Subroutines

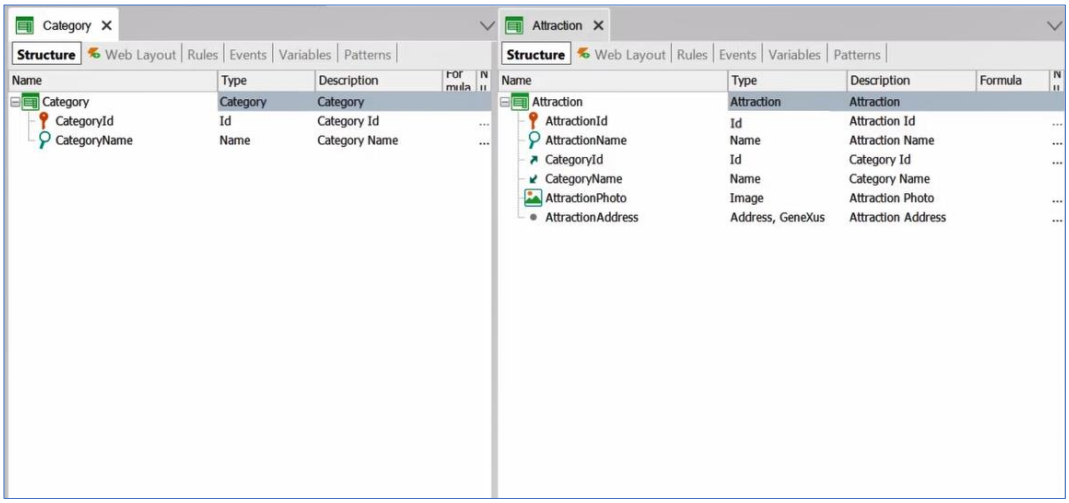
GeneXus™

In this video, we will see what subroutines are in GeneXus. We will see how they work and how they can be implemented, and we will examine what they can be useful for.

A subroutine is basically a block of code, which we can invoke as many times as we want, as long as it is within the same object.
In this way, we can execute the same block from several places of the object, or from the same place, but several times. The subroutine allows us to write that code only once, assign a name to it, and then simply invoke it by the given name.

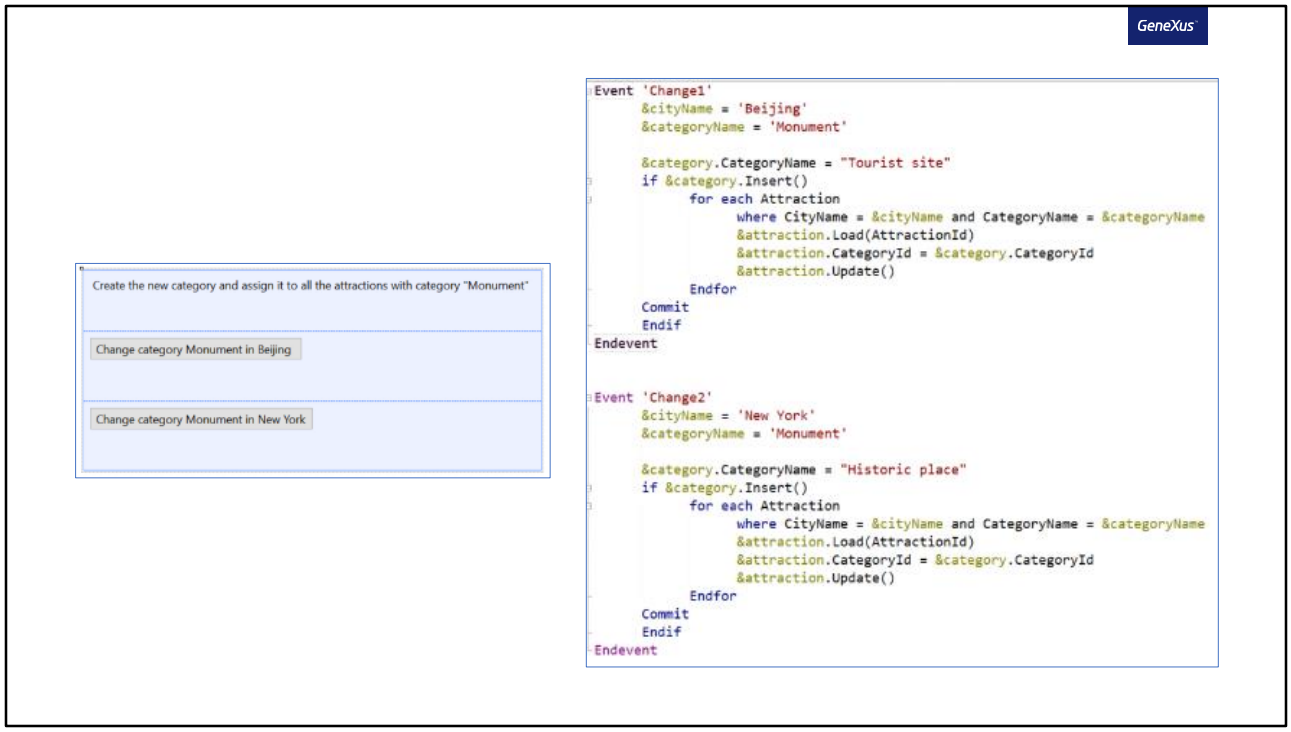
Subroutines can be used in all objects that accept programming, except for Data Providers. They are as follows: Web Panels, Procedures, Panels, Transactions.

Let's see how it works with a simple example.



We will do it on an application that we are developing for a travel agency; among its transactions we have one called Category, to register the different categories, and another one called Attraction. In the latter, in addition to its own attributes, it has the CategoryId and CategoryName attributes of the Category transaction.

Let's see one of the application's Web Panels, and examine its implementation.

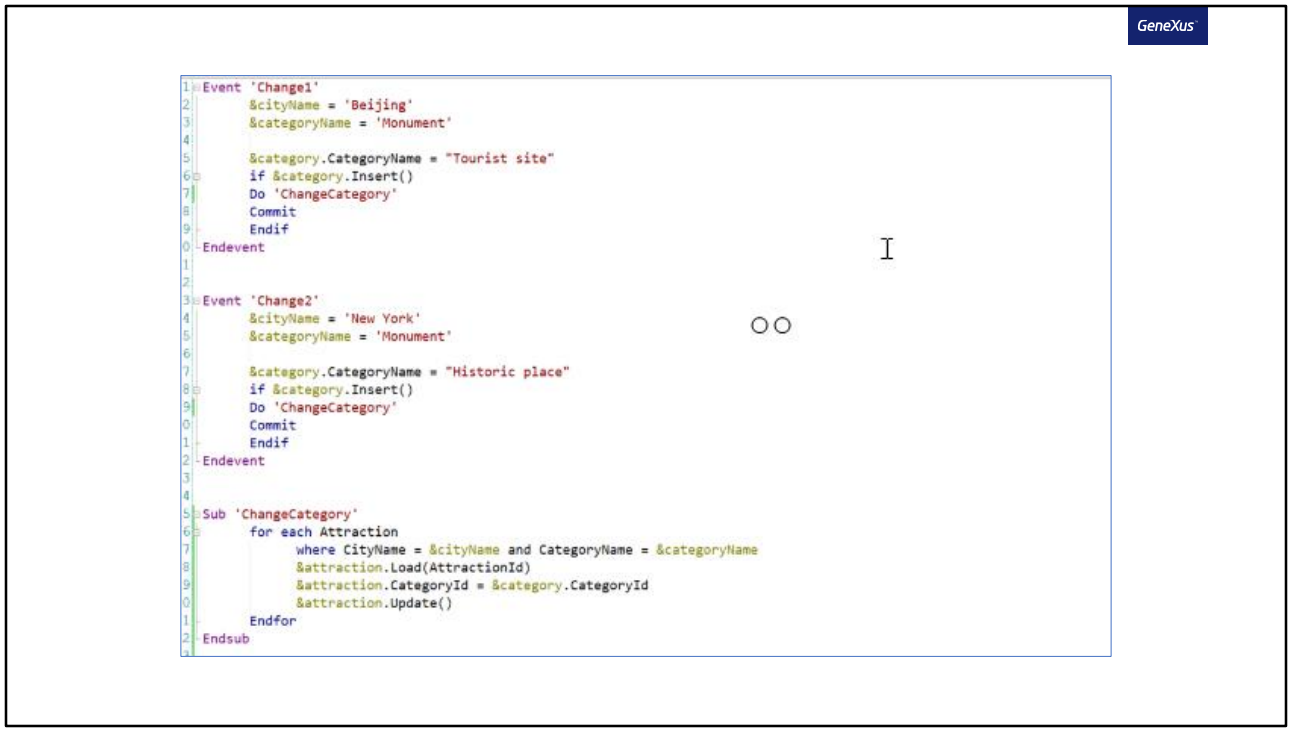


In the Layout we see two buttons, one named "Change category Monument in Beijing" and the other "Change category Monument in New York." The first one will have Change1 as event name and the second one will have Change2.

In the events section, we see the code assigned to these two events. Let's look at the first one. Two variables, cityname and categoryname, are declared and assigned the text Beijing and monument, respectively.
Then we have a category variable, of the Business Component Category type, which is told that the name of the category will be Tourist site. Subsequently, it will try to insert this new category in the Category table.
If the record was inserted correctly, the attraction table will be run through, and it will filter only the records in which CityName has the same value as the CityName variable and CategoryName has the same value as the CategoryName variable.
For these records, the attraction category will be updated, changing it to the one we have just entered, in this case Tourist Site.

The second button will do the same as the first one, but with different data. The name of the category to be inserted will be Historic Place, and New York's attractions that have Monument as category will be filtered.

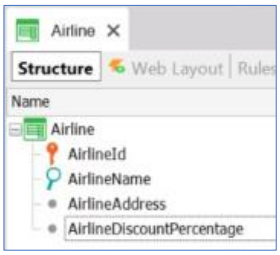
If we look at the first and the second event, they have exactly the same code block.
In this case, we could declare that code in a single place, and simply call it from the event we want.
Let's do it.



The Sub command will allow us to define a subroutine, and then we must assign a name to it. With endsub we mark where it ends.
Inside it we must enter the code that we will invoke later. We copy it, and paste it here.
We delete that block in the events, and call the subroutine instead. We do this by using the Do command, followed by the name.
With this implementation, the way it works will be exactly the same as before declaring the subroutine.

In this way, we manage to modularize our code, making it clearer and easier to read. Another advantage is that if we need to change something in this block, we do it only once and it is applied everywhere it is called. Also, we can reuse this code through the subroutine as many times as we want, as long as it is within this same object, in this case the Web Panel InsertCategoriesAndAttractions.

Now let's look at this other example, to understand a little more about how it works.

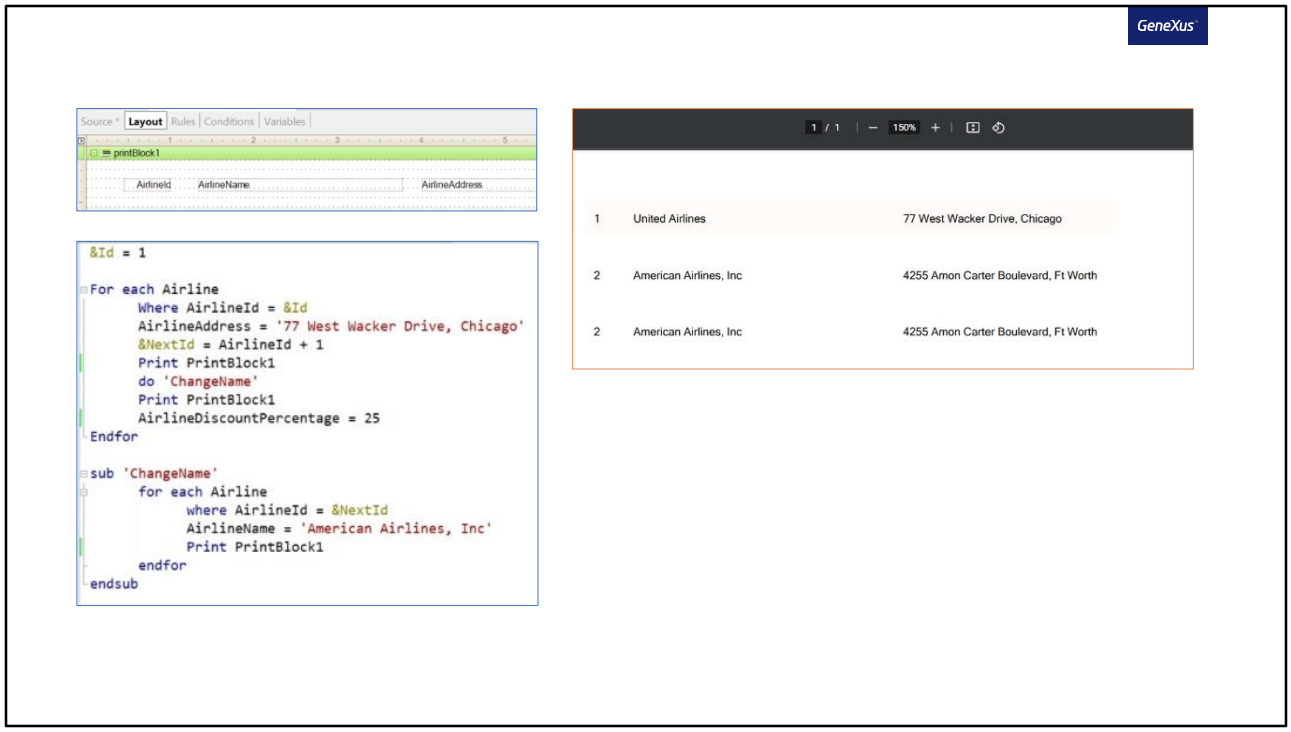



```
&Id = 1

For each Airline
      Where AirlineId = &Id
      AirlineAddress = '77 West Wacker Drive, Chicago'
      &NextId = AirlineId + 1
      do 'ChangeName'
Endfor

sub 'ChangeName'
      for each Airline
            where AirlineId = &NextId
            AirlineName = 'American Airlines, Inc'
      endfor
endsub
```

In our application, we have the Airline transaction to record the different airlines, with the following attributes.

In this procedure object, we have the following code in the source.
Through this For each, the Airline table will be run through, filtering by the airline with ID 1, since this is the value that we assigned to this variable.
We want to update the address of this record, using the AirlineAddress attribute.
Then we have a variable named nextId, to which we will assign the value of AirlineId plus one; in this case it will be two. Next, we call the ChangeName subroutine.
This subroutine runs a For each also on the Airline table, filtering by the record that has AirlineId equal to two.
If such a record is found, its name will be updated.

By the time the subroutine execution is finished, and we go back to the main For each, in which record will we be located? If here we have an attribute of the Airline transaction, for example this one (AirlineDiscountPercentage), and we assign it a value, to which record will it apply? To the record with ID one, which is where we were located before we called the subroutine? Or to the record with ID two?



Let's see this by declaring three attributes in the layout, to show the ID, name and address of the airline.

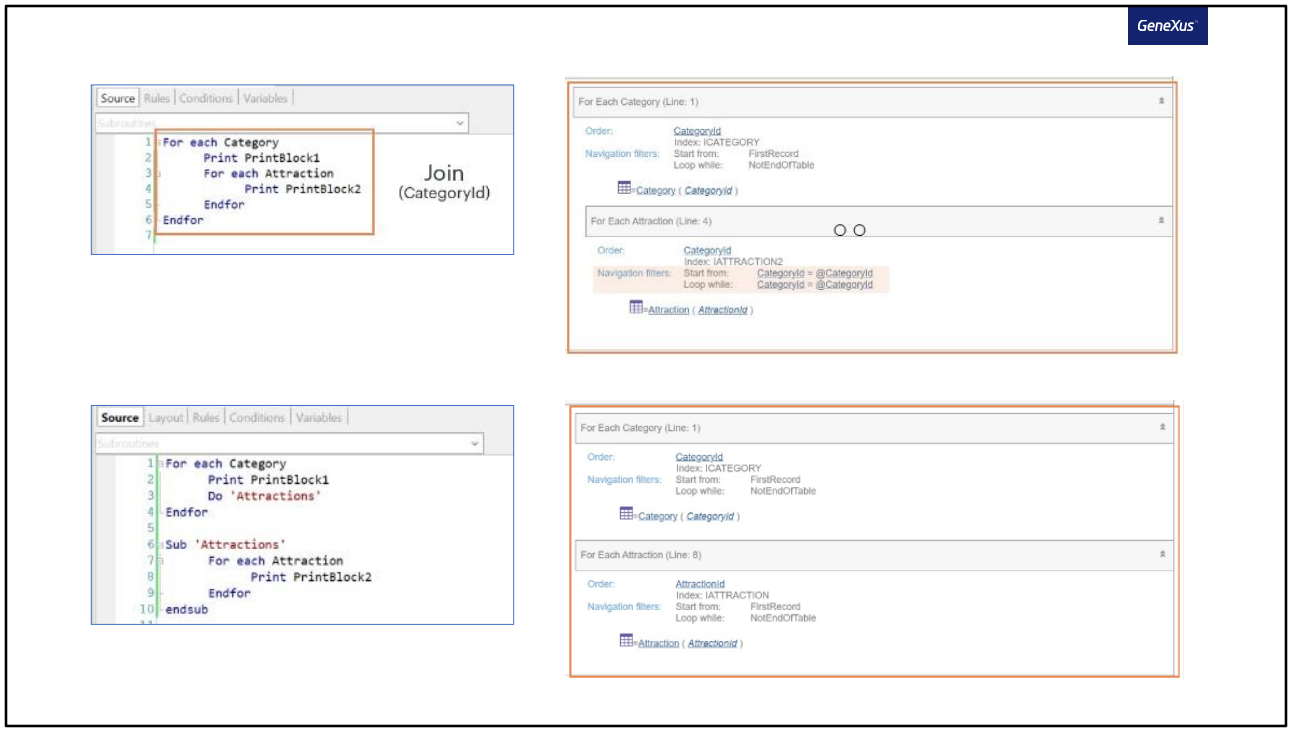
And in the source, we added three Print Printblocks to see in which airline we are positioned at any given moment. We place one before calling the subroutine, one during subroutine execution, and another one immediately after leaving the subroutine.

Let's try it.
We see that in the first print, before calling the subroutine, we are positioned in the airline with ID 1, already with the updated address.
The second print that is executed corresponds to the one inside the subroutine. We see that the record with ID two is printed on the screen, and already with the new updated name.
The third print is executed once we exit the subroutine. And we see that, at that moment, we are still positioned in the record with ID two, which is where we were inside the subroutine. And not in the record with ID one, which is where we were positioned before calling the subroutine.
Therefore, if at this moment we update the AirlineDiscountPercentage attribute with one value, it will be done on the record with ID two.
It works in this way because the declared attributes will be global to the object. Therefore, if an attribute takes value in a certain section of an object, and subsequently a subroutine is called that also assigns a value to the same attribute, when returning from the invoked subroutine and querying the attribute's value, it will have the value assigned in the subroutine. We've just seen this with the AirlineId attribute.

Subroutines don't support sending parameters; therefore, to exchange data we use variables, which are global to the objects.

If we didn't want this behavior we have just seen, instead of using a subroutine, we could call a procedure, for example. In this case, using a parameter to send the variable by which we want to filter.

Now let's look at a third example.



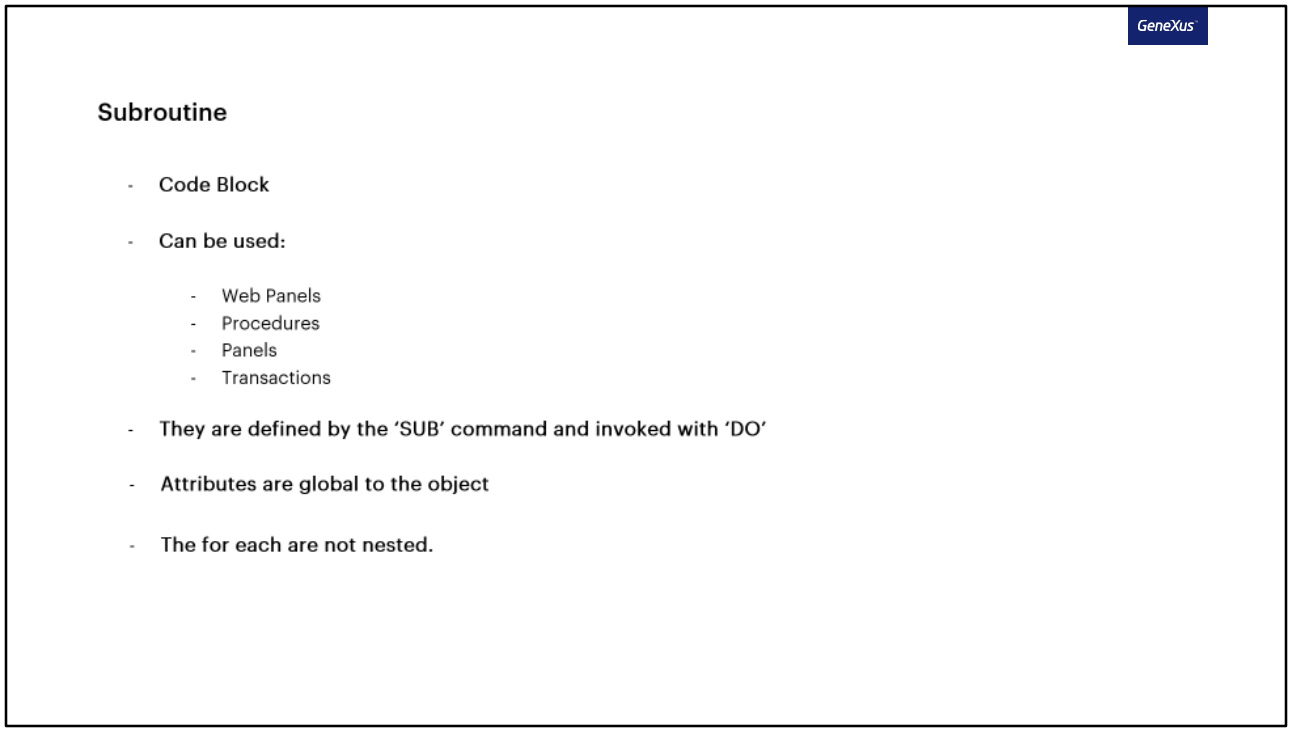
In this case, we have the following procedure, which performs a For each that navigates the Category table, and a nested For each that will navigate the Attraction table. As we know, in this case, GeneXus will perform a Join by CategoryId, since every attraction will have an assigned category. And CategoryId is the common attribute that will allow joining both tables. Let's see this in the navigation list.

Viewing the source of the procedure, we see that we implemented two PRINTS: one, implemented the name of the category, and the second, the name of the attraction.

If we run it, we see that the name of the category is printed, and inside it the attractions that have that category associated with it.
If we don't want this behavior, i.e. that the Join is not performed, but that the category is printed and then all the attractions are printed, regardless of which category they belong to... How can we implement it?
One option is to put this code inside a subroutine.
Let's check the navigation list.
We can see that the entire Category table is run through, and then the entire Attraction table is run through, from the first to the last record, without applying any type of filter.

Thus, GeneXus no longer performs the automatic inference, and doesn't filter by CategoryId. There will be two independent navigations.

So far, we have seen different examples about the use and operation of subroutines.



Here is a short summary:

Subroutines

- They are blocks of code, which allow us to modularize the code. They can be invoked as many times as we want within the same object.
- They can be used in Web Panels, Procedures, Panels, Transactions, etc.
- They are defined through the SUB command and then invoked through the DO command.
- They don't support sending parameters; variables are used to exchange data.
- If an attribute has a value, when calling the subroutine it changes. When returning from the subroutine and querying the value, it will have the value that was assigned in it, since the attributes are global to the object.
- If the call is made from within a For each and the subroutine also has a For each command, they will not be nested; i.e., no inferences or filters will be made.

For more information on this topic, you can visit our Wiki.


GeneXus™