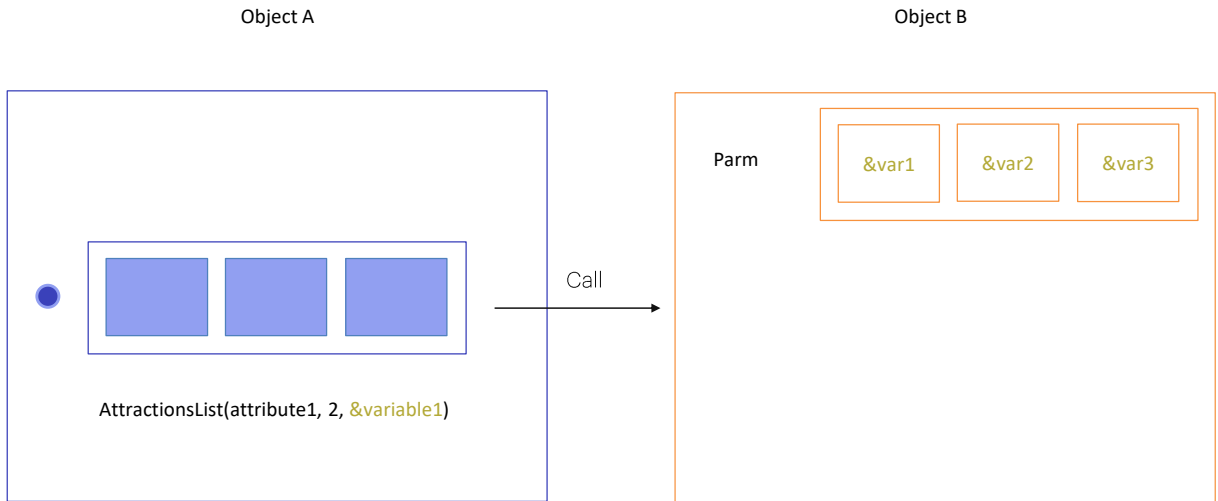


# Invocations between objects (Cont.)

GeneXus™



In the previous video we saw how to state parameters in an object to allow it to receive data from another object and perform the corresponding actions according to this data. To this end we used the Parm rules and variables. The examples we saw involved input parameters; that is to say, parameters only received by the object.

In this way, if object B has a Parm rule stated with three variables, to invoke object B, any object will have to send it three values that, as we've seen, may be saved in attributes, be an expression (as in the case of a fixed value), or be saved in variables.

Now we'll see what happens when object B must return a value to the caller object, when it ends running.

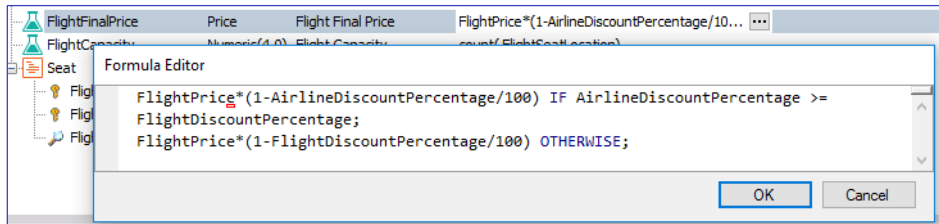
Name	Type	Description	Formula	Nullable
Flight	Flight	Flight		
FlightId	Id	Flight Id		No
FlightDepartureAirportId	Id	Flight Departure Airport Id		No
FlightDepartureAirportName	Name	Flight Departure Airport Name		
FlightDepartureCountryId	Id	Flight Departure Country Id		
FlightDepartureCountryName	Name	Flight Departure Country Name		
FlightDepartureCityId	Id	Flight Departure City Id		
FlightDepartureCityName	Name	Flight Departure City Name		
FlightArrivalAirportId	Id	Flight Arrival Airport Id		No
FlightArrivalAirportName	Name	Flight Arrival Airport Name		
FlightArrivalCountryId	Id	Flight Arrival Country Id		
FlightArrivalCountryName	Name	Flight Arrival Country Name		
FlightArrivalCityId	Id	Flight Arrival City Id		
FlightArrivalCityName	Name	Flight Arrival City Name		
FlightPrice	Price			
FlightDiscountPercentage	Percentage			
AirlineId	Id			
AirlineName	Name			
AirlineDiscountPercentage	Percentage	Airline Discount Percentage		
FlightFinalPrice	Price	Flight Final Price	FlightPrice*(1-AirlineDiscountPercentage/100) IF Airline...	
FlightCapacity	Numeric(4,0)	Flight Capacity	count( FlightSeat.Location)	
Seat	Seat	Seat		
FlightSeatId	Id	Flight Seat Id		No
FlightSeatChar	SeatChar	Flight Seat Char		No
FlightSeat.Location	Location	Flight Seat Location		No

Formula Editor

```
FlightPrice*(1-AirlineDiscountPercentage/100) IF AirlineDiscountPercentage >=
FlightDiscountPercentage;
FlightPrice*(1-FlightDiscountPercentage/100) OTHERWISE;
```

OK Cancel

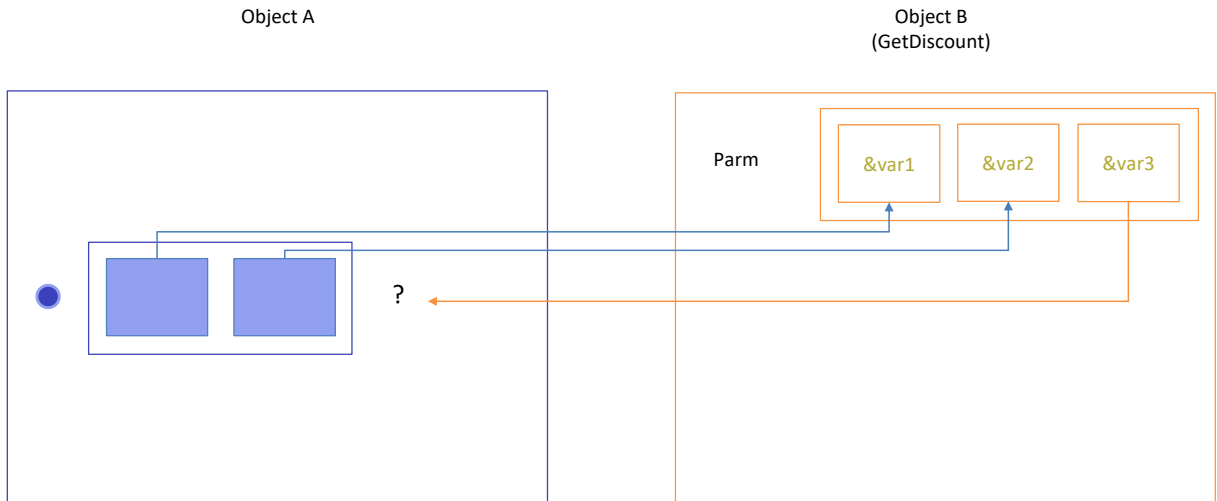
In the Flight transaction we had a formula that calculated the price of a flight according to the discount percentage offered by the airline and the percentage indicated for the flight itself. It selected the biggest discount and applied it.



Name	Type	Description	Formula	Nullable
Invoice	Invoice	Invoice		
InvoiceId	Id	Invoice Id		No
InvoiceDate	Date	Invoice Date		No
CustomerId	Numeric(4,0)	Customer Id		No
CustomerName	Character(20)	Customer Name		
CustomerLastName	Character(20)	Customer Last Name		
InvoiceTotalAmount	Price	Invoice Total Amount		No
Flight	Flight	Flight		
FlightId	Id	Flight Id		No
FlightPrice	Price	Flight Price		
FlightArrivalCountryName	Name	Flight Arrival Country Name		
FlightArrivalCityName	Name	Flight Arrival City Name		
InvoiceFlightDiscount	Percentage	Invoice Flight Discount		No
InvoiceFlightAmount	Price	Invoice Flight Amount		No

Suppose we're creating a transaction to record the invoices issued to customers when they purchase flight tickets.

Also, suppose that the discount is a more complex calculation that implies not only the flight, but also some condition related to the customer who is purchasing a flight ticket. For example, the number of tickets that he has purchased in the past, if he is a recurring customer, and if a destination is offered at a discount. The discount percentage is determined according to these more complex conditions.



For cases such as this, we may need to implement a procedure that makes these calculations and returns the resulting value to the caller.

For example, we could call this procedure GetDiscount.

The procedure will have to receive the customer and flight as input parameters.

It will return the resulting discount.

The first question is how this result is received by the object that needs the procedure result. It has to be considered as a function that is called in order to perform an action with the result returned.

Name	Type	Description	Formula	Nullable
Invoice	Invoice	Invoice		
InvoiceId	Id	Invoice Id		No
InvoiceDate	Date	Invoice Date		No
CustomerId	Numeric(4,0)	Customer Id		No
CustomerName	Character(20)	Customer Name		
CustomerLastName	Character(20)	Customer Last Name		
InvoiceTotalAmount	Price	Invoice Total Amount		No
Flight	Flight	Flight		
FlightId	Id	Flight Id		No
FlightPrice	Price	Flight Price		
FlightArrivalCountryName	Name	Flight Arrival Country Name		
FlightArrivalCityName	Name	Flight Arrival City Name		
InvoiceFlightDiscount	Percentage	Invoice Flight Discount	GetDiscount( CustomerId, FlightId )	...
InvoiceFlightAmount	Price	Invoice Flight Amount		No

```

1 InvoiceFlightDiscount = GetDiscount( CustomerId, FlightId );
   &discount = GetDiscount (CustomerId, FlightId);
   msg ("Free Flight") if GetDiscount (CustomerId, FlightId) = 100;
   If GetDiscount (CustomerId, FlightId) > 10
     ....
     ....
   endif

```

One possibility is to assign the result to an attribute. For example, we could define the InvoiceFlightDiscount attribute in the Invoice transaction structure as a formula that is calculated by invoking GetDiscount.

In this way, the formula will be evaluated in every object where the InvoiceFlightDiscount attribute is mentioned. The GetDiscount procedure will be invoked and executed, and when it ends running, the returned result will be shown as the formula attribute value.

If we don't want to set this attribute as a formula, but rather we want it to be an attribute stored in the corresponding table, and have it stored with the procedure result only when the transaction is executed, we Type

In addition, the result of the procedure's execution could be assigned to a variable.

Also, it may not be assigned, but used in an expression instead. For example, to condition the triggering of a Rule.

Or of instructions in a procedure or an event.

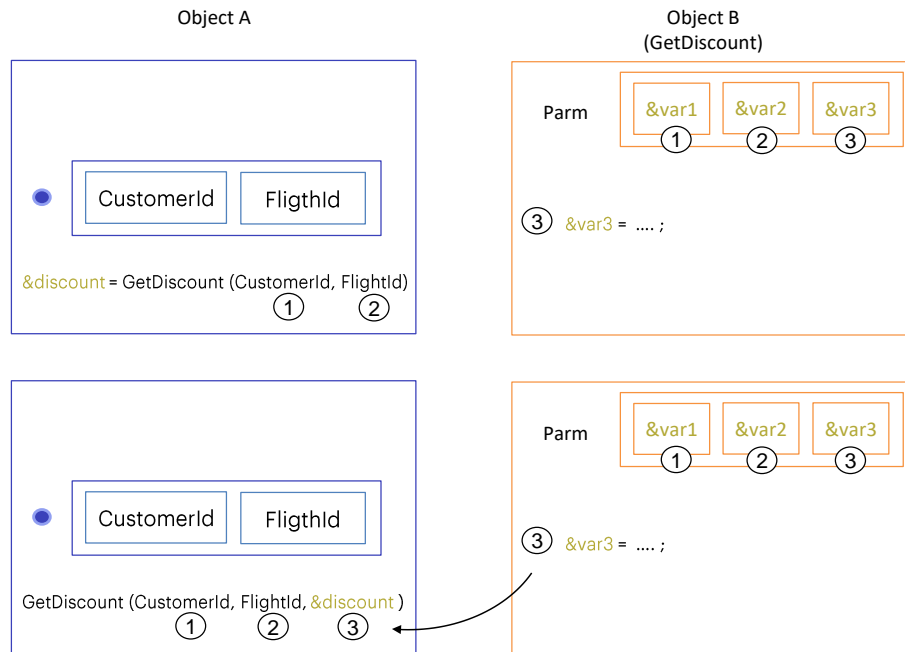
```
If GetDiscount( CustomerId, FlightId) > 10
```

```
...
```

```
Endif
```

We will not talk about how to implement the GetDiscount procedure, because it isn't relevant for what we're studying now. However, we must

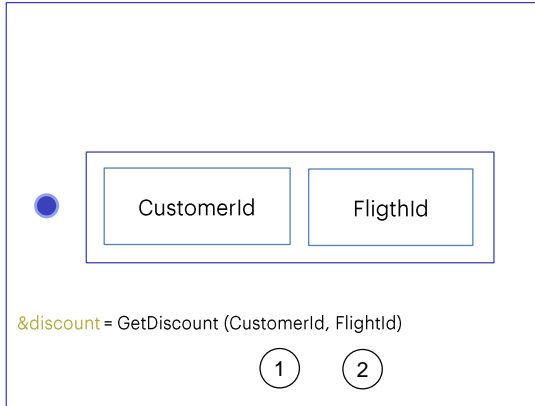
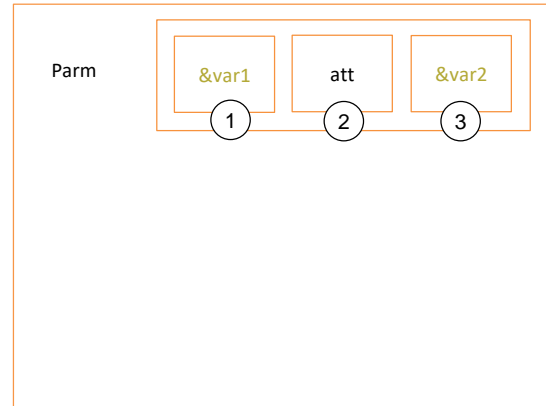
see how the parm rule is stated in the called object when the call syntax assumes that the object returns a value, as in the examples that we've just mentioned.



In the rules section of the GetDiscount procedure we must state the Parm rule with the number of parameters described in the call. Plus one at the end that will have to be a variable whose value is loaded in the object code (and in our case, in the procedure Source). The value taken by the variable when the code ends running will be the value returned to the object that called it.



Object A

Object B  
(GetDiscount)

Lastly, let's look at the case where a parameter of the `Parm` rule is an attribute instead of a variable.

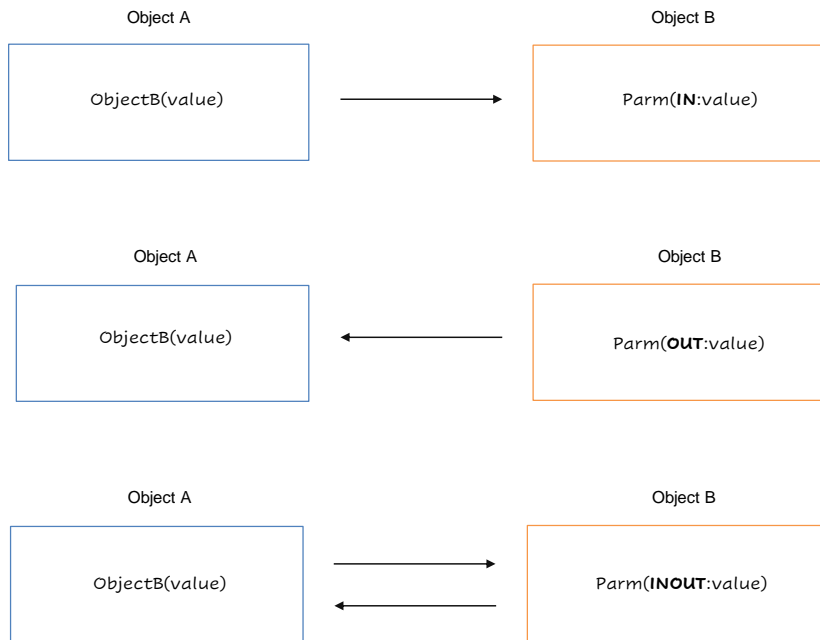
What's the difference between using a variable or an attribute in the `Parm` rule of the invoked object?

## Object B



```
parm (inout: &var1, in: &var2, out: &var3);
```

If the value is received in a variable, it can be freely used in the programming: it can be used as a filtering condition for filters such as equality, higher than, higher than or equal to, and so on... also, it can be used for an arithmetic operation, or for whatever is necessary. It's a space in memory with a name that we use within the object through explicit instructions, to do what we want.



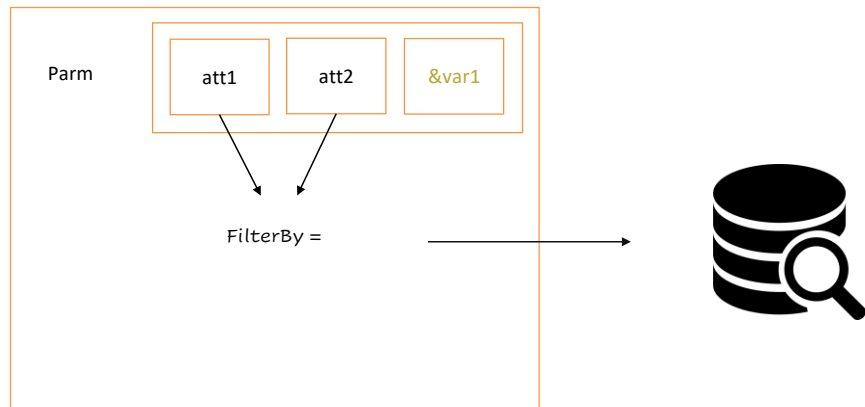
For each parameter declared in the Parm rule, you can define whether the parameter is used to receive a value, to return a value, or both. This is done by means of in, out, and inout operators, respectively.

The **IN** operator indicates that the parameter is an input parameter; that is, the parameter comes with a value and that value cannot be changed.

Parameters with **OUT** operator are output parameters. They don't bring any values and after the called object is executed, the output parameter will contain the resulting value that will be returned to the calling object.

Lastly, an operator called **INOUT** makes the parameter input and output at the same time. With this operator, the parameter comes with a value and can be changed during the execution of the object. When finished, the parameter will contain the value that is returned to the object that called it.

If none of these operators is declared in the parm rule, GeneXus will assign the INOUT operator to all parameters, even if this is not displayed.



If, on the other hand, the value is received in an attribute, this is fixed, determined, and implicit. We receive values in an attribute when we access the database from inside the object. In particular, a table in whose extended table this attribute is stored. So, when a value is received in that attribute through a parameter, an equality filter will be applied. Only the records that have that value for the attribute will be considered. Let's see this with an example.

```

1 param( in: &CountryId );
2
3 Output_file('AttractionsList.pdf', 'pdf');
4
5

```



```

Subroutines
1 print Title
2 print ColumnTitles
3 For each Attraction order CountryId
4   where CountryId = &CountryId
5   print Attractions
6 endfor

```

```

1 param( in: CountryId );
2
3 Output_file('AttractionsList.pdf', 'pdf');
4
5

```



```

Subroutines
1 print Title
2 print ColumnTitles
3 For each Attraction order CountryId
4   print Attractions
5
6 endfor

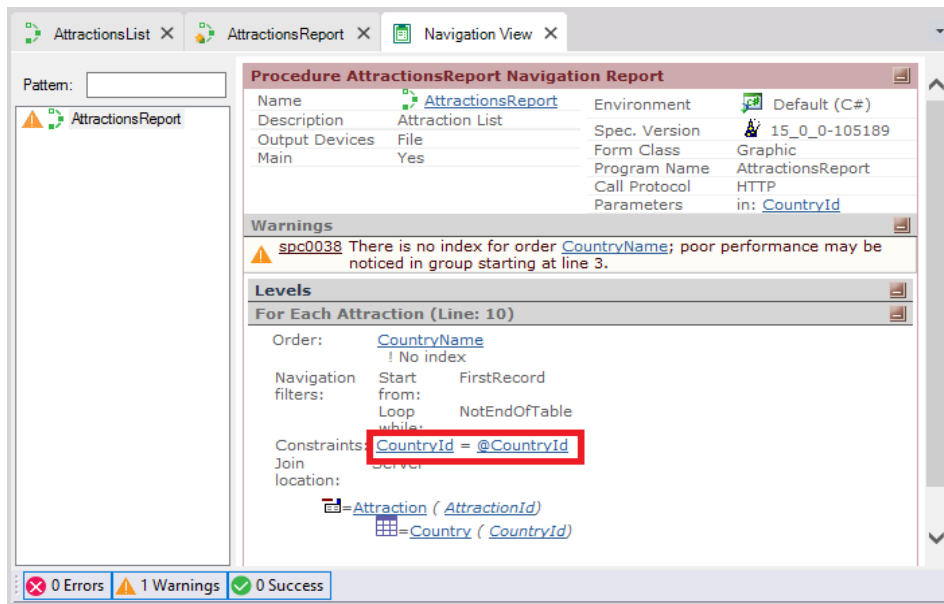
```

We make a copy of the AttractionsList procedure with the AttractionsReport name. Remember that the procedure on which it is based uses a variable as a parameter: &CountryId. It used it to filter the attractions in the Attraction table by country.

As we can see, it is implementing an equality filter: it will list only those attractions whose CountryId matches the value of the &CountryId variable received in a parameter. We could have implemented exactly the same without explicitly indicating that filter.

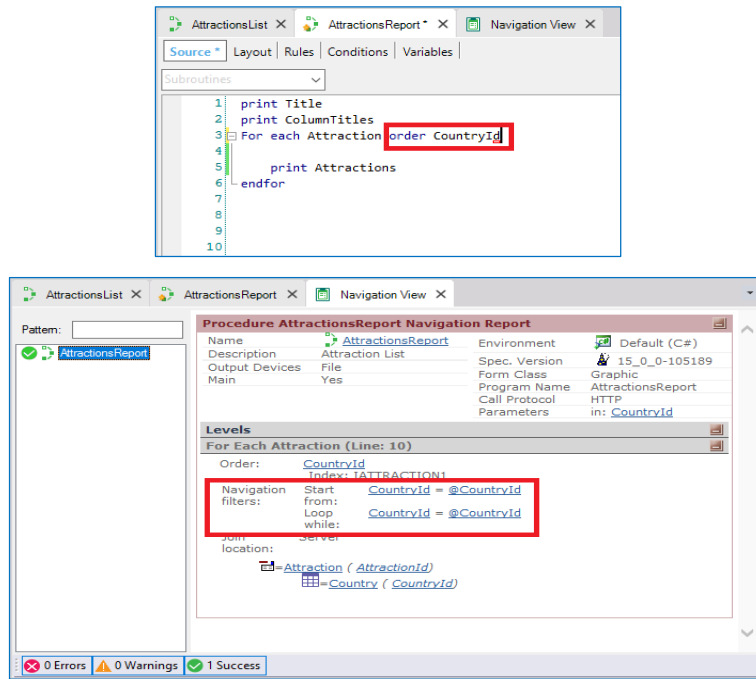
How? By receiving the value directly in the CountryId attribute. When we receive the value in an attribute in the Parm rule, GeneXus uses an equality filter; that is to say, only those records that have the same country identifier are accessed.

If we look at this object's navigation list



we see that the filter is applied even if the Where clause is not written.

It's interesting to note that since the For Each command is being run through ordered by CountryName, the entire table has to be run through to filter by the CountryId values corresponding to the parameter.



On the other hand, if we order by the attribute used to filter. We see in the navigation list that the entire table is no longer run through. We run it... And upload everything to GeneXus Server.

If we received more than one value using attributes to receive them, only the records that have the same value as each attribute received would be accessed.

Also, we can't change these attribute values.

If our objective is not to receive values to apply an equality filter, the solution will be to receive values in variables instead of using attributes. In addition, they can be freely used in the programming, for example, to assign other values to them if necessary.

Communication between objects is essential for any GeneXus application, because it enables an object to start running another object and send or receive information to and from it.

There are other ways of communication between objects in which there is no passing of data by parameters.

One example of this occurs when information persists in memory through WebSession variables, or when global events are used. These cases will not be seen in this video, but it is important to know that there are several and diverse forms of communication between objects.

# GeneXus™

[training.genexus.com](http://training.genexus.com)  
[wiki.genexus.com](http://wiki.genexus.com)  
[training.genexus.com/certifications](http://training.genexus.com/certifications)