its variables section. That is to say, variables defined in an object are only known in this object.

So, if two objects have a CountryId variable, even if they have the same name, they will be two different variables.

Establishing communication between objects.
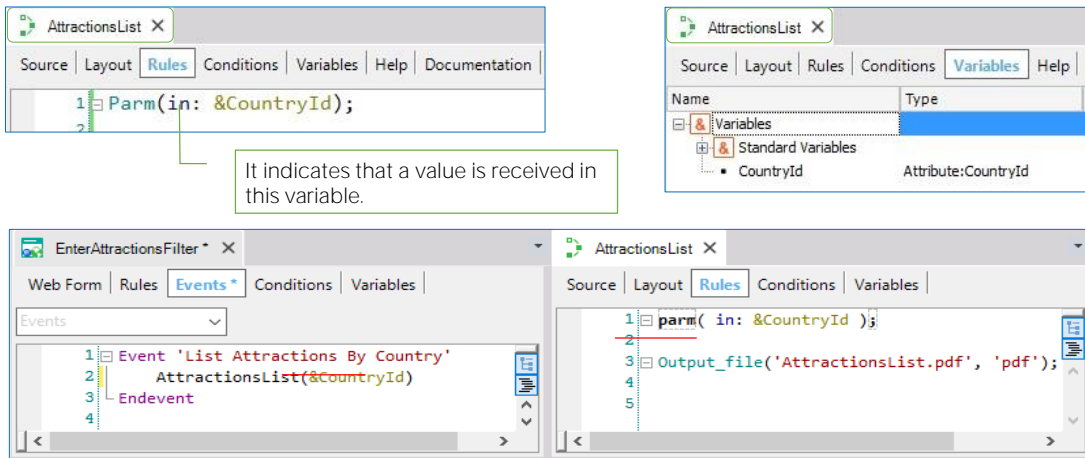
Object A   Object B

Parm
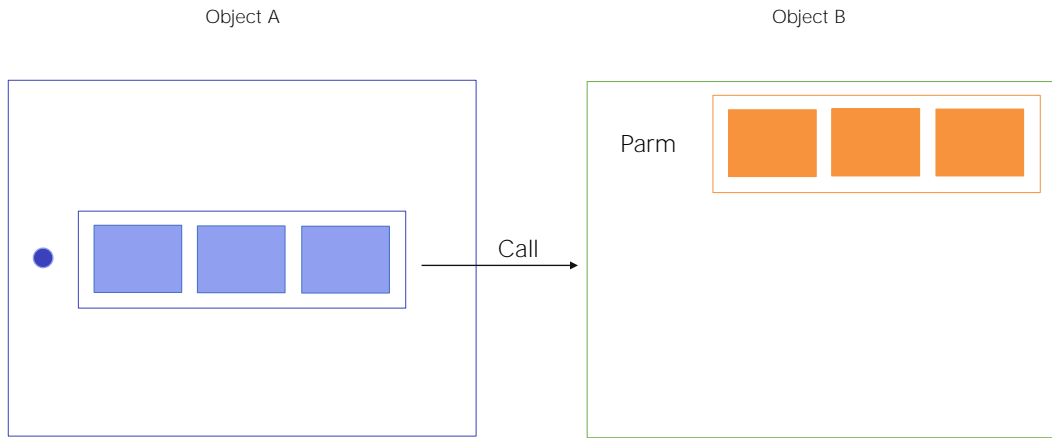
Call

So, how do we make an object A call another object B at a given moment, sending it values? Also, this object B should be able to load in its internal variables the values sent to it, in order to do something with that information.

For an object to be able to receive values (which we call parameters), we must open its Rules section and write a Parm rule. This Parm rule declares the parameters that the object can receive and/or return to the caller.

Parm rule

For an object to be able to receive values (parameters), the Parm rule must be used.

It indicates that a value is received in this variable.

Since in our example the values will be received by the AttractionsList procedure object, we open the object
and go to its rules section. We type...

Note that the Toolbox shows all the rules that we could enter in an object of this type. Among them is the Parm rule, and we could have dragged it from there.

In addition, we are informed that we have to replace this with an attribute or variable. Next, we will talk about attributes. For now, we will only look at variables.
With "in" we indicate that the &CountryId variable will be an input parameter. This means that it will only be used to receive a value from the caller. It will not return any values. We can skip this information and have GeneXus infer it.

We have written the variable name, but we haven't entered it as a variable in the object. To do so, we can click on the name, right-click, and select **"Add variable":**
If now we open the variables tab we can see that it has been defined. By default, it is based on the CountryId attribute. This is because it has the same name as an attribute.

In this object we have created the variable with the same name and data type as the one we used in the web panel for the user to enter the country.

However, as we've said, they are two different variables. One is only valid in the web panel and the other in the procedure. We could have used different names in both objects, but for the communication and sharing of data Video recorded with to be correct, the data type of the caller and called objects must be compatible.

Now, our procedure object is ready to receive a country identifier, in this case from the EnterAttractionsFilter web panel.

Parm rule

For an object to be able to receive values (parameters), the Parm rule must be used.

AttractionsList ✕

Source | Layout | **Rules** | Conditions | Variables | Help | Documentation

```
1 Parm(in: &CountryId);
2
```

It indicates that a value is received in this variable.

AttractionsList ✕

Source | Layout | Rules | Conditions | **Variables** | Help

| Name	Type
& Variables	
& Standard Variables	
• CountryId	Attribute:CountryId

We change the Source:

```
print Title
print ColumnTitles
For each Attraction order CountryName
    where CountryId = 2
    print Attractions
endfor
```

We change it to →

```
print Title
print ColumnTitles
For each Attraction order CountryName
    where CountryId = &CountryId
    print Attractions
endfor
```

Now we only have to remove the fixed filter that we had (country value 2) in the For each command and change it for the variable whose value is received as a parameter.

Object A

Object B
(AttractionsList)

Parm   &CountryId

&CountryId

AttractionsList (CountryId)   Call

AttractionsList (2)

```
AttractionsList * ×
Source * | Layout | Rules * | Conditions | Variables * |
Subroutines
1    print Title
2    print ColumnTitles
3  □ For each Attraction order CountryName
4        where CountryId = &CountryId
5        print Attractions
6  └ endfor
7
```
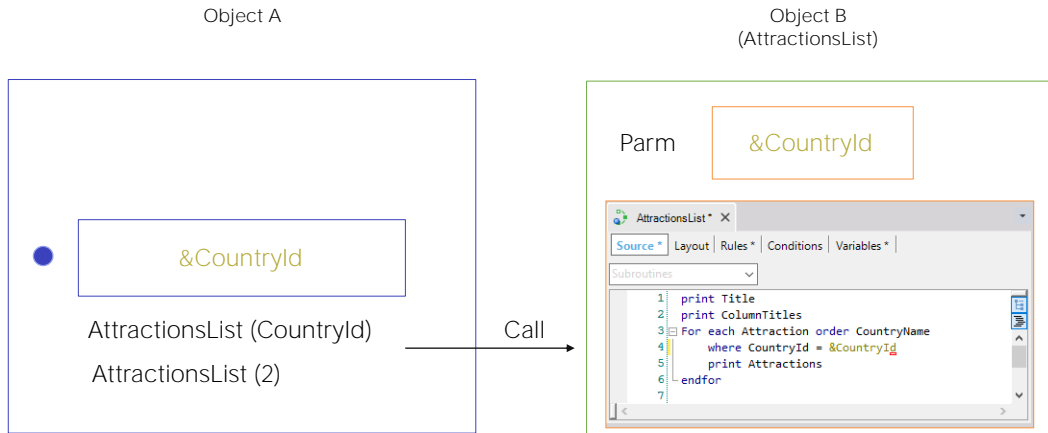
Note that since the Parm rule has been stated in this way, from now on any object that calls the procedure will be able to (and will have to) send the country identifier value. It will no longer be possible to call this procedure without sending it a value of this type. That's why the AttractionsList procedure will no longer be displayed in the Developer Menu.

In the web panel case we had this value in a variable (that user entered in screen). But if we had the data in an attribute, we would include the corresponding attribute between the brackets.
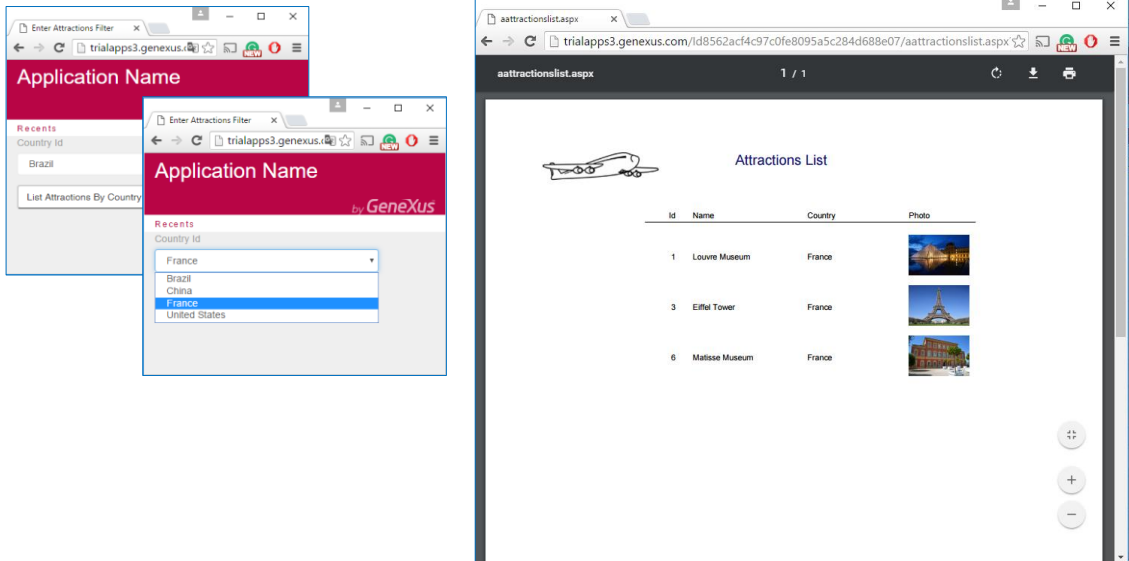
We may also send a value.
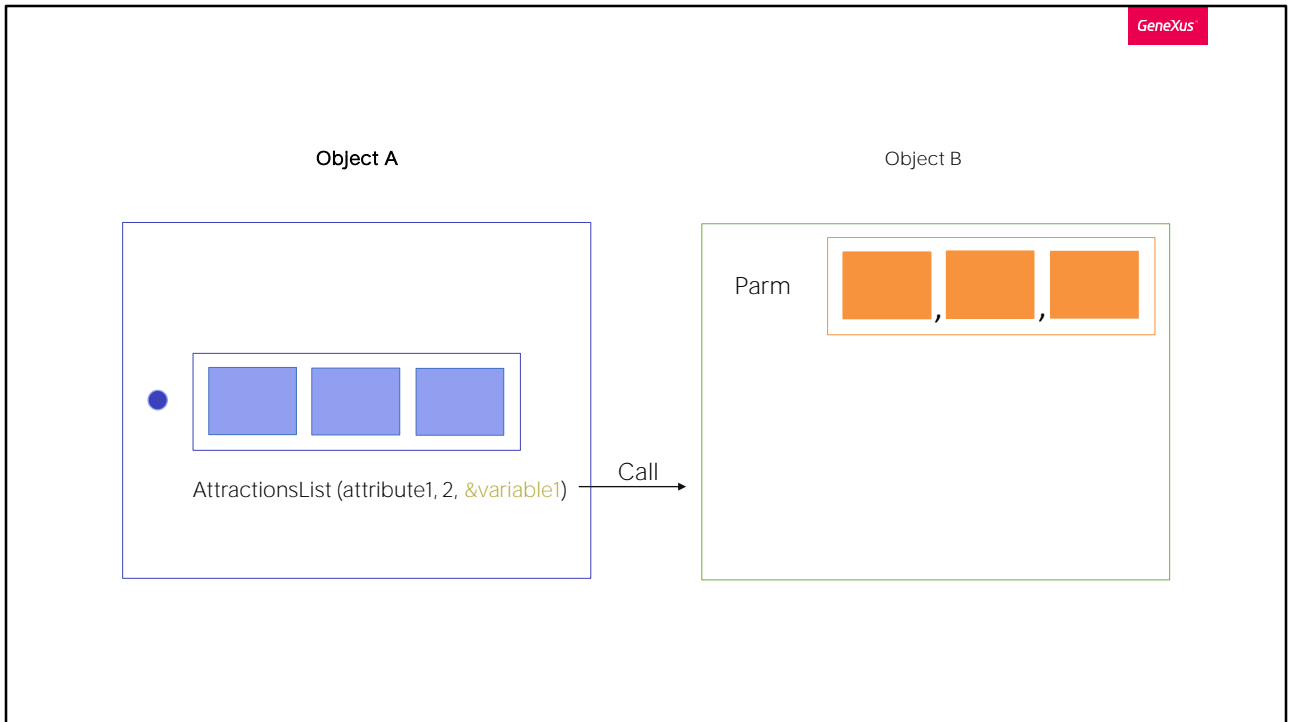
Or, if we had to send two or more values, we would send several attributes, and/or explicit values, and/or variables separated by commas.

These parameters are also declared in the parm rule in an ordered manner, separated by commas.

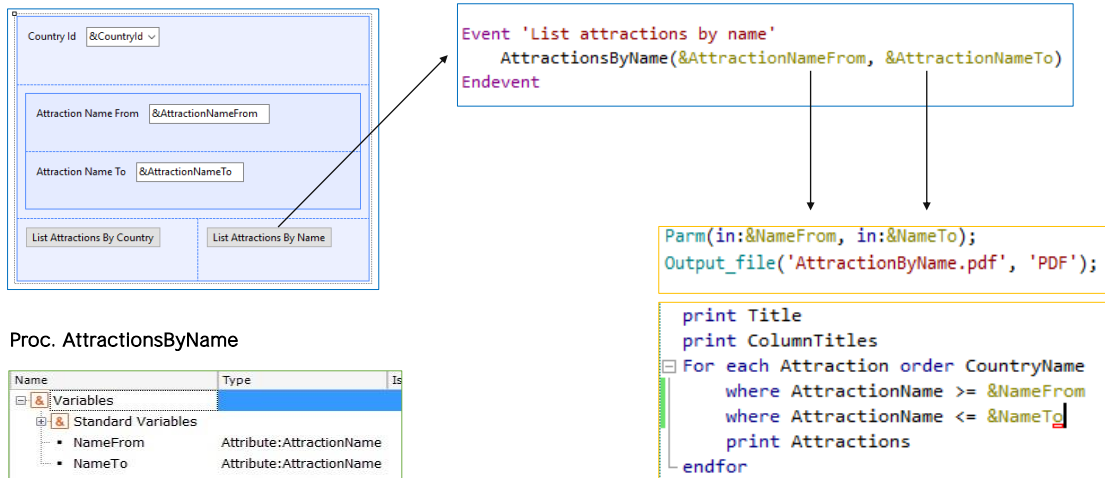Obviously, an object that doesn't receive parameters must not declare the Parm rule.

We try what we have done by pressing F5. We see that the AttractionsList procedure is no longer displayed. Now we can only call it through the web panel...

In the country combo, we choose France... and press the button.

By choosing the value France from the dynamic combo, the identifier value of France was internally selected (in this case, value 2); that value is sent to the AttractionslList procedure.

We see that the report is run, showing only the attractions of the country France.

List attractions in a specific name range

```
Country Id  &CountryId ∨

Attraction Name From   &AttractionNameFrom

Attraction Name To    &AttractionNameTo

List Attractions By Country    List Attractions By Name
```

```
Event 'List attractions by name'
    AttractionsByName(&AttractionNameFrom, &AttractionNameTo)
Endevent
```

Proc. AttractionsByName

Name	Type	Is
⊟ & Variables		
⊞ & Standard Variables		
• NameFrom	Attribute:AttractionName	
• NameTo	Attribute:AttractionName	

```
Parm(in:&NameFrom, in:&NameTo);
Output_file('AttractionByName.pdf', 'PDF');
```

```
 print Title
 print ColumnTitles
⊟For each Attraction order CountryName
    where AttractionName >= &NameFrom
    where AttractionName <= &NameTo
    print Attractions
└endfor
```

Now, let's suppose that we want to list all the attractions whose names **match a range of values selected by the user. For example, between "A" and "D".**

To do so, to the web panel that we previously created we will add the possibility for the user to enter a start name and an end name. In this way, pressing a button will call a list to show all the tourist attractions whose names are within that range.

We open the EnterAttractionsFilter web panel and add a table with two variables:

&AttractionNameFrom, is based on the AttractionName attribute, and &AttractionNameTo, which is also based on the definition of AttractionName.
As we've said before, this means that the variable definition is linked to the attribute definition, and if we change the attribute data type, the variable will be automatically changed accordingly.

Next, we add an event button called "List Attractions By Name". We click on the button we've just added, right-click and select Go to event. From here we need to call the procedure that will print the tourist attractions

within the selected range.

We click on the button we've just added, right-click and select Go to event. From here we need to call the procedure that will print the tourist attractions within the selected range.

We already had the AttractionsList **report... but it received in a parameter** the country identifier, not the name range. We will save it with another name, AttractionsByName, and change its Parm rule, so that now it receives two input parameters: The &NameFrom variable and &NameTo variable.

We have to define them as variables and set their data types, so we right **click on the first one and select "Add Variable".**
When we edit its properties, we base it on the AttractionName attribute.

We do the same with &NameTo.
Now we will use these variables/parameters in the For Each command, to keep the attractions that meet the condition that their AttractionName is greater than or equal to the value of the &NameFrom variable, as well as lower than or equal to the value of the &NameTo variable.

In this way, the procedure is ready and we only have to call it from the web panel.
We open the web panel and add the invocation. We drag the AttractionsByName report from this window to avoid having to type it; between brackets we type the parameters separated by commas, which in this case are the variables &AttractionNameFrom and &AttractionNameTo that are offered to the user on the screen.
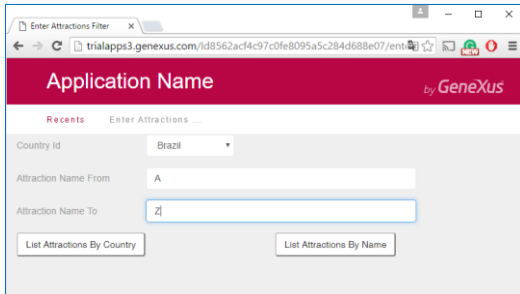
Note that the first parameter we wrote in the call will be loaded in the first parameter defined in the Parm rule of the called object, and the second parameter of the call will be loaded in the second parameter of the invoked object.

We must pay attention to the order used in the invocation and definition of the Parm rule. It's good practice to use related names as we've done here, in order to better understand the code.

Note that the names we've chosen for the variables in the web panel and in the procedure are different. As we've said before, what's most important is that the data types sent and received must match.
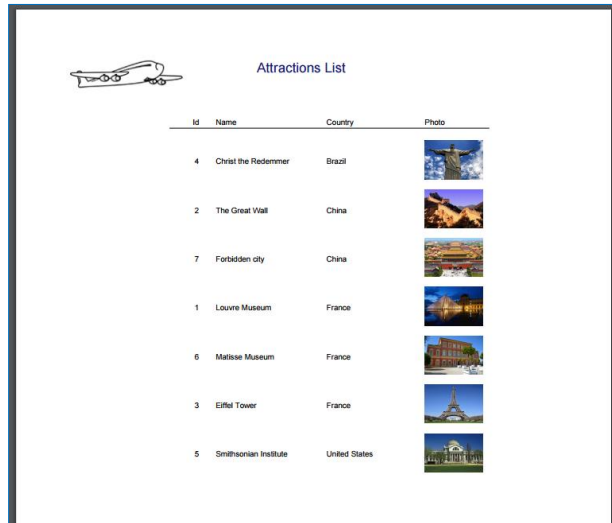
Let's press F5 to run it.

We open the web panel, and first of all we want to see all the attractions between "A" and "Z".
We press the button "List Attractions By Name"... and see that all attractions are listed.

Now we reduce the range a bit more by setting it between 'A' and 'F'.
We see how the filter has worked.

List attractions with a single button and a single procedure

Web panel

Country Id &CountryIld ∨

Attraction Name From &AttractionNameFrom

Attraction Name To &AttractionNameTo

List Attractions

Proc. AttractionByNameAndCountry

Name	Type
⊟ & Variables	
⊞ & Standard Variables	
● CountryId	Attribute:CountryId
● NameFrom	Attribute:AttractionName
● NameTo	Attribute:AttractionName

```
Output_file("AttractionsList.pdf", "pdf");

parm(in: &CountryId, in: &NameFrom, in: &NameTo);
```

```
print Title
print ColumnTitles
For each Attraction order CountryName
    where AttractionName >= &NameFrom when not &NameFrom.IsEmpty()
    where AttractionName <= &NameTo when not &NameTo.IsEmpty()
    where CountryId = &CountryId when not &CountryId.IsEmpty()
    print Attractions
endfor
```

```
Event 'List Attractions'
    AttractionsByNameAndCountry(&CountryIld, &AttractionNameFrom, &AttractionNameTo)
Endevent
```

We could also have implemented this with one procedure instead of two; let's see how.

First of all, the following must be done: return to the EnterAttractionsFilter web panel, configure the Empty Item of the CountryId variable to True, and in the Empty Item Text property enter "Select." This way, whenever this panel is accessed, there will not be a default country selected as before, but the text "Select" will appear, and one will have to be selected.

Now we create a procedure that will be a mix between AttractionsByName and AttractionsList. To facilitate its creation, we make a "Save As..." of AttractionsByName, and call it AttractionsByNameAndCountry.
In this new procedure, we must receive three parameters, which will correspond to the three filters that we have created in the web panel. So, in addition to those already entered, the CountryId variable is added, and included in the list of variables of this procedure.

In the Source section, we add another Where clause, to see the filter by country, where CountryId is equal to the CountryId variable.
Now we only have to create the new button in the web panel to execute this procedure. We delete the two we had created, and generate one named List Attractions.

We right-click on it and select Go To Event to program its behavior.

We comment the previous events since we are not going to use them, and from the new event we invoke the new procedure object created, AttractionsByNameAndCountry, passing by parameter the three variables that we have created and that are offered to the user on screen, CountryId, AttractionNameFrom and AttractionNameTo.
We save the changes and run the application to test the operation.

We choose for example the country France, to show only the attractions with names starting with the letters "F" to "Z."
We press the List Attractions button and see that the filter works correctly. If we select the country France again without entering filters by name because we are interested in all the attractions of this country, if we press the button, we can see that no attraction appears in the list. The same is true if we do not choose any country, and we want to filter the attractions by name only, regardless of the country. Suppose we want to see the attractions between "A" and "T." We press the List Attractions button and see that the list is empty. Why?

This is because in the Where clauses of the procedure, we do not include the possibility that any of the values of the variables could be empty; that is to say, that the user does not enter any value in that filter.
To solve this, we use the When clauses.

By using the When clause following the definition of the Where clause, we indicate that we want to apply a restriction to the latter. In future videos we will see in more detail how it works.
For example, in the case of our first Where, we add at the end When not &NameFrom. isEmpty(); that is, when the variable NameFrom doesn't have an empty value.
We do the same for the other two conditions.

What we are doing here, is indicating that we want the Where condition to be applied only when the variable has some value assigned; that is, when it is not empty. Otherwise, this clause will not be applied and the following line will be taken.
Now we save and run again with F5.

We select the country France, and we will not apply filters by name. We click on the button List Attractions.
The list of all the attractions in France appears as expected.

Now we will try leaving the country unselected. We want to list the attractions from "A" to "G." We see that all the attractions that meet this condition are listed, regardless of the country, which is exactly what we were looking for.

As we have just seen, doing it this way, we can filter only by country, only by name, or by country and name. In addition, it has only one procedure and only one button in our web panel.

We send everything we have done to GeneXus Server.
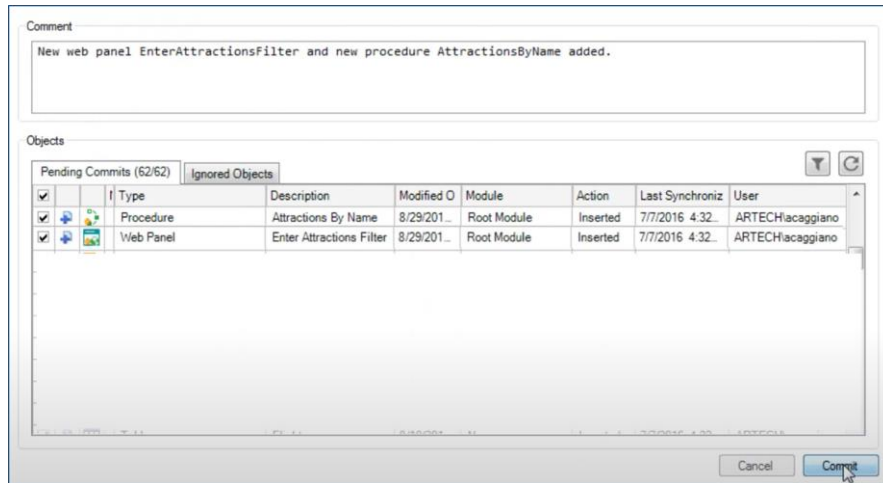
Lastly, we send everything we have done to GeneXus Server.

In the next video we will see other ways to send and receive parameters, including the effects of placing an attribute in the Parm rule instead of in a variable.