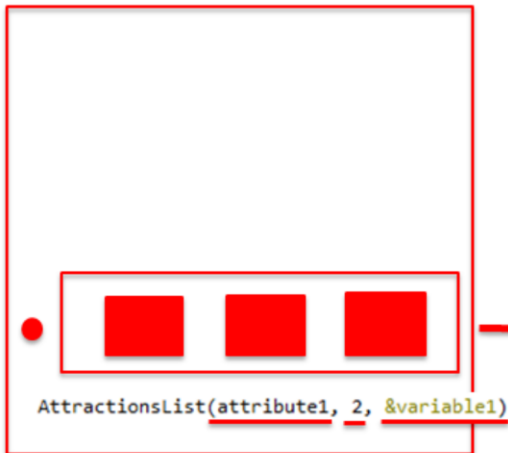


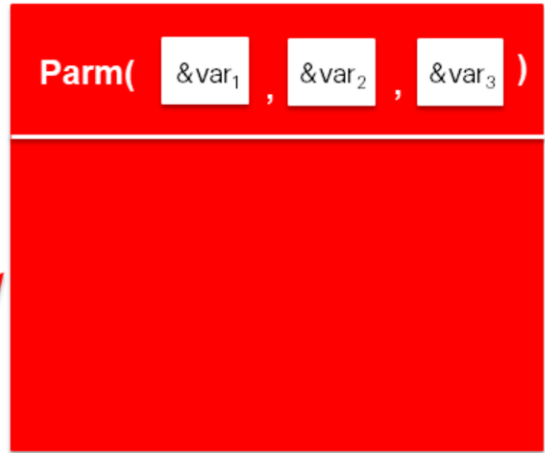
Cuando el objeto invocado devuelve un valor

GeneXus™ 16

Object A



Object B



Hasta aquí vimos cómo declarar en un objeto parámetros para permitirle recibir datos de otro objeto y tomar las acciones pertinentes de acuerdo a esos datos. Para ello utilizábamos la regla `Parm` y variables. Los ejemplos que vimos eran de parámetros de entrada, es decir, parámetros que el objeto únicamente recibe.

Así, si el objeto B tiene declarada una regla `Parm` con tres variables, todo objeto que quiera invocar al B deberá enviarle tres valores, que, como vimos, podían estar guardados en atributos, ser una expresión (como el caso de un valor fijo), o estar guardados en variables.

Ahora veremos qué sucede cuando el objeto B debe **devolver** un valor a quien lo llama, al finalizar su ejecución.

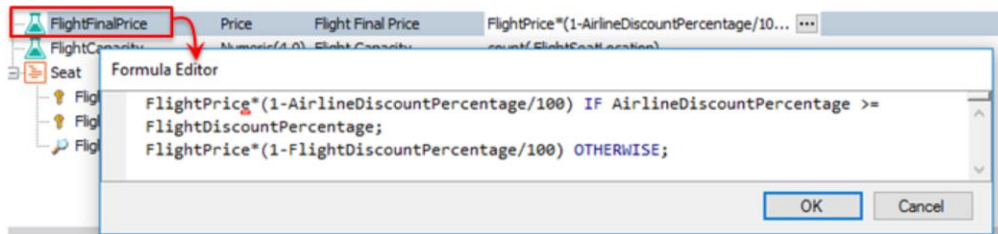
The screenshot displays the GeneXus IDE interface with the 'Flight' object selected. The 'Structure' tab is active, showing a tree view of the object's attributes. A red arrow points from the 'FlightFinalPrice' attribute in the tree to the 'Formula Editor' dialog box. The dialog box contains the following formula:

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

The 'Flight' object structure is as follows:

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			
FlightArrivalCountryName	Name			
FlightArrivalCityId	Id			
FlightArrivalCityName	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
FlightSeatLocation	Location	Flight Seat Location		No

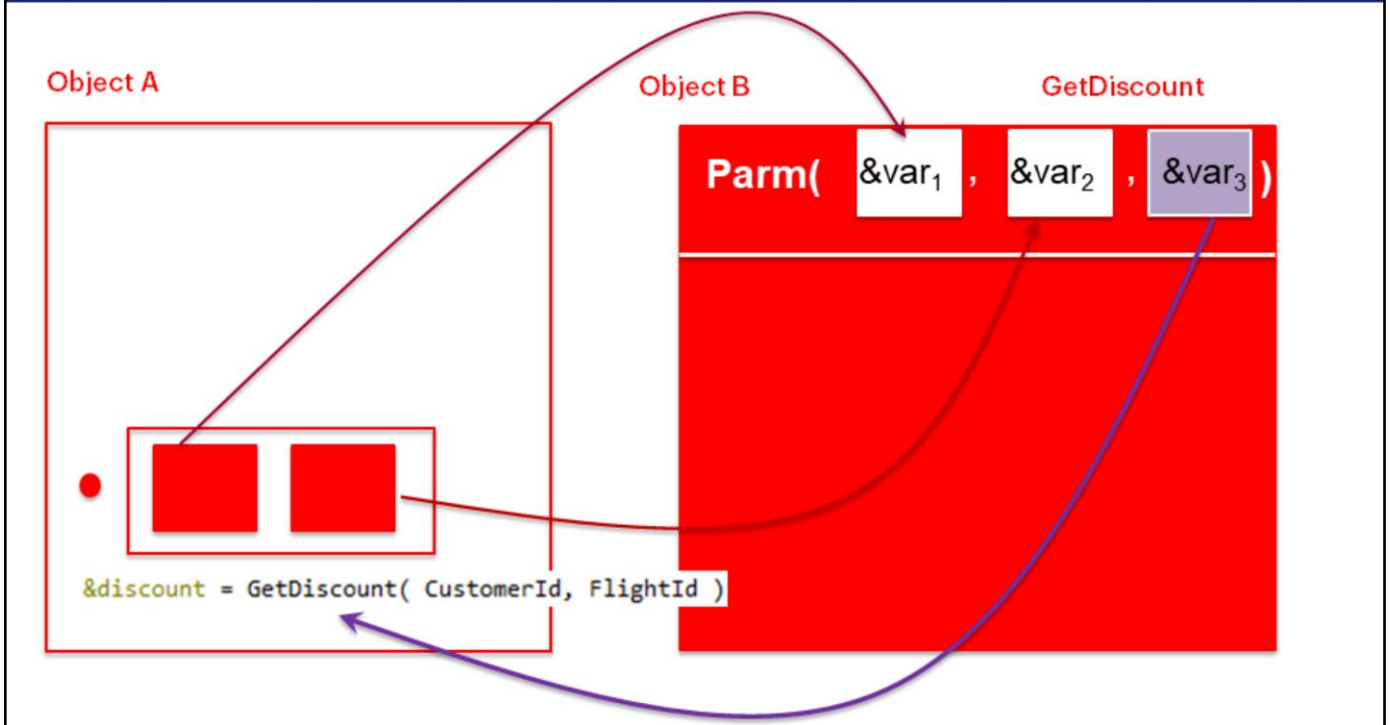
Teníamos en la transacción Flight una fórmula que calculaba el precio de un vuelo de acuerdo al porcentaje de descuento que brindaba la aerolínea y el porcentaje que se especificaba para el propio vuelo. Elegía el mejor descuento y ese era el que aplicaba.



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

Supongamos que estamos creando una transacción para registrar las facturas que se expiden a los clientes cuando compran vuelos.

Y supongamos que el descuento es un cálculo más complejo, que implica no sólo al vuelo, sino por ejemplo a alguna condición relativa al cliente que está comprando el vuelo. Por ejemplo, la cantidad de vuelos que ha comprado antes, qué tan buen cliente es, etcétera. Y, por ejemplo, si un destino está en oferta. Dependiendo de todas esas condiciones más complejas, se determina el porcentaje de descuento.

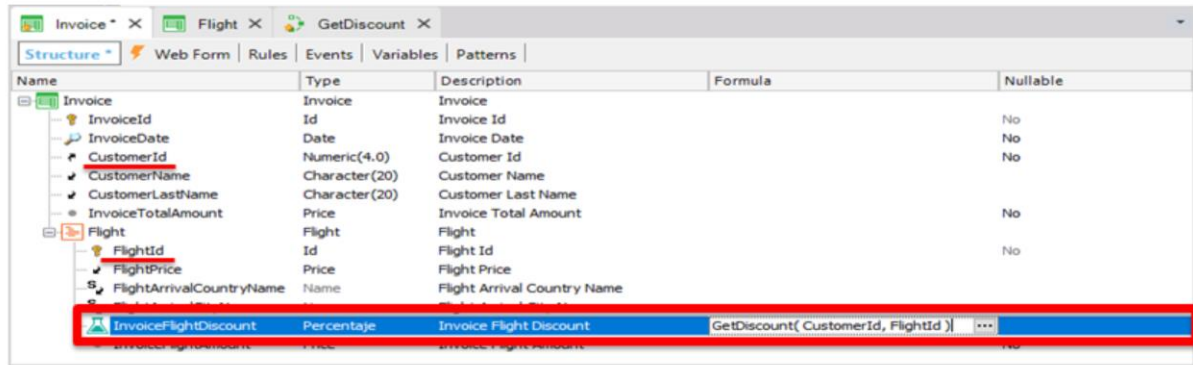


Para casos como este, podemos necesitar implementar un procedimiento que realice estos cálculos, y devuelva el valor resultante a quien lo llama.

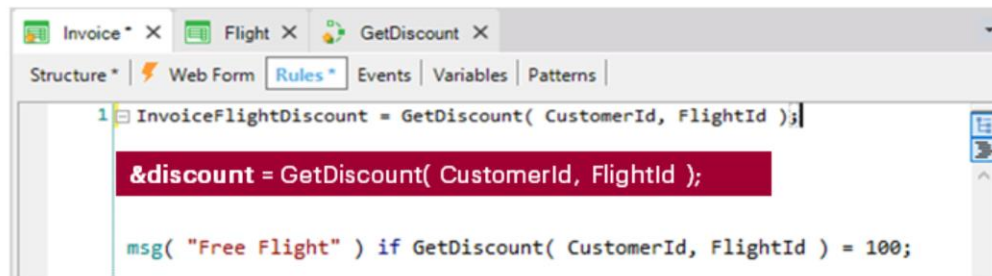
Por ejemplo, podríamos llamarle a nuestro procedimiento `GetDiscount`.

El procedimiento deberá recibir como parámetros de entrada el cliente del que se trata y el vuelo. Y devolverá el descuento resultante.

La primera pregunta es cómo recibe el objeto que necesita el resultado del procedimiento, ese resultado. Tenemos que pensarlo como una función, a la que llamamos y luego hacemos algo con lo que nos devuelve.



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		
InvoiceFlightDiscount	Percentage	Invoice Flight Discount	GetDiscount(CustomerId, FlightId)	



```

1 InvoiceFlightDiscount = GetDiscount( CustomerId, FlightId );

&discount = GetDiscount( CustomerId, FlightId );

msg( "Free Flight" ) if GetDiscount( CustomerId, FlightId ) = 100;

```

Una posibilidad es asignar el resultado a un atributo. Por ejemplo, podríamos definir al atributo FlightDiscount en la estructura de la transacción Invoice como una fórmula que se calcula invocando a GetDiscount.

De esta manera en todo objeto en el que se mencione al atributo InvoiceFlightDiscount se evaluará la fórmula, invocando al procedimiento GetDiscount, que se ejecutará, devolviendo, al finalizar, el resultado que será el que se mostrará como valor del atributo fórmula.

Si no queríamos definir a ese atributo como fórmula, sino que queremos que sea un atributo almacenado en la tabla correspondiente, y que solo al ejecutarse la transacción se almacene con el resultado del procedimiento, escribiríamos en las reglas la primera asignación que vemos arriba.

Pero también se podría asignar el resultado de la ejecución del procedimiento a una variable.

O incluso no asignárselo a nadie, sino utilizarlo en una expresión. Por ejemplo, para condicionar el disparo de una regla.

O de unas instrucciones en un procedimiento o en un evento:

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

```
...
```

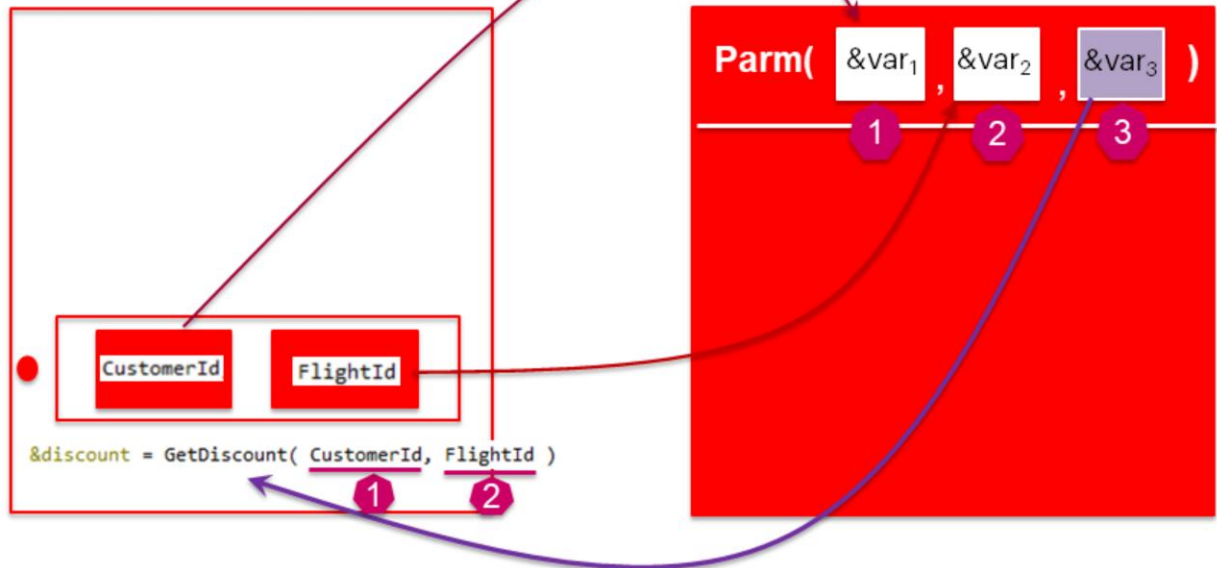
```
Endif
```

No veremos cómo implementar el procedimiento GetDiscount, pues no importa a los efectos de lo que estamos estudiando. Pero sí es importante que veamos cómo se declara en el objeto llamado **la regla parm** cuando en la sintaxis de la llamada se asume que el objeto devuelve un valor, como es el caso de los ejemplos que acabamos de señalar.

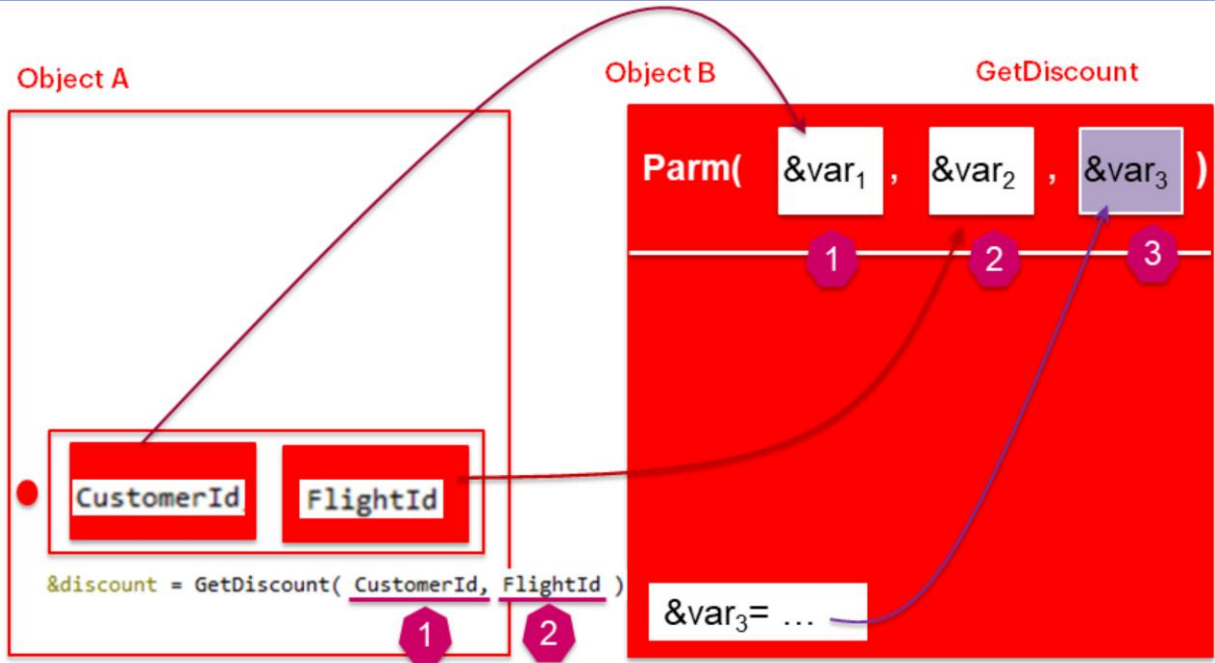
Object A

Object B

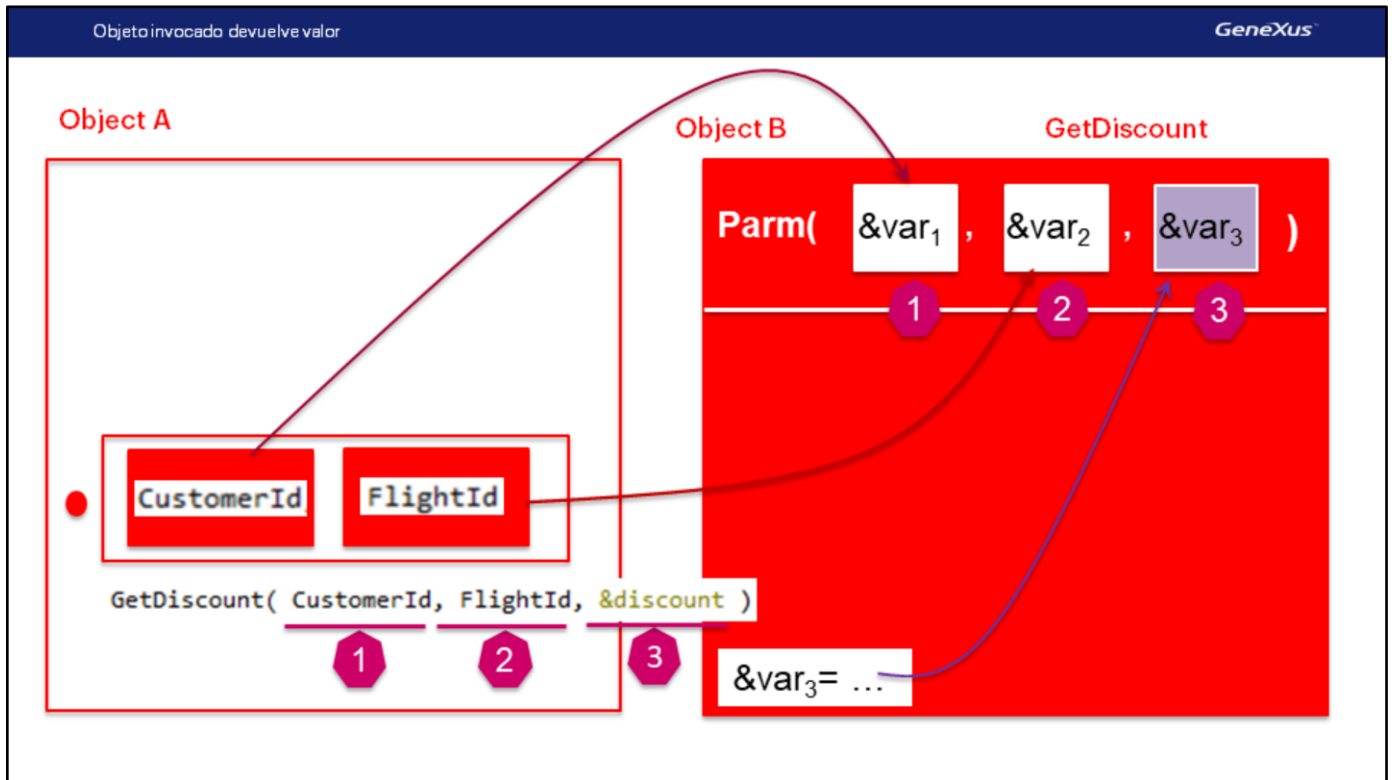
GetDiscount



En la sección de reglas del procedimiento **GetDiscount** deberemos declarar la regla **parm** con la cantidad de parámetros que se escriben en la invocación... más uno al final...



...que deberá ser una variable cuyo valor se cargue en el código del objeto (en nuestro caso, en el Source del procedimiento). El valor con el que quede la variable al final de la ejecución del código será el valor devuelto al objeto que lo llamó.



Para este mismo Objeto B podríamos haber realizado la invocación de la forma que veníamos viendo antes:

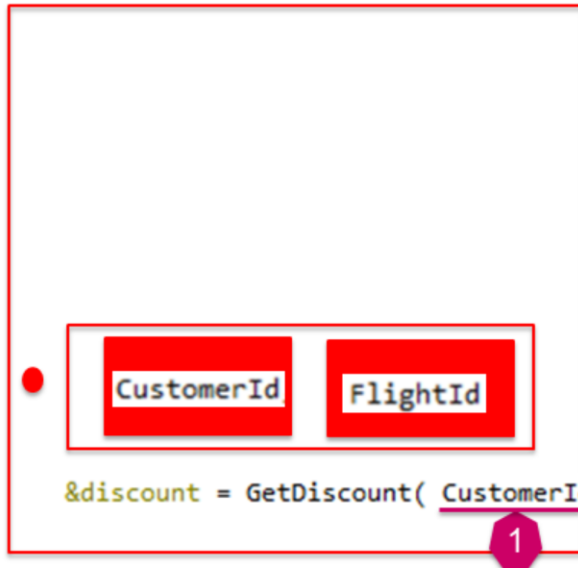
donde los primeros dos parámetros son de entrada y el último es de salida.

Sólo que de esta forma en la invocación no queda claro que `&discount` va a volver cargado. Dicho de otro modo, no se evidencia que el objeto invocado va a devolver un valor. Para estos casos suele convenir utilizar la sintaxis de invocación que sí lo hace evidente.

Recibir en variable o en atributo

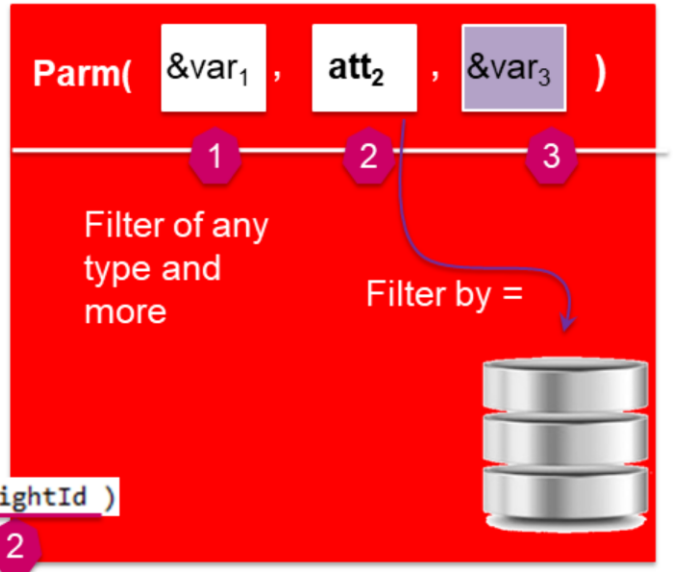
Por último, abordemos el caso en el que un parámetro de la regla Parm en lugar de ser una variable, es un atributo.

Object A



Object B

GetDiscount



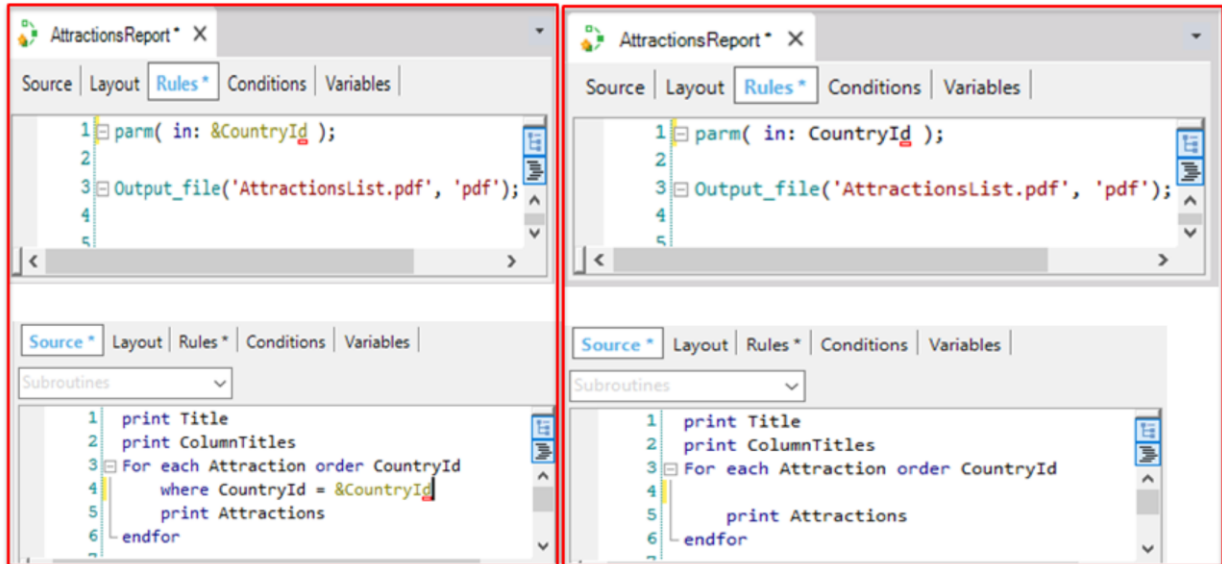
¿Cuál es la diferencia entre usar una variable o un atributo en la regla **parm** del objeto invocado?

Si se recibe el valor en una **variable**, la misma se podrá utilizar libremente en la programación: se la podrá utilizar como condición de filtro por igualdad, por mayor, mayor o igual, etcétera... se la podrá utilizar para alguna operación aritmética, o lo que se necesite. Es un espacio de memoria con nombre, que utilizamos dentro del objeto a través de instrucciones explícitas, para hacer lo que queramos.

Si en cambio se recibe el valor en un **atributo**, esto tiene un sentido fijo, determinado, implícito. Recibimos en un atributo cuando dentro del objeto vamos a acceder a la base de datos. En particular a una tabla en cuya extendida se encuentre ese atributo. Así, al recibir por parámetro un valor en ese atributo, se aplicará un **filtro por igualdad**. Sólo se considerarán los registros que tengan ese valor para el atributo.

Veremos esto con un ejemplo.

Ejemplo: recibir en variable vs recibir en atributo



Hagamos una copia del procedimiento AttractionsList, con el nombre AttractionsReport. Recordemos que el procedimiento en el que se basa utilizaba una variable como parámetro: &CountryId. Y la utilizaba para filtrar las atracciones de la tabla Attraction por país.

Vemos que se está implementando un filtro por igualdad: listará únicamente aquellas atracciones cuyo CountryId corresponda al valor de la variable &CountryId recibida por parámetro. Podríamos haber implementado exactamente lo mismo sin necesidad de establecer ese filtro en forma explícita. ¿Cómo? Recibiendo directamente en el atributo CountryId.

Cuando recibimos el valor en un atributo en la regla Parm, GeneXus filtra por igualdad, es decir, solamente se va a acceder a los registros que tengan ese valor de identificador de país.

Si pedimos ver la navegación de este objeto...

Listado de navegación

The screenshot shows the GeneXus Navigation View for the **AttractionsReport** procedure. The interface includes a left pane with a tree view showing the project structure, a central pane with the navigation report, and a bottom status bar.

Procedure AttractionsReport Navigation Report

Property	Value
Name	AttractionsReport
Description	Attraction List
Output Devices	File
Main	Yes
Environment	Default (C#)
Spec. Version	15_0_0-105189
Form Class	Graphic
Program Name	AttractionsReport
Call Protocol	HTTP
Parameters	in: CountryId

Warnings

- spc0038 There is no index for order CountryName; poor performance may be noticed in group starting at line 3.

Levels

For Each Attraction (Line: 10)

Order: CountryName
! No index

Navigation Start: FirstRecord

filters: from: Loop: NotEndOfTable

Constraints: CountryId = @CountryId

Join location: =Attraction (AttractionId)
=Country (CountryId)

Status Bar: 0 Errors, 1 Warnings, 0 Success

vemos que se está realizando el filtro aunque no esté escrito el Where.

Como hecho interesante y al margen, obsérvese que como el for each se está recorriendo ordenado por CountryName, se tiene que recorrer toda la tabla para filtrar por los valores de CountryId que correspondan al parámetro.

Navigation List

The screenshot displays the GeneXus IDE interface with three windows: AttractionsList, AttractionsReport, and Navigation View. The AttractionsList window shows a code editor with the following code:

```
1 print Title
2 print ColumnTitles
3 For each Attraction order CountryId
4
5     print Attractions
6 endfor
7
8
9
10
```

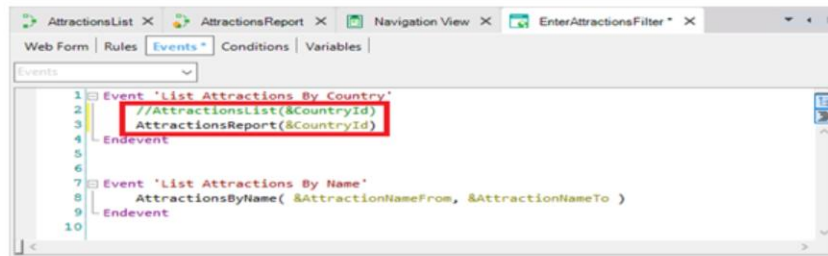
The AttractionsReport window shows the configuration for the report, including the Name, Description, Output Devices, and Main. The Navigation View window shows the configuration for the Navigation List, including the Order, Navigation, and Filters. The Navigation List configuration is highlighted with a red box, showing the following settings:

- Order: CountryId
- Navigation: Start: CountryId = @CountryId
- Filters: from: CountryId = @CountryId
- Loop: while: CountryId = @CountryId

The bottom status bar indicates 0 Errors, 0 Warnings, and 1 Success.

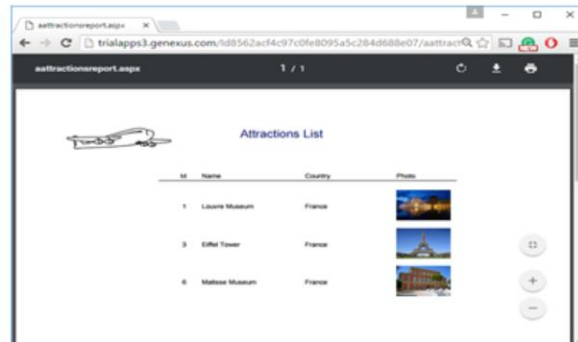
En cambio, se ordenamos por el atributo por el que filtramos, vemos en la navegación que ya no se recorre toda la tabla.

DEMO



```
1 Event 'List Attractions By Country'
2 //AttractionsList(&CountryId)
3 AttractionsReport(&CountryId)
4 Endevent
5
6
7 Event 'List Attractions By Name'
8 AttractionsByName( &AttractionNameFrom, &AttractionNameTo )
9 Endevent
10
```

- F5 → ejecutar Web panel, elegir France



Cambiamos la invocación en el web panel que llamaba a AttractionsList, por una invocación a este otro procedimiento: Con doble barra lo que sigue queda como comentario, es decir, no es interpretado por GeneXus como una instrucción. Así, comentamos la primera invocación, y escribimos la nueva:

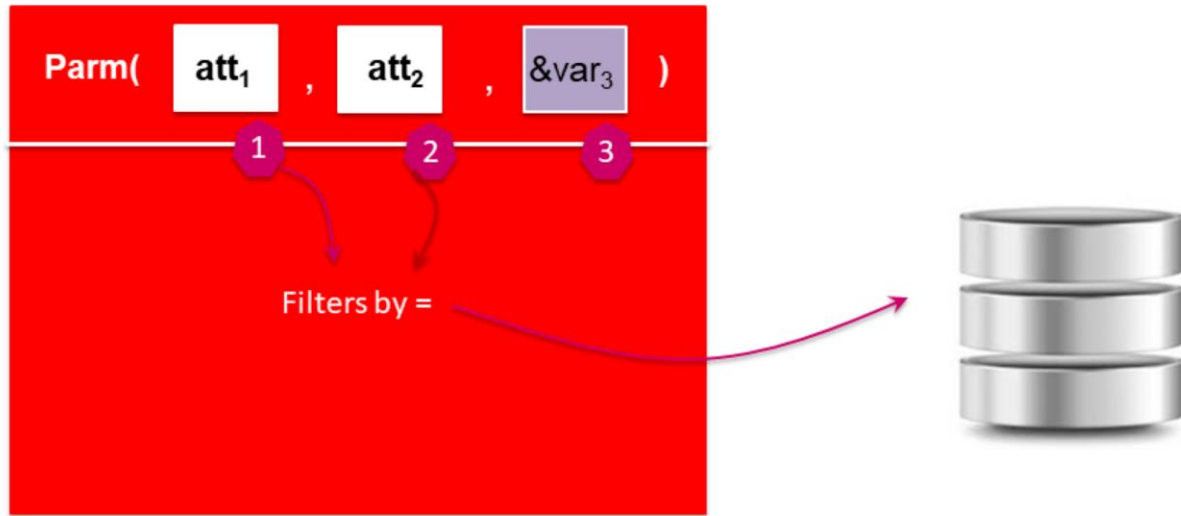
Ejecutemos.

Elegimos Francia y listamos...

Y vemos que salen, efectivamente, las atracciones turísticas de Francia.

Por último, subamos todo a GeneXus Server.

Object B



Si recibiéramos más de un valor utilizando atributos para recibirlos, se accederán únicamente los registros que tengan el mismo valor de cada atributo recibido.

Y a estos atributos no podremos cambiarles el valor.


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

Object B

Parm(&var₁ , &var₂ , &var₃)

1

2

3

Filters by... > >= < <= Like, etc.

&var₁ = ...

Si nuestro objetivo no es recibir valores para filtrar por igualdad, entonces en lugar de utilizar atributos la solución es recibir los valores en variables...

...a las que además podremos utilizar libremente en la programación, por ejemplo para asignarle otros valores si es necesario.

En este sentido, también debemos decir que no sólo la última variable de la regla parm puede ser utilizada como parámetro de salida, es decir, a ser devuelto a quien lo invocó. Todas las variables pueden ser de entrada, de salida, o de entrada/salida. En el ejemplo, la &variable 1, que como indicamos modifica su valor dentro del código del objeto, podría ser una variable de salida, o de entrada/salida.

Esto puede indicarse anteponiendo a la variable las palabras "in", "inout" o "out" como vemos aquí.

La comunicación entre objetos es vital para cualquier aplicación GeneXus, ya que es el mecanismo para que un objeto inicie la ejecución de otro y pueda enviar o recibir información del mismo.



Videos

training.genexus.com

Documentation

wiki.genexus.com

Certifications

training.genexus.com/certifications