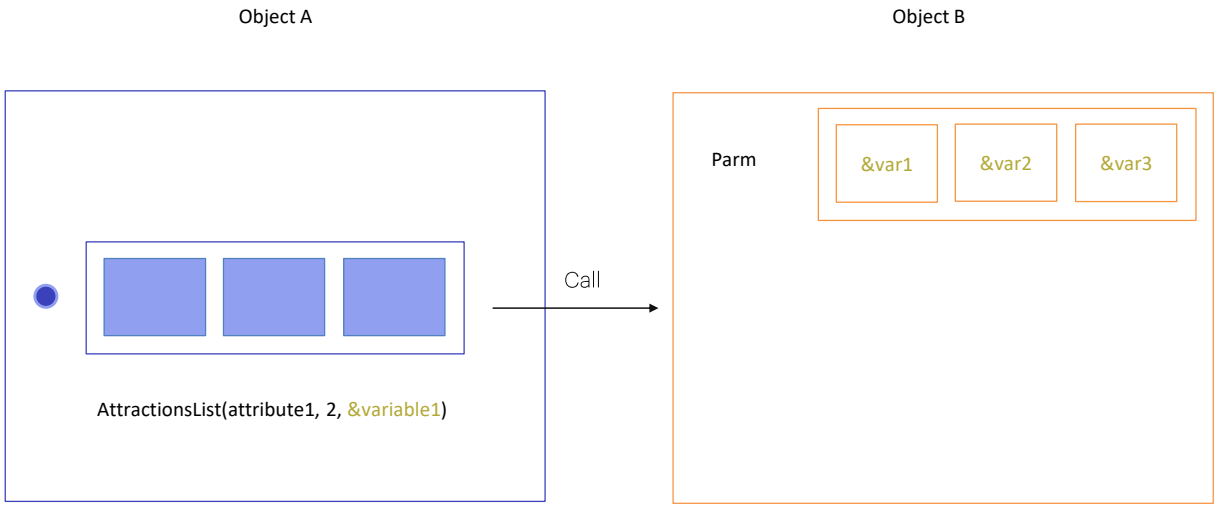
Invocaciones entre objetos (Cont.)

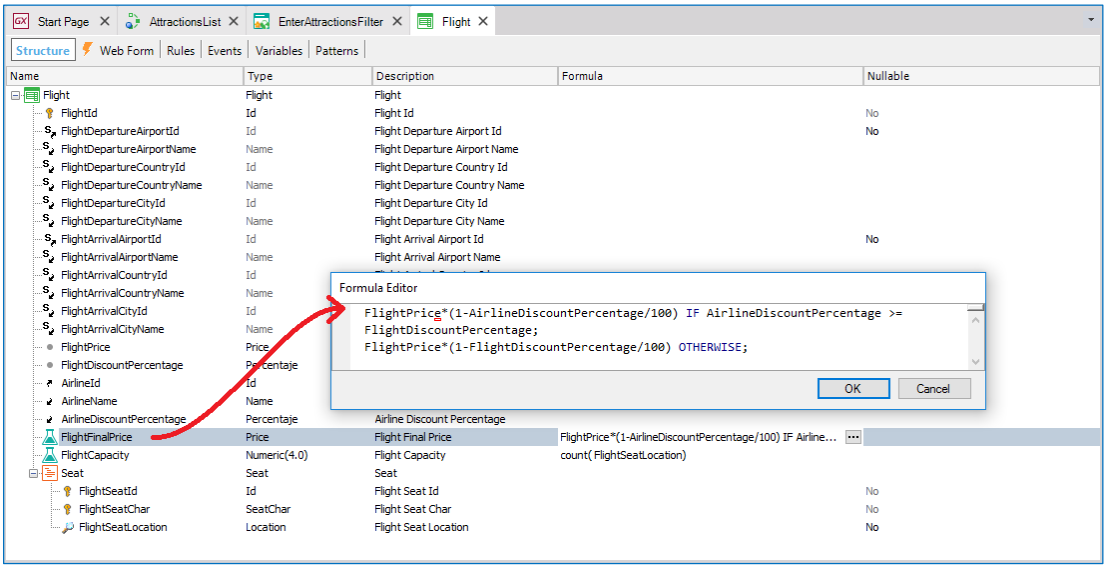
GeneXus™



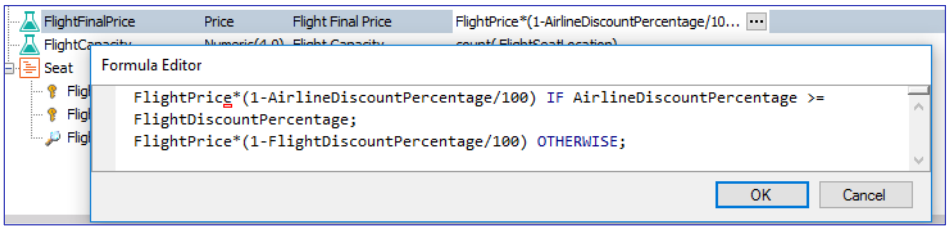
Hasta aquí vimos cómo declarar en un objeto parámetros para permitirle recibir datos de otro objeto y tomar las acciones pertinentes de acuerdo a esos datos. Para ello utilizábamos la regla Parm y variables. Los ejemplos que vimos eran de parámetros de entrada, es decir, parámetros que el objeto únicamente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

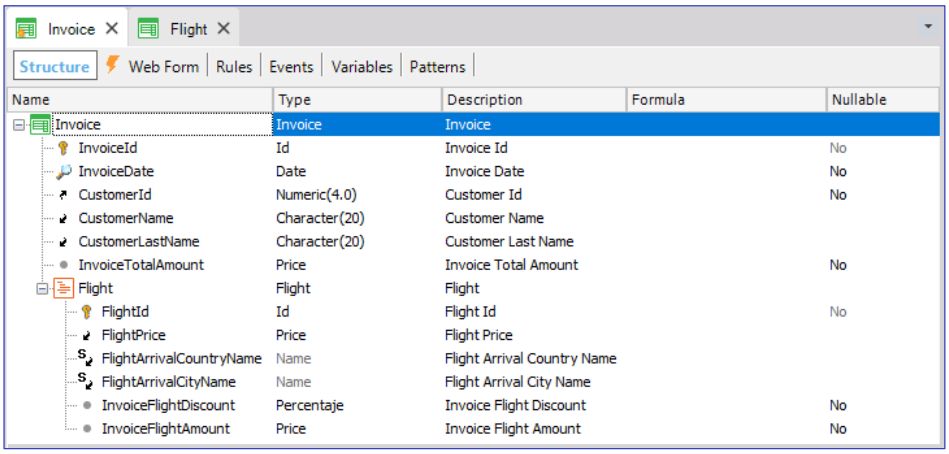
Así, si el objeto B tiene declarada una regla Parm con tres variables, todo objeto que quiera invocar al B deberá enviarle tres valores, que, como vimos, podían estar guardados en atributos, ser una expresión (como el caso de un valor fijo), o estar guardados en variables.

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



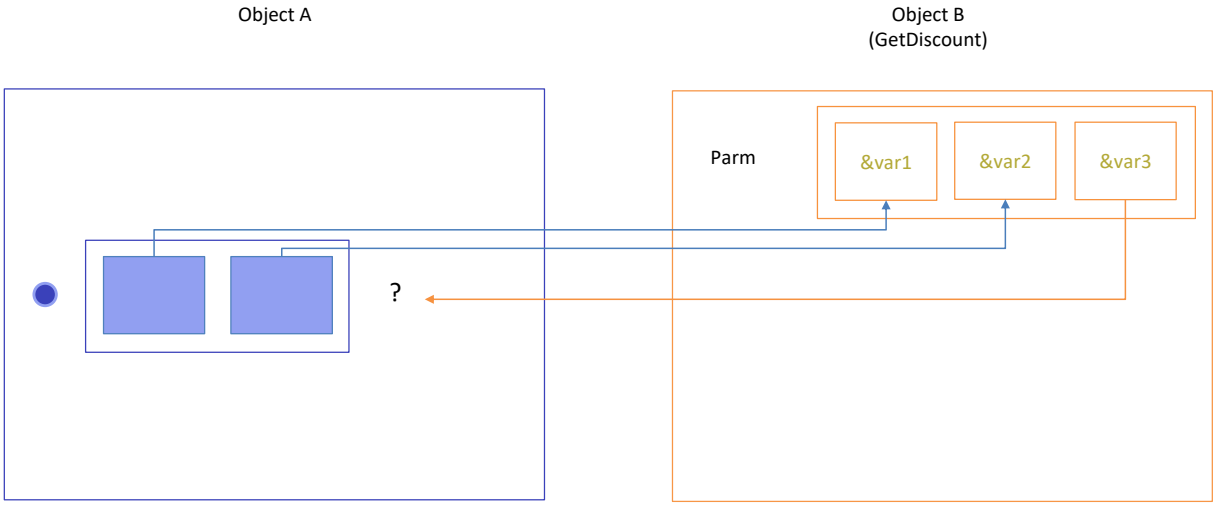
Teníamos en la transacción Flight una fórmula que calculaba el precio de un vuelo de acuerdo al porcentaje de descuento que brindaba la aerolínea y el porcentaje que se especificaba para el propio vuelo. Elegía el mejor descuento y ese era el que aplicaba.





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

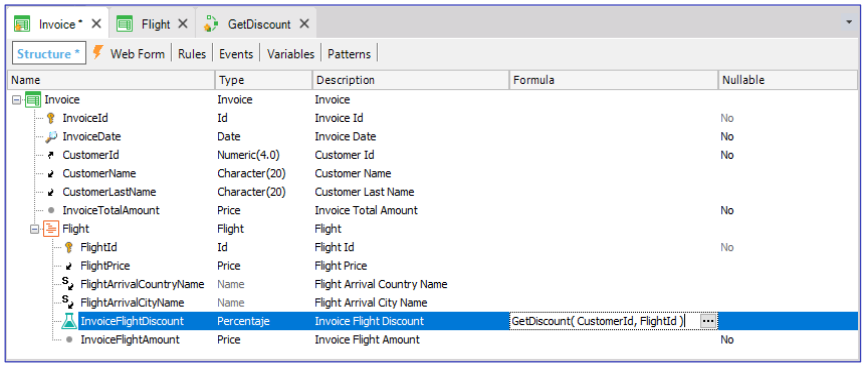
Y supongamos que el descuento es un cálculo más complejo, que implica no sólo al vuelo, sino por ejemplo a alguna condición relativa al cliente que está comprando el vuelo. Por ejemplo, la cantidad de vuelos que ha comprado antes, qué tan buen cliente es, etcétera. Y, por ejemplo, si un destino está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

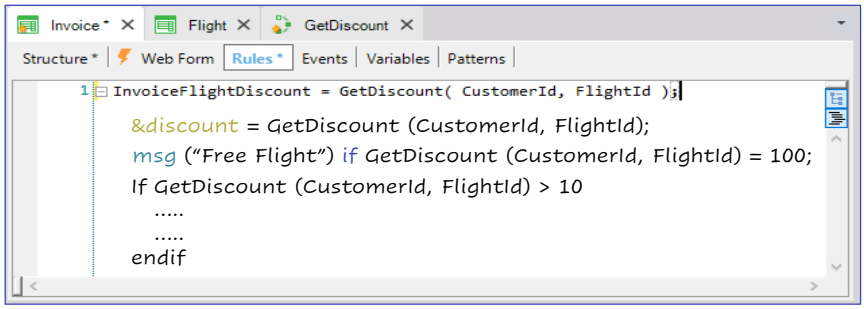


Para casos como este, podemos necesitar implementar un procedimiento que realice estos cálculos, y devuelva el valor resultante a quien lo llama.

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.





Una posibilidad es asignar el resultado a un atributo. Por ejemplo, podríamos definir al atributo InvoiceFlightDiscount 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.

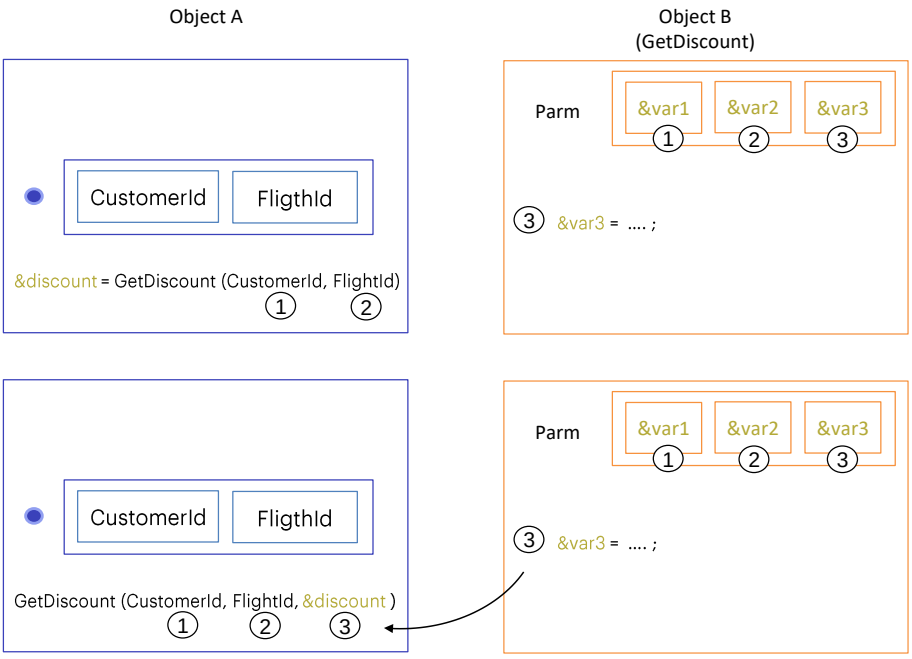
Pero también se podría asignar el resultado de la ejecución del procedimiento a una variable.

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

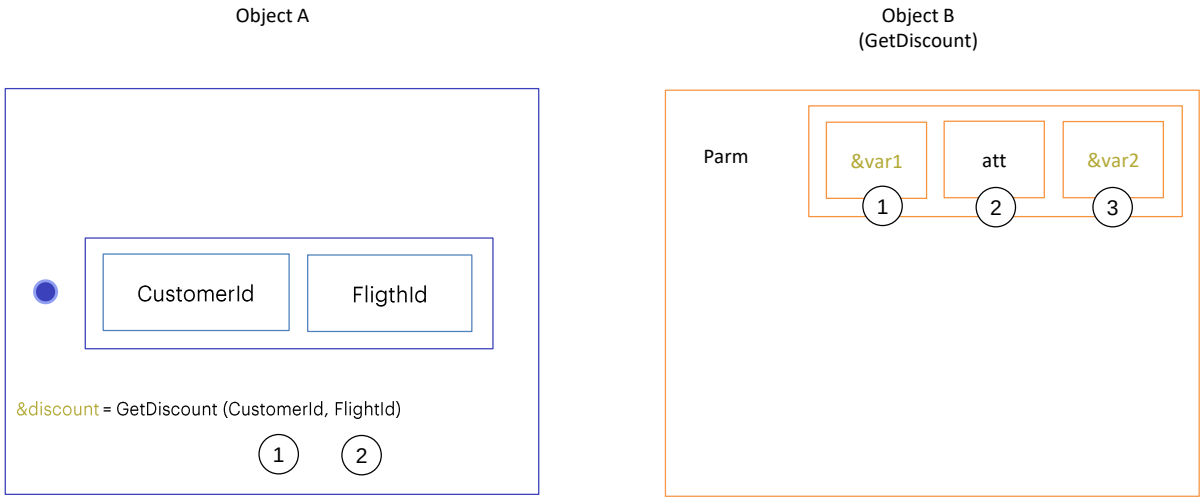
O de unas instrucciones en un procedimiento o en un evento:

```
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.



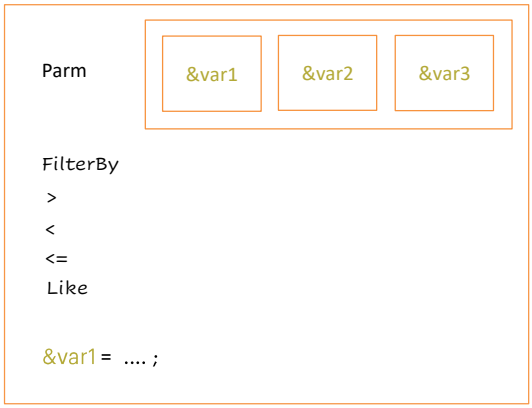
En la sección de reglas del procedimiento GetDiscount deberemos declarar la regla parm con la cantidad de parámetros que se escriben en la invocación... más uno al final que deberá ser una variable cuyo valor se cargue en el código del objeto (en nuestro caso, en el Source del procedimiento). El valor con el que quede la variable al final de la ejecución del código será el valor devuelto al objeto que lo llamó.



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

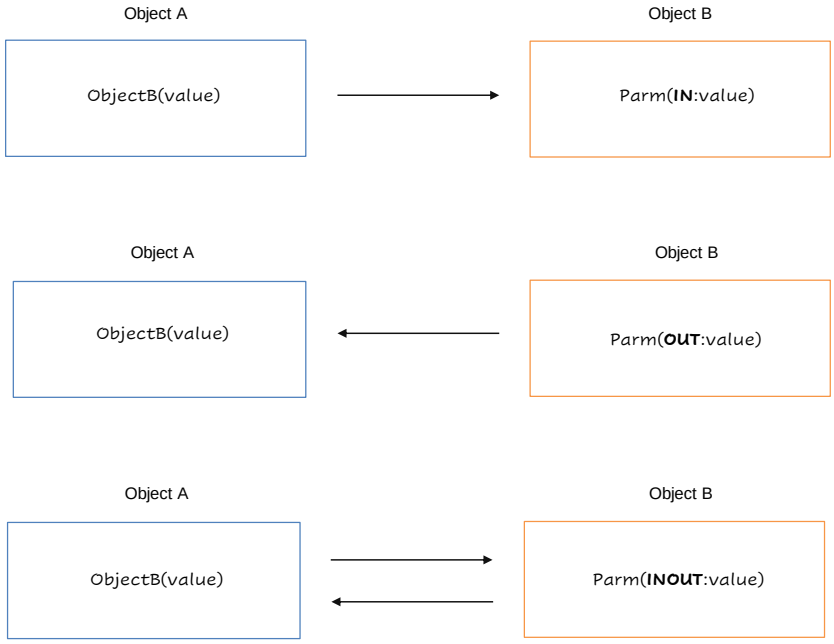
¿Cuál es la diferencia entre usar una variable o un atributo en la regla parm del objeto invocado?

Object B




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

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.



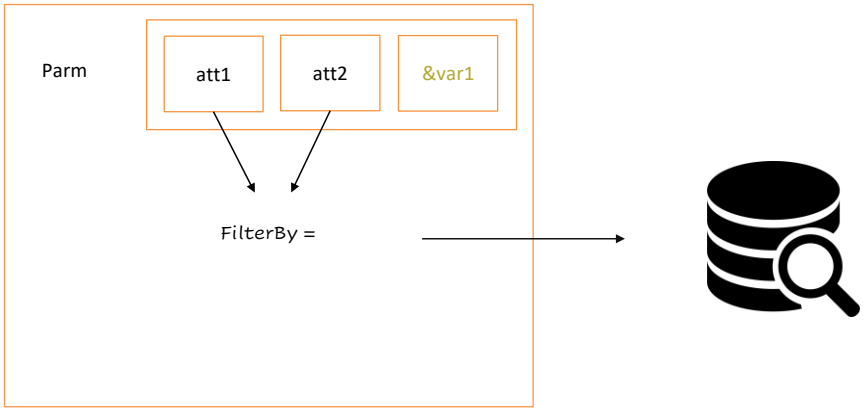
Para cada parámetro declarado en la regla Parm, podemos definir si el parámetro se usa para recibir un valor, para devolver un valor o para ambas cosas. Esto lo hacemos mediante los operadores in, out e inout respectivamente.

El operador IN, indica que el parámetro es de entrada, o sea el parámetro llega con un valor y ese valor no puede cambiarse.

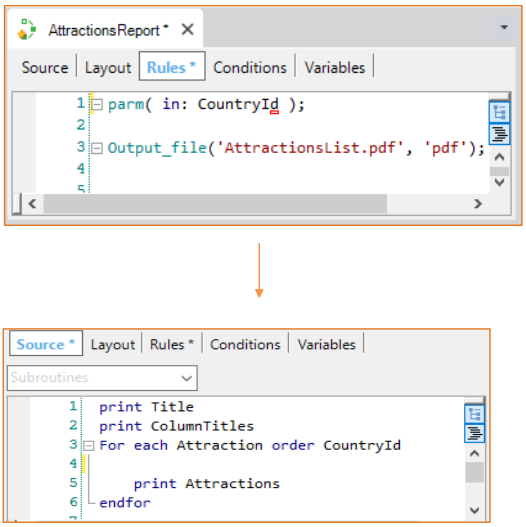
Los parámetros con operador OUT, son los parámetros de salida. Éstos no llegan con ningún valor y luego que se ejecuta el objeto llamado, el parámetro de salida contendrá el valor resultante que será devuelto al objeto llamador.

Por último, tenemos el operador INOUT, que hace que el parámetro sea de entrada y salida al mismo tiempo. Con este operador, el parámetro llega con un valor y puede ser cambiado durante la ejecución del objeto. Cuando finaliza, el parámetro contendrá el valor que se devuelve al objeto que lo llamó.

Si no declaramos ninguno de estos operadores en la regla parm, GeneXus asignará el operador INOUT a todos los parámetros, aunque esto no nos aparezca en pantalla.



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.

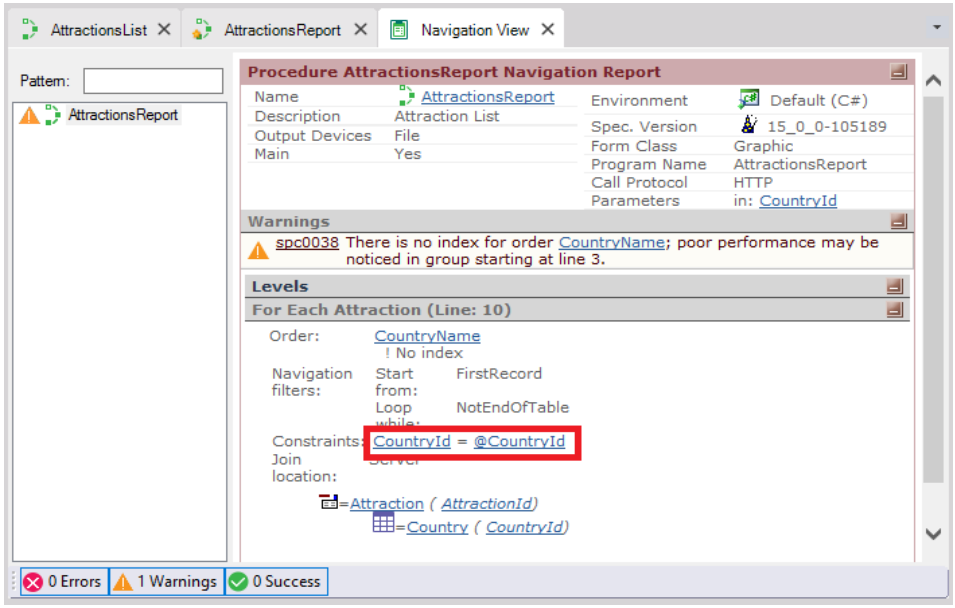


Hagamos una copia del procedimiento AttractionsList, con el nombre AttractionsReport.
Recordemos que el procedimiento en el que se basa utilizaba una variable como parámetro: &CountryId. Y la utilizaba para filtrar las atracciones de la tabla Attraction por país.

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.

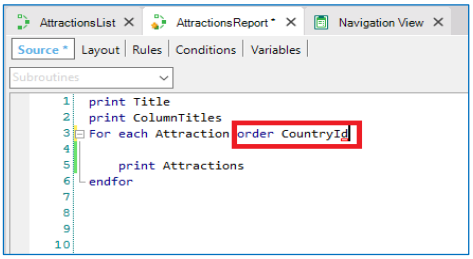
Cuando recibimos el valor en un atributo en la regla Parm, GeneXus filtra por igualdad, es decir, solamente se va a acceder a los registros que tengan ese valor de identificador de país.

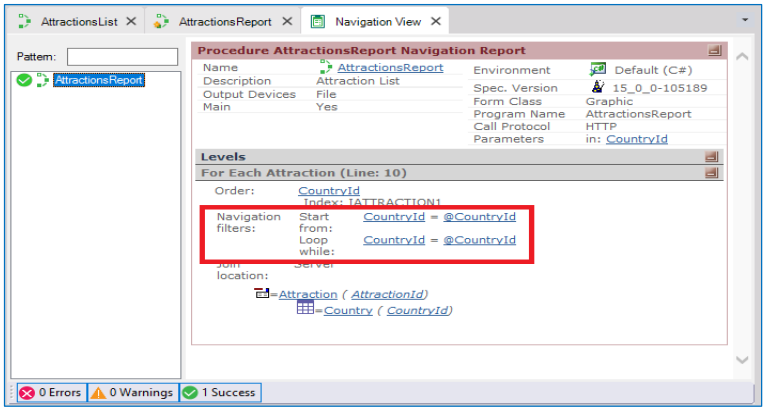
Si pedimos ver la navegación de este objeto...



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.





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

Ejecutemos.

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.

Si nuestro objetivo no es recibir valores para filtrar por igualdad, entonces en lugar de utilizar atributosla solución es recibir los valores en variables, a las que además, podremos utilizarlas libremente en la programación, por ejemplo para asignarle otros valores si es necesario.
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.

Existen otras maneras de comunicación entre objetos en las que no hay pasaje de datos por parámetros. Un ejemplo de esto, es cuando persistimos información en memoria mediante las variables del tipo WebSession, o cuando hacemos uso de los eventos globales. Estos casos no los veremos en este video, pero es importante saber que existen varias y diversas ~~las~~ formas de comunicación entre objetos.