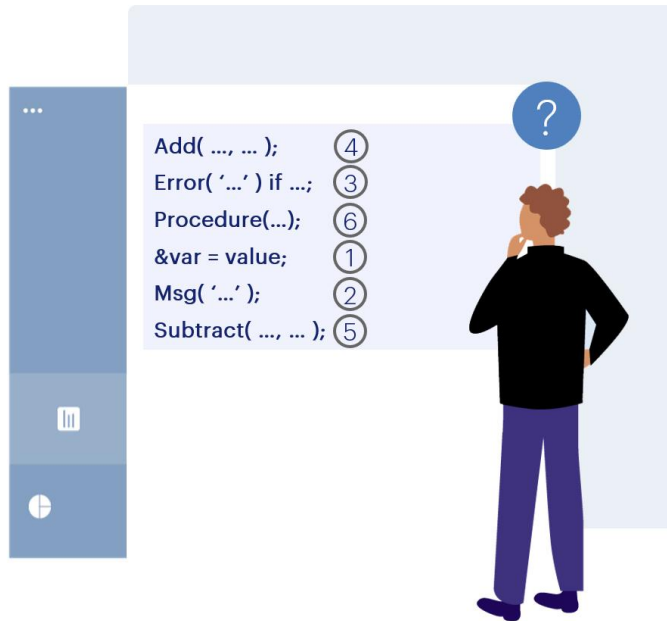


Evaluation Tree of Rule and Formula Triggering

GeneXus

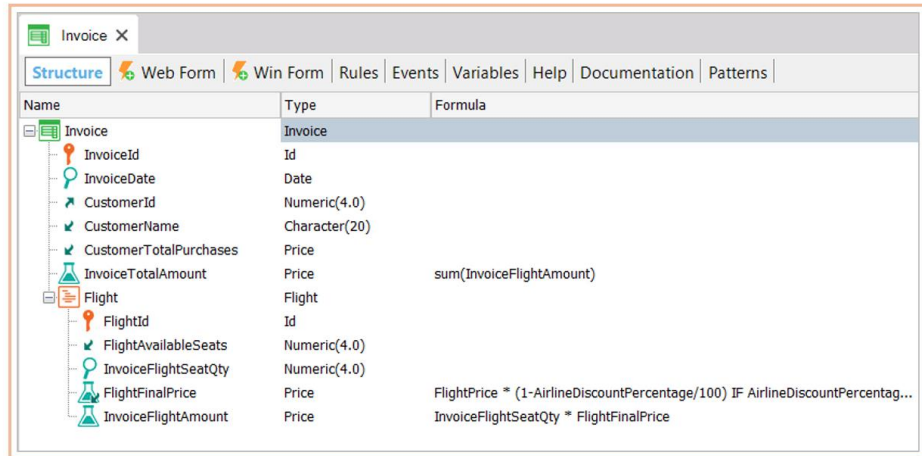
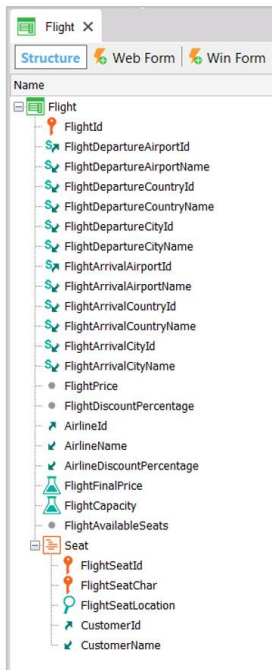
Rules in Transactions



We know that the rules in a transaction are stated in any order and GeneXus determines when each one is triggered. This is sometimes confusing for developers, because they feel they are no longer in control.

But this is actually an advantage. Developers should only focus on declaring the logic and GeneXus will automatically infer where and when each rule should be triggered.

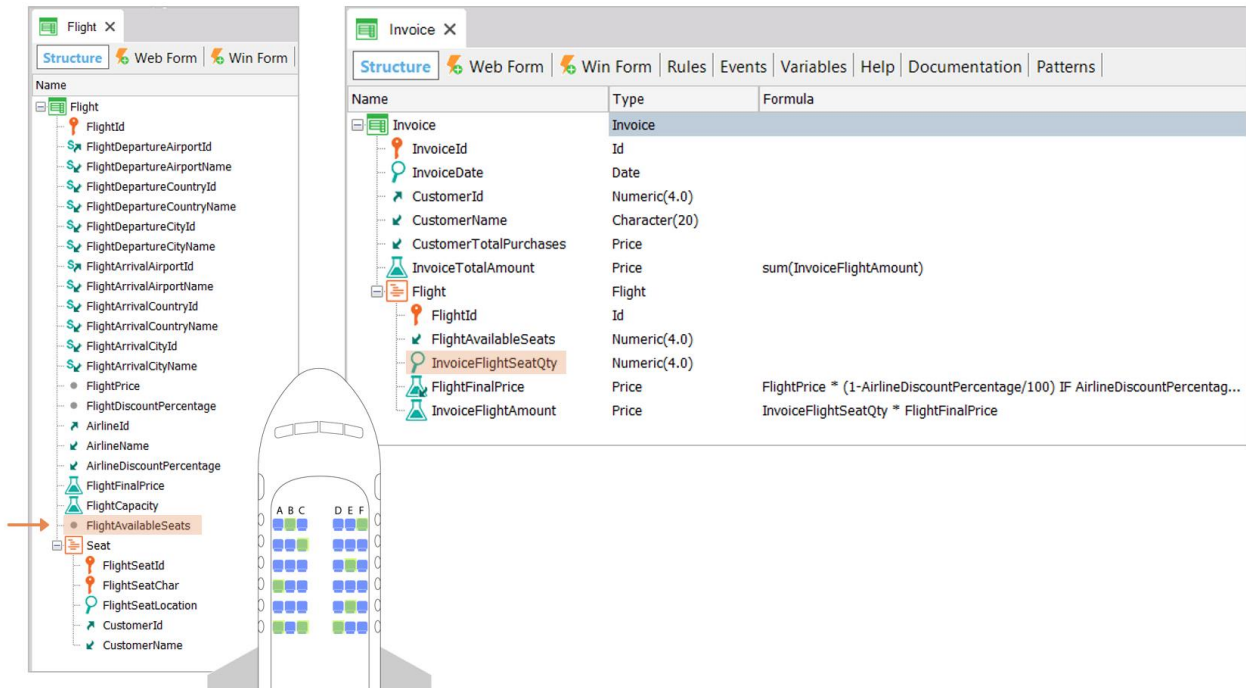
Reality



To explain this topic, we will use the Invoice transaction that has a second level (Flight), to represent the flights included in the invoice.

We are going to focus on this transaction and on triggering its rules.

Reality



Note that in the Flight transaction we have added the **FlightAvailableSeats** attribute, which will be used to record the seats available in each flight. The number of seats available will decrease every time that an invoice is issued to a customer who has purchased a number of seats in the flight.

We have added it, then, at this level. It will be an inferred attribute.

Reality

The image displays the GeneXus development environment with three main components:

- Flight Structure:** A tree view showing attributes such as FlightId, FlightDepartureAirportId, FlightDepartureAirportName, FlightDepartureCountryId, FlightDepartureCountryName, FlightDepartureCityId, FlightDepartureCityName, FlightArrivalAirportId, FlightArrivalAirportName, FlightArrivalCountryId, FlightArrivalCountryName, FlightArrivalCityId, FlightArrivalCityName, FlightPrice, FlightDiscountPercentage, AirlineId, AirlineName, AirlineDiscountPercentage, FlightFinalPrice, FlightCapacity, FlightAvailableSeats, and Seat (with sub-attributes FlightSeatId, FlightSeatChar, FlightSeatLocation, CustomerId, and CustomerName).
- Invoice Structure:** A table listing attributes and their types/formulas:

Name	Type	Formula
Invoice	Invoice	
InvoiceId	Id	
InvoiceDate	Date	
CustomerId	Numeric(4,0)	
CustomerName	Character(20)	
CustomerTotalPurchases	Price	
InvoiceTotalAmount	Price	$\text{sum}(\text{InvoiceFlightAmount})$
Flight	Flight	
FlightId	Id	
FlightAvailableSeats	Numeric(4,0)	
InvoiceFlightSeatQty	Numeric(4,0)	
FlightFinalPrice	Price	$\text{FlightPrice} * (1 - \text{AirlineDiscountPercentage} / 100)$ IF AirlineDiscountPercentag...
InvoiceFlightAmount	Price	$\text{InvoiceFlightSeatQty} * \text{FlightFinalPrice}$
- Customer Structure:** A table listing attributes and their types:

Name	Type
Customer	Customer
CustomerId	Id
CustomerName	Name
CustomerLastName	Name
CustomerAddress	Address, GeneXus
CustomerPhone	Phone, GeneXus
CustomerEmail	Email, GeneXus
CustomerAddedDate	Date
CustomerTotalPurchases	Price

A diagram of an airplane cabin layout is shown in the center, with rows labeled A, B, C, D, E, F and seats represented by blue squares.

In the Customer transaction we've also added the **CustomerTotalPurchases** attribute to record the total amount spent by the customer in flight ticket purchases. We also added it to our transaction as an inferred attribute.

The attributes InvoiceTotalAmount, FlightFinalPrice, and InvoiceFlightAmount of the Invoice structure have been defined as formulas.

Invoice Rules

Name	Type	Formula
Invoice	Invoice	
InvoiceId	Id	
InvoiceDate	Date	
CustomerId	Numeric(4,0)	
CustomerName	Character(20)	
CustomerTotalPurchases	Price	
InvoiceTotalAmount	Price	sum(InvoiceFlightAmount)
Flight	Flight	
FlightId	Id	
FlightAvailableSeats	Numeric(4,0)	
InvoiceFlightSeatQty	Numeric(4,0)	
FlightFinalPrice	Price	FlightPrice * (1-AirlineDiscountPercentage/100) IF AirlineDiscountPercentag...
InvoiceFlightAmount	Price	InvoiceFlightSeatQty * FlightFinalPrice

```

1 Default(InvoiceDate, &Today);
2
3 Subtract(InvoiceFlightSeatQty, FlightAvailableSeats);
4
5 Error("There are no more seats for sale")
6   if FlightAvailableSeats < 0;
7
8 Add(InvoiceTotalAmount, CustomerTotalPurchases);
9
10

```

In the Invoice transaction, we have defined the following rules to specify its behavior:

The **Default** rule, which initializes the invoice date attribute with today's date; the **Subtract** rule, which decreases the number of available seats on the flight according to the number of seats purchased on the invoice – note that it will be decreasing a Flight table attribute: Here, FlightAvailableSeats is inferred; the **Error** rule displays an error message if the flight no longer has available seats; and the **Add** rule adds the invoice total to the total purchases made by the customer –again, an attribute included in a table of the extended table, Customer.

Evaluation Tree

(R) Default(InvoiceDate, &Today);
 (R) Add(InvoiceTotalAmount, CustomerTotalPurchases);
 (F) InvoiceTotalAmount = Sum(InvoiceFlightAmount)
 (F) InvoiceFlightAmount = FlightFinalPrice * InvoiceFlightSeatQty
 (F) FlightFinalPrice = FlightPrice * (1 - AirlineDiscountPercentage...)
 (R) Subtract(InvoiceSeatQty, FlightAvailableSeats);
 (R) Error("There are no more seats for sale") if FlightAvailableSeats < 0;

Invoice

<ErrorViewer: ErrorViewer>

<Toolbar>

Id

Date

Customer Id

Customer Name

Customer Total Purchases

Total Amount

Flight

Flight Id	Flight Available Seats	Seat Qty	Flight Final Price	Flight Amount
FlightId	FlightAvailableSeats	InvoiceFlightSeatQty	FlightFinalPrice	InvoiceFlightAmount

<FormButtons>

In short, we have all these rules and formulas defined in the Invoice transaction:

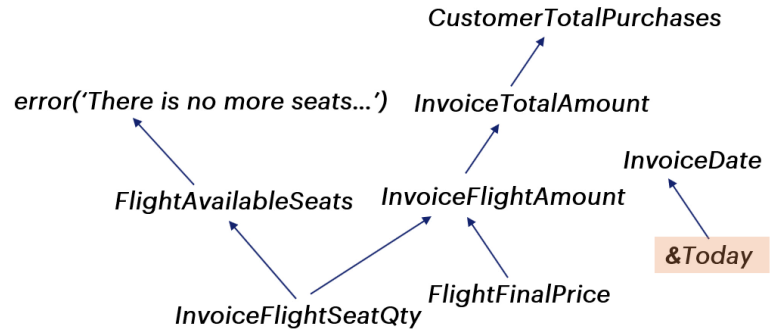
The big question is how does GeneXus know in what order to trigger them, and when to trigger them and when not to?

Of course, first there is a natural order that corresponds to the ordering of the attributes on the screen (from top to bottom and from left to right).

Evaluation Tree

(R) Default(InvoiceDate, &Today);
 (R) Add(InvoiceTotalAmount, CustomerTotalPurchases);
 (F) InvoiceTotalAmount = Sum(InvoiceFlightAmount)
 (F) InvoiceFlightAmount = FlightFinalPrice * InvoiceFlightSeatQty
 (F) FlightFinalPrice = FlightPrice * (1 - AirlineDiscountPercentage...)
 (R) Subtract(InvoiceSeatQty, FlightAvailableSeats);
 (R) Error("There are no more seats for sale") if FlightAvailableSeats < 0;

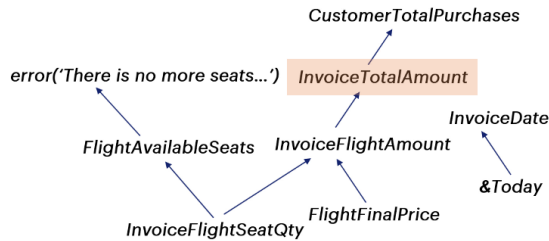
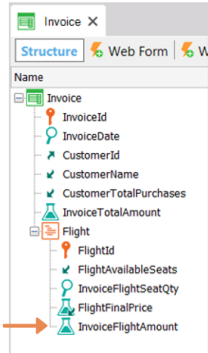
Invoice	
	<< < > >> SELECT
Id	<input type="text" value=""/>
Date	<input type="text" value="10/13/20"/> <input type="button" value="28"/>
Customer Id	<input type="text" value="0"/> <input type="button" value="↑"/>
Customer Name	



A rule or formula is triggered as soon as the required information is accessed. For example, the Default rule only needs the value of the &Today variable and to know that it is in Insert mode. That's why as soon as we open the screen in Insert mode we already see the value in the field, even though we haven't even reached it (we are barely positioned on Invoiceld).

Evaluation Tree

- (R) Default(InvoiceDate, &Today);
- (R) Add(InvoiceTotalAmount, CustomerTotalPurchases);
- (F) InvoiceTotalAmount = Sum(InvoiceFlightAmount)
- (F) InvoiceFlightAmount = FlightFinalPrice * InvoiceFlightSeatQty
- (F) FlightFinalPrice = FlightPrice * (1 - AirlineDiscountPercentage...)
- (R) Subtract(InvoiceSeatQty, FlightAvailableSeats);
- (R) Error("There are no more seats for sale") if FlightAvailableSeats < 0;



Invoice

Id:

Date: 10/13/20

Customer Id:

Customer Name:

Customer Total Purchases: 0.00

Total Amount: → 0.00

Flight

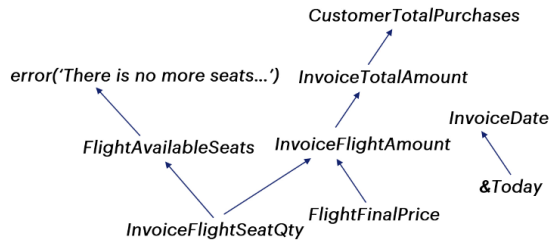
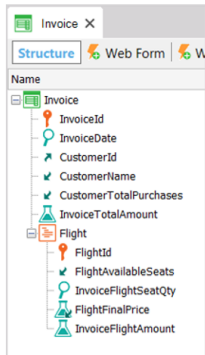
Flight Id	Flight Available Seats	Seat Qty	Flight Final Price	Flight Amount
<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	0.00	0.00
<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	0.00	0.00
<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	0.00	0.00
<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	0.00	0.00
<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	0.00	0.00

[New row]

Let's think about what happens with the formula of the first level: InvoiceTotalAmount, which is a sum of a second-level attribute. Since the Sum formula needs only the InvoiceFlightAmount attribute, it will be triggered for the header even before the first line could be entered, giving 0.

Evaluation Tree

- (R) Default(InvoiceDate, &Today);
- (R) Add(InvoiceTotalAmount, CustomerTotalPurchases);
- (F) InvoiceTotalAmount = Sum(InvoiceFlightAmount)
- (F) InvoiceFlightAmount = FlightFinalPrice * InvoiceFlightSeatQty
- (F) FlightFinalPrice = FlightPrice * (1 - AirlineDiscountPercentage...)
- (R) Subtract(InvoiceSeatQty, FlightAvailableSeats);
- (R) Error("There are no more seats for sale") if FlightAvailableSeats < 0;



Invoice

<< < > >> SELECT

Id

Date 29

Customer Id ↓

Customer Name Joseph ⊞

Customer Total Purchases 25110.00

Total Amount 9900.00

Flight

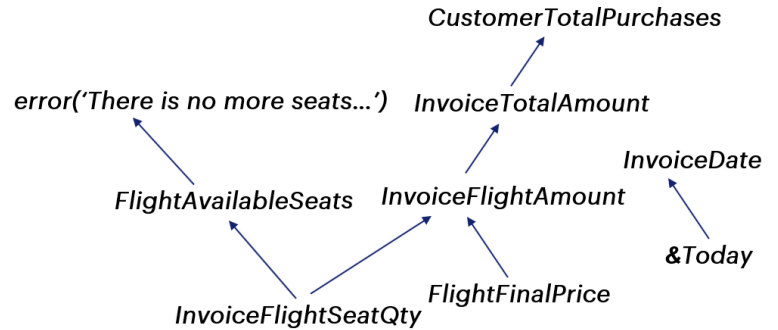
	Flight Id	Flight Available Seats	Seat Qty	Flight Final Price	Flight Amount
×	<input type="text" value="1"/> ↓	148	<input type="text" value="2"/> ↓	2700.00	5400.00
×	<input type="text" value="2"/> ↓	497	<input type="text" value="1"/> ↓	4500.00	4500.00
	<input type="text" value="0"/> ↓	0	<input type="text" value="0"/> ↓	0.00	0.00
	<input type="text" value="0"/> ↓	0	<input type="text" value="0"/> ↓	0.00	0.00
	<input type="text" value="0"/> ↓	0	<input type="text" value="0"/> ↓	0.00	0.00

[New row]

But then, as we enter lines, it will be triggered again for each one. But what happens if we access the transaction in Update mode and, for example, change something in a line that does not modify the InvoiceFlightAmount at all? (in this case, we don't have any attributes to be modified that don't modify that value, because the only two editable attributes are the line ID, which cannot be changed because it is part of the primary key, and then the InvoiceFlightSeatQty, which does modify the value of InvoiceFlightAmount. However, imagine that there were one; for example, that it must be indicated if a passenger has diabetes, and the line in question said Yes, and now we want to change it to No). Obviously, in that case, the header formula would not be recalculated.

Evaluation Tree

```
(R) Default( InvoiceDate, &Today );
(R) Add( InvoiceTotalAmount, CustomerTotalPurchases );
(F) InvoiceTotalAmount = Sum( InvoiceFlightAmount )
(F) InvoiceFlightAmount = FlightFinalPrice * InvoiceFlightSeatQty
(F) FlightFinalPrice = FlightPrice * (1 - AirlineDiscountPercentage...)
(R) Subtract( InvoiceSeatQty, FlightAvailableSeats );
(R) Error( "There are no more seats for sale" ) if FlightAvailableSeats < 0;
```



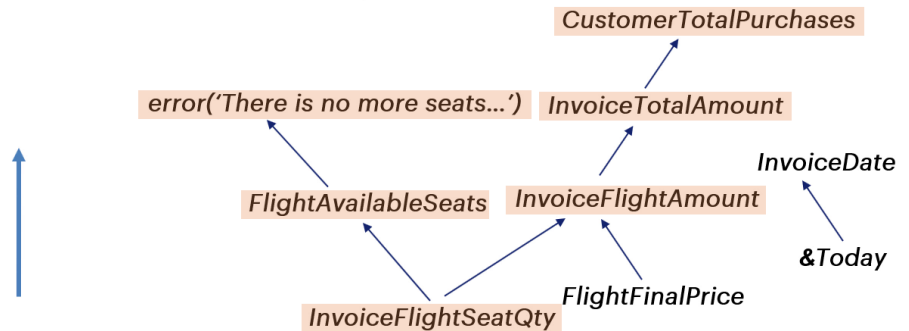
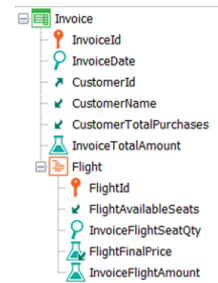
This is also obvious to GeneXus, which internally extracts the existing dependencies between places assumed by the controls on the screen, the rules and formulas, to build a dependency tree (known as evaluation tree) that will determine which rules and formulas will have to be triggered again when changes are made to attributes on the screen. In this example it will be as shown below:

Evaluation Tree

```

(R) Default( InvoiceDate, &Today );
(R) Add( InvoiceTotalAmount, CustomerTotalPurchases );
(F) InvoiceTotalAmount = Sum( InvoiceFlightAmount )
(F) InvoiceFlightAmount = FlightFinalPrice * InvoiceFlightSeatQty
(F) FlightFinalPrice = FlightPrice * (1 - AirlineDiscountPercentage...)
(R) Subtract( InvoiceSeatQty, FlightAvailableSeats );
(R) Error( "There are no more seats for sale" ) if FlightAvailableSeats < 0;

```



For example, note what happens with the attribute InvoiceFlightSeatQty. On it depends the update using subtract of the attribute FlightAvailableSeat, on which, in turn, the triggering of the error rule depends. For this reason, the condition of the error rule must be written knowing that, because of this dependency, the subtract will always have already been executed and that is why the condition 'less than zero' is placed. Remember that, as we have already studied, if there are negative entries, everything done in the tree that led to this error will be undone.

At the same time, the updating of the InvoiceFlightAmount formula also depends on InvoiceFlightSeatQty, as does that of the header, InvoiceTotalAmount, on which the updating of the client's total purchases depends.

We can imagine that the tree is executed in a bottom-up manner. That is to say, every time an attribute value is updated, all the rules and formulas that depend on this attribute (and that are located upwards in the tree) are executed.

Evaluation Tree

Invoice

Id: 0

Date: 09/17/20

Customer Id: 1

Customer Name: Joseph

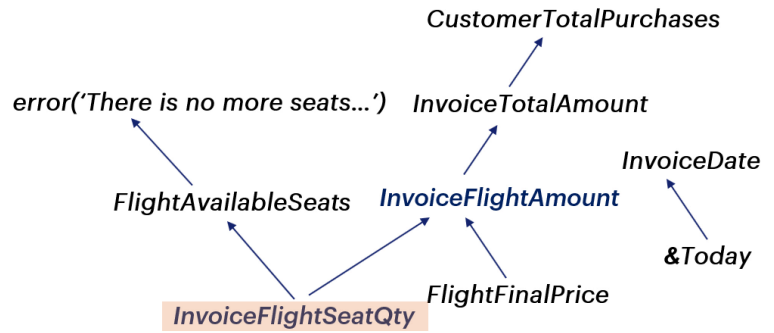
Customer Total Purchases: 5400.00

Total Amount: 5400.00

Flight

Flight Id	Flight Available Seats	Seat Qty	Flight Final Price	Flight Amount
2	98	2	2700.00	5400.00
0	0	0	0.00	0.00
0	0	0	0.00	0.00
0	0	0	0.00	0.00
0	0	0	0.00	0.00

[New row]



$$\text{InvoiceFlightAmount} = \text{FlightFinalPrice} * \text{InvoiceFlightSeatQty}$$

Let's continue with the previous example:

If the number of seats in an invoice line (InvoiceFlightSeatQty) is updated, since this attribute is part of the formula that calculates the cost of the flight (InvoiceFlightAmount), this formula will be triggered again.

Evaluation Tree

Invoice

Id: 0

Date: 09/17/20

Customer Id: 1

Customer Name: Joseph

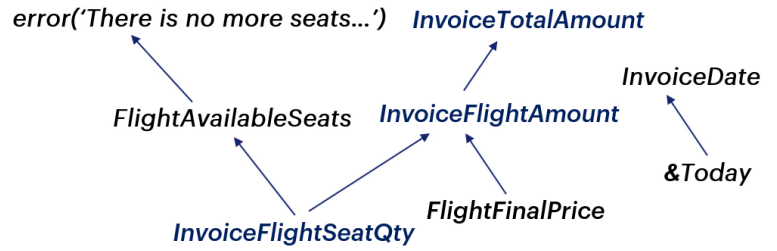
Customer Total Purchases: 5400.00

Total Amount: 5400.00

Flight

Flight Id	Flight Available Seats	Seat Qty	Flight Final Price	Flight Amount
2	98	2	2700.00	5400.00
	0	0	0.00	0.00
	0	0	0.00	0.00
	0	0	0.00	0.00
	0	0	0.00	0.00

[New row]



$$\text{InvoiceFlightAmount} = \text{FlightFinalPrice} * \text{InvoiceFlightSeatQty}$$

$$\text{InvoiceTotalAmount} = \text{Sum}(\text{InvoiceFlightAmount})$$

When it is triggered again, the formula corresponding to the total amount of the invoice (InvoiceTotalAmount) will also have to be triggered again.

Evaluation Tree

Invoice

« < > » SELECT

Id:

Date:

Customer Id:

Customer Name: Joseph

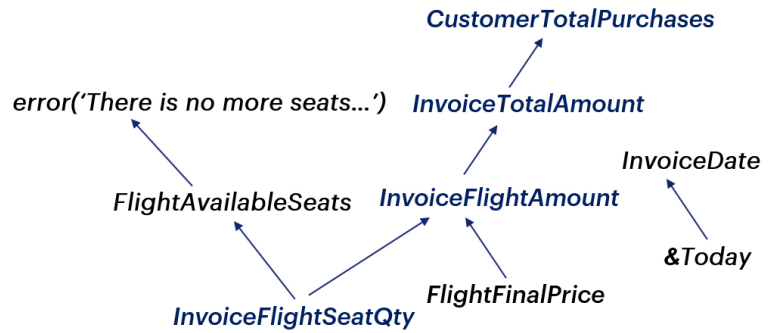
Customer Total Purchases: 5400.00 ←

Total Amount: 5400.00

Flight

Flight Id	Flight Available Seats	Seat Qty	Flight Final Price	Flight Amount
×	2 <input type="button" value="↑"/>	98	2	2700.00 5400.00
<input type="text" value=""/>	<input type="text" value="0"/> <input type="button" value="↑"/>	0	0	0.00 0.00
<input type="text" value="0"/>	<input type="text" value="0"/> <input type="button" value="↑"/>	0	0	0.00 0.00
<input type="text" value="0"/>	<input type="text" value="0"/> <input type="button" value="↑"/>	0	0	0.00 0.00
<input type="text" value="0"/>	<input type="text" value="0"/> <input type="button" value="↑"/>	0	0	0.00 0.00

[New row]



InvoiceFlightAmount = FlightFinalPrice * InvoiceFlightSeatQty

InvoiceTotalAmount = Sum(InvoiceFlightAmount)

Add(InvoiceTotalAmount, CustomerTotalPurchases);

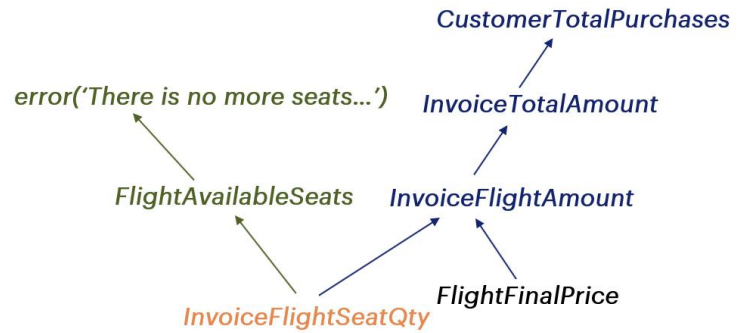
Lastly, changing the total also implies having to trigger the rule `Add(InvoiceTotalAmount, CustomerTotalPurchases)` because the customer's total purchases have to be updated.

Evaluation Tree

```

(R) Default( InvoiceDate, &Today );
(R) Add( InvoiceTotalAmount, CustomerTotalPurchases );
(F) InvoiceTotalAmount = Sum( InvoiceFlightAmount )
(F) InvoiceFlightAmount = FlightFinalPrice * InvoiceFlightSeatQty
(F) FlightFinalPrice = FlightPrice * (1 - AirlineDiscountPercentage...)
(R) Subtract( InvoiceSeatQty, FlightAvailableSeats );
(R) Error( "There are no more seats for sale" ) if FlightAvailableSeats < 0;

```



In addition to triggering all the formulas and rules included in the right branch of the tree from the attribute InvoiceFlightSeatQty, the formulas and rules included in the left branch will also be triggered.

Evaluation Tree

Invoice

« < > » SELECT

Id: 0

Date: 09/17/20

Customer Id: 1

Customer Name: Joseph

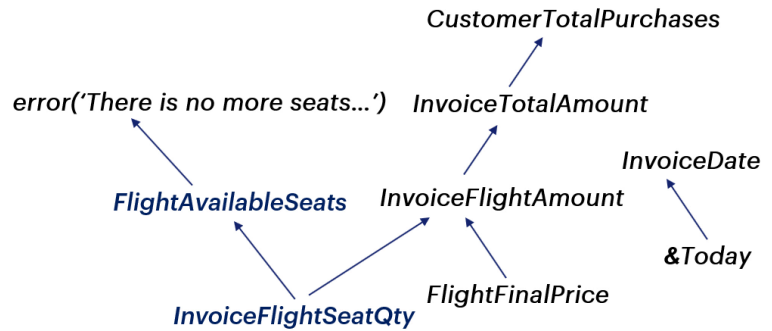
Customer Total Purchases: 5400.00

Total Amount: 5400.00

Flight

Flight Id	Flight Available Seats	Seat Qty	Flight Final Price	Flight Amount
2	98	2	2700.00	5400.00
	0	0	0.00	0.00
	0	0	0.00	0.00
	0	0	0.00	0.00
	0	0	0.00	0.00

[New row]



Subtract(InvoiceSeatQty, FlightAvailableSeats);

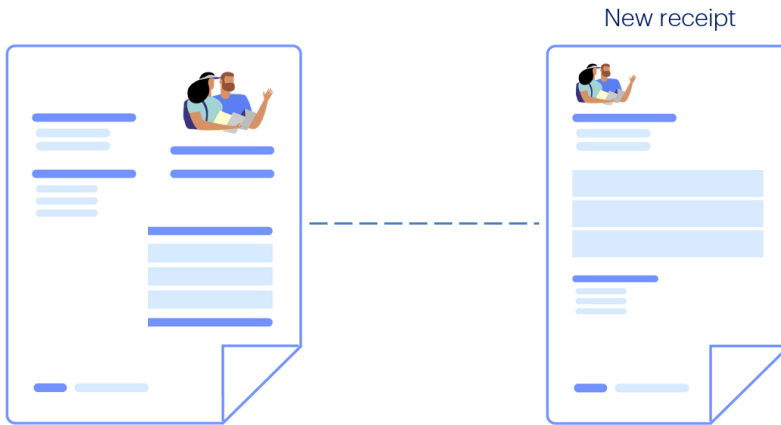
**Error("There are no more seats for sale")
if FlightAvailableSeats < 0;**

As we've seen, when the value of the InvoiceFlightSeatQty attribute is changed, the rule Subtract(InvoiceFlightSeatQty, FlightAvailableSeats) that updates the number of seats available on the flight (FlightAvailableSeats) will also be triggered again.

Consequently, by making changes to this rule, the value of the FlightAvailableSeats attribute will be evaluated to see if the Error rule indicating that there aren't any more available seats should be triggered.

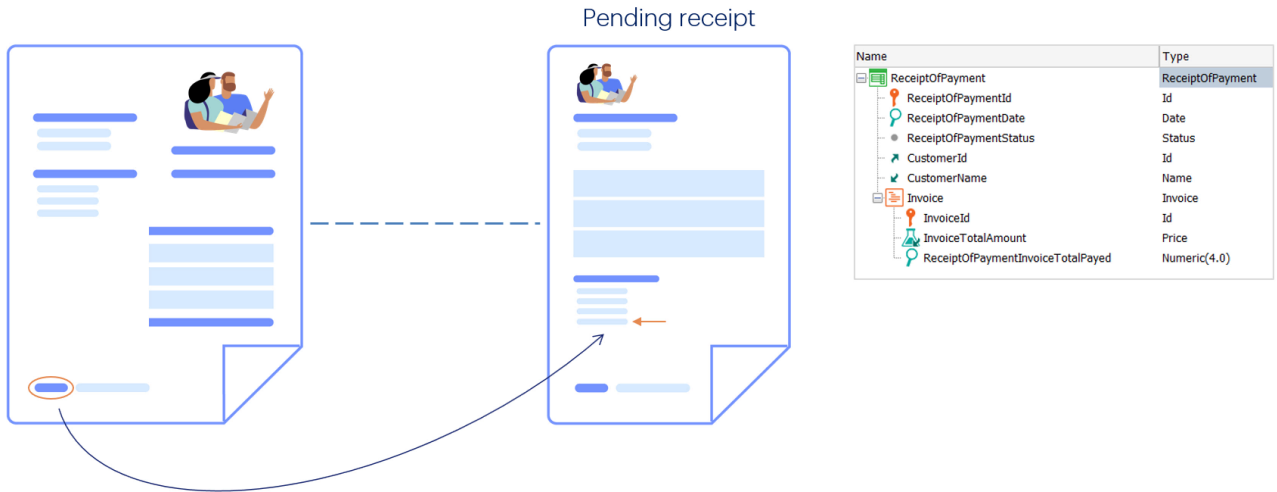
If the condition for triggering the error is met, everything done in the tree since the change in the InvoiceFlightSeatQty attribute will be automatically undone, and the database data will return to the state prior to the execution of the error rule.

Reality



Now let's see an example in which a rule is not always triggered at the desired time. Let's suppose that immediately after invoicing a customer for a number of flight seats purchased, we want to generate a receipt for payment. If the customer is up to date with payments, then we generate a new receipt;

Reality



Otherwise, we add the amount of this invoice to the previous outstanding receipt. We have created a ReceiptOfPayment transaction. It is made up of the receipt identifier, the date, the state (which is an enumerated domain with the values pending and completed), the customer, and a second level to record the invoices for which the payment receipt is issued.

Reality

Invoice

Id: 0

Date: 10/11/20

Customer Id: 1

Customer Name: Joseph

Customer Total Purchases: 14400.00

Total Amount: 0.00

Flight

Flight Id	Flight Available Seats	Seat Qty	Flight Final Price	Flight Amount
3	100	1	890.00	0.00
0	0	0	0.00	0.00
0	0	0	0.00	0.00
0	0	0	0.00	0.00
0	0	0	0.00	0.00
[New row]				

CONFIRM CANCEL

Diagram illustrating the data flow from the Invoice form to the RECEIPTOFPAYMENT Table. A dashed arrow points from the Date field (10/11/20) to a database icon, which then points to the RECEIPTOFPAYMENT Table. The table contains several rows of data, with a magnifying glass icon indicating a search operation. Below the table, the search criteria are defined:

```

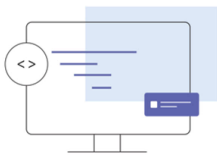
where CustomerId = &CustomerId
where ReceiptOfPaymentStatus = Status.Pending

```

Then, when a new invoice is entered, the program searches if there is already a receipt from that customer in Pending status.

Reality

id	Date	Customer Id	Invoice Total Amount
✓ 1	10/07/20	1	5400.00
✓ 2	10/09/20	1	9000.00
✓ 3	10/11/20	1	810.00



RECEIPTOFPAYMENT Table

...
...
...

if exists

where CustomerId = &CustomerId
 where ReceiptOfPaymentStatus = Status.Pending

Receipt Of Payment

<< < > >> SELECT

Payment Id:

Payment Date: 20

Payment Status: Pending

Customer Id:

Customer Name: Joseph

Invoice

	Invoice Id	Invoice Total Amount	Total Payed
×	1	5400.00	0
×	2	9000.00	0
×	3	810.00	0
	0	0.00	0
	0	0.00	0
	0	0.00	0
	0	0.00	0
	0	0.00	0

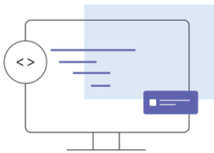
[New row]

If it there is one, a new line is added with this invoice.

Reality

id	Date	Customer Id	Invoice Total Amount
✓ 1	10/07/20	1	5400.00
✓ 2	10/09/20	1	9000.00
✓ 3	10/11/20	1	810.00

CANCEL



RECEIPTOFPAYMENT Table

...
...
...
...
...

if not exists

where CustomerId = &CustomerId
 where ReceiptOfPaymentStatus
 = Status.Pending

Receipt Of Payment

<< < > >> SELECT

Payment Id	<input type="text"/>
Payment Date	10/11/20 <input type="button" value="20"/>
Payment Status	<input type="radio"/> Pending <input type="radio"/> Paid
Customer Id	1 <input type="button" value="↑"/>
Customer Name	Joseph

Invoice

Invoice Id	Invoice Total Amount	Total Paid
✕ 3 <input type="button" value="↑"/>	\$10.00	0
0 <input type="button" value="↑"/>	0.00	0
0 <input type="button" value="↑"/>	0.00	0
0 <input type="button" value="↑"/>	0.00	0
0 <input type="button" value="↑"/>	0.00	0
0 <input type="button" value="↑"/>	0.00	0

[New row]

Otherwise, the header and line are created, and the header is left in Pending status.

Reality

Payment Id: 1

Payment Date: 10/11/20

Payment Status: Pending

Customer Id: 1

Customer Name: Joseph

Status Completed
if InvoiceTotalAmount =
ReceiptOfPaymentInvoiceTotalPaid

Invoice		
Invoice Id	Invoice Total Amount	Total Paid
1	5400.00	5400
2	9000.00	0
3	810.00	0
0	0.00	0
0	0.00	0
0	0.00	0
0	0.00	0
0	0.00	0
[New row]		

Name	Type
ReceiptOfPayment	ReceiptOfPayment
ReceiptOfPaymentId	Id
ReceiptOfPaymentDate	Date
ReceiptOfPaymentStatus	Status
CustomerId	Id
CustomerName	Name
Invoice	Invoice
InvoiceId	Id
InvoiceTotalAmount	Price
ReceiptOfPaymentInvoiceTotalPaid	Numeric(4,0)

CONFIRM
CANCEL
DELETE

Then the customer comes to pay, so the employee opens the transaction and changes the ReceiptOfPaymentInvoiceTotalPaid attribute for the invoices the customer wants to pay.

The ReceiptOfPaymentStatus can only be changed to Completed if the values of InvoiceTotalAmount and ReceiptOfPaymentInvoiceTotalPaid match for all lines. Let's assume that this change is made by the user, who changes the value of ReceiptOfPaymentStatus, so it must be confirmed that changing to Completed is not allowed if there is an unpaid or incorrectly paid invoice.

Reality

Payment Id: 1

Payment Date: 10/11/20

Payment Status: Completed

Customer Id: 1

Customer Name: Joseph

Invoice

Invoice Id	Invoice Total Amount	Total Paid
1	5400.00	5400
2	9000.00	0
3	810.00	0
0	0.00	0
0	0.00	0
0	0.00	0
0	0.00	0
0	0.00	0

[New row]

CONFIRM CANCEL DELETE

```

1 Error('Incomplete payments')
2   if ReceiptOfPaymentStatus = Status.Completed
3     and InvoiceTotalAmount <> ReceiptOfPaymentInvoiceTotalPaid
4     and Update;

```

We could think of placing the following error rule, conditioning it to be triggered when we are updating a receipt, the status is Completed, and the value of the attribute indicating what should be paid and the one indicating what has been paid for a line do not match:

But... when will this rule be triggered?

Clearly if we access the transaction, change the status to Completed and for a line we enter a different value than expected, it will be triggered. But, what if we don't even access another line that has a 0 amount paid? Will the rule be triggered?

The answer is no. When executing a transaction in Update mode we may not want to modify the header and change only one line. In that case, what will be triggered? The header will always be updated, and then only the modified line. For it, everything will be triggered according to the evaluation tree.

Reality

The screenshot shows a web form for 'ReceiptOfPayment' with the following fields:

- Payment Id: 1
- Payment Date: 10/11/20
- Payment Status: Pending
- Customer Id: 1
- Customer Name: Joseph

Below the form is an 'Invoice' table:

Invoice Id	Invoice Total Amount	Total Payed
1	5400.00	5400
2	9000.00	0
3	810.00	0
0	0.00	0
0	0.00	0
0	0.00	0
0	0.00	0
0	0.00	0
0	0.00	0

At the bottom of the form are buttons: CONFIRM, CANCEL, DELETE.

Overlaid on the right is a code editor window titled 'ReceiptOfPayment * X' showing the following rule:

```

1 &ok = CheckAllValid(ReceiptOfPaymentId)
2   if Update and ReceiptOfPaymentStatus = Status.Completed
3     on BeforeComplete; //equivalent with on AfterLevel event
4
5 Error('Incomplete payments')
6   if Update and ReceiptOfPaymentStatus = Status.Completed and not &ok
7     on BeforeComplete; //equivalent with on AfterLevel event
8

```

Next to the code editor is a diagram of a computer monitor with a code icon in a circle to its left. Above the monitor is the text: `Parm(ReceiptOfPaymentId);`

Below the diagram is the following code snippet:

```

if InvoiceTotalAmount <> ReceiptOfPaymentInvoiceTotalPayed
  &ok = false
else
  &ok = true
endif

```

How would we solve this case?

One solution is to call a procedure to which we send the receipt ID, once we have waited for all the lines to be modified, and after these modifications have been made in the database, but before the commit, so we can undo them. This procedure runs through ALL the lines and makes sure that none is left with a different value for the attributes we're interested in.

Although for the Error rule it may seem that we do not need to condition the BeforeComplete event since the &ok variable to be evaluated is already conditioned, it is actually necessary. If we didn't condition the error to exactly the same event, it would break the dependencies between both rules.

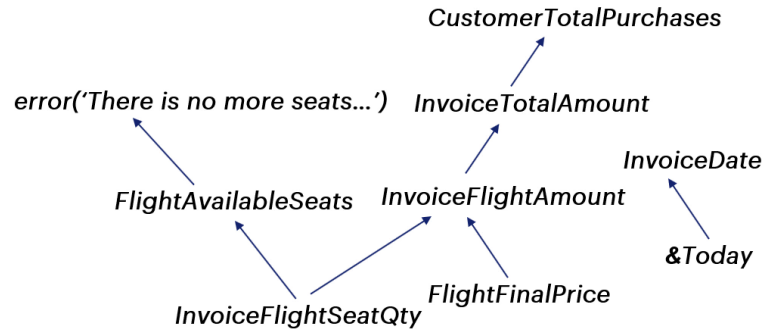
Each triggering event has its own evaluation tree, which means that if we condition many rules to the same event, it will order them **at the time the event occurs** according to its dependencies, as we saw before.

Evaluation Tree

```

(R) Default( InvoiceDate, &Today );
(R) Add( InvoiceTotalAmount, CustomerTotalPurchases );
(F) InvoiceTotalAmount = Sum( InvoiceFlightAmount )
(F) InvoiceFlightAmount = FlightFinalPrice * InvoiceFlightSeatQty
(F) FlightFinalPrice = FlightPrice * (1 - AirlineDiscountPercentage...)
(R) Subtract( InvoiceSeatQty, FlightAvailableSeats );
(R) Error( "There are no more seats for sale" ) if FlightAvailableSeats < 0;

```



In sum, the rules and formulas stated in a transaction are usually interrelated, and GeneXus determines the dependencies between them, as well as the order in which they are evaluated.

Reality

Payment Id

Payment Date

Payment Status

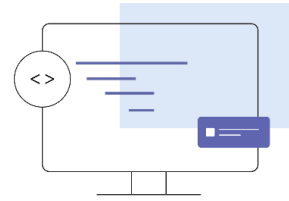
Customer Id

Customer Name Joseph

Invoice

Invoice Id	Invoice Total Amount	Total Payed
× 1 <input type="button" value="↑"/>	5400.00	5400
× 2 <input type="button" value="↑"/>	9000.00	9000
× 3 <input type="button" value="↑"/>	810.00	0
<input type="text" value="0"/> <input type="button" value="↑"/>	0.00	0
<input type="text" value="0"/> <input type="button" value="↑"/>	0.00	0
<input type="text" value="0"/> <input type="button" value="↑"/>	0.00	0
<input type="text" value="0"/> <input type="button" value="↑"/>	0.00	0
<input type="text" value="0"/> <input type="button" value="↑"/>	0.00	0

[New row]



```

if InvoiceTotalAmount <> ReceiptOfPaymentInvoiceTotalPayed
    &ok = false
else
    &ok = true
endif

```

```

ReceiptOfPayment * X
Structure | Web Form | Win Form | Rules * | Events | Variables | Help | Documentation | Patterns
1 &ok = CheckAllValid(ReceiptOfPaymentId)
2   if Update and ReceiptOfPaymentStatus = Status.Completed
3     on BeforeComplete; //equivalent with on AfterLevel event
4
5 Error('Incomplete payments')
6   if Update and ReceiptOfPaymentStatus = Status.Completed and not &ok
7     on BeforeComplete; //equivalent with on AfterLevel event
8

```

Sometimes, the evaluation tree doesn't determine the order of execution that we want: a clear example of this is the one we just saw, where we had to delay the triggering moment of the procedure that checks the line records and the subsequent error.

Navigation Report

Detailed Navigation Report

Table	Program	In Parameters	Out Parameters
Flight	Gx00B0		FlightId
InvoiceFlight	Gx00F1	InvoiceId	FlightId
Customer	Gx0010		CustomerId
Invoice	Gx00E0		InvoiceId

```

Level InvoiceFlight

FlightAvailableSeats Enabled = 0;
CustomerTotalPurchases Enabled = 0;

READ InvoiceFlight
WHERE
  InvoiceFlight InvoiceId = InvoiceId
  InvoiceFlight FlightId = FlightId
  INTO InvoiceFlightSeatQty

READ Flight
WHERE
  Flight FlightId = InvoiceFlight FlightId
  INTO Airline FlightAvailableSeats FlightPrice FlightDiscountPercentage

READ Airline ALLOWING NULLS
WHERE
  Airline AirlineId = Flight AirlineId
  INTO AirlineDiscountPercentage

FlightFinalPrice = FlightPrice * ( 1 - AirlineDiscountPercentage / 100) IF AirlineDiscountPercentage >= Flight
InvoiceFlightAmount = InvoiceFlightSeatQty * FlightFinalPrice

InvoiceTotalAmount =
  InvoiceTotalAmount.getoldvalue() + InvoiceFlightAmount IF insert; InvoiceTotalAmount.getoldvalue() + Invo
CustomerTotalPurchases = CustomerTotalPurchases.getoldvalue() + InvoiceTotalAmount - InvoiceTotalAmo
FlightAvailableSeats = FlightAvailableSeats.getoldvalue() + InvoiceFlightSeatQty.getoldvalue() IF delete; E
Error( "There is no more seats for sale" ) IF FlightAvailableSeats < 0

INSERT INTO InvoiceFlight (InvoiceId, InvoiceFlightSeatQty, FlightId)
UPDATE InvoiceFlight (InvoiceFlightSeatQty)
DELETE FROM InvoiceFlight
UPDATE Flight (FlightAvailableSeats)

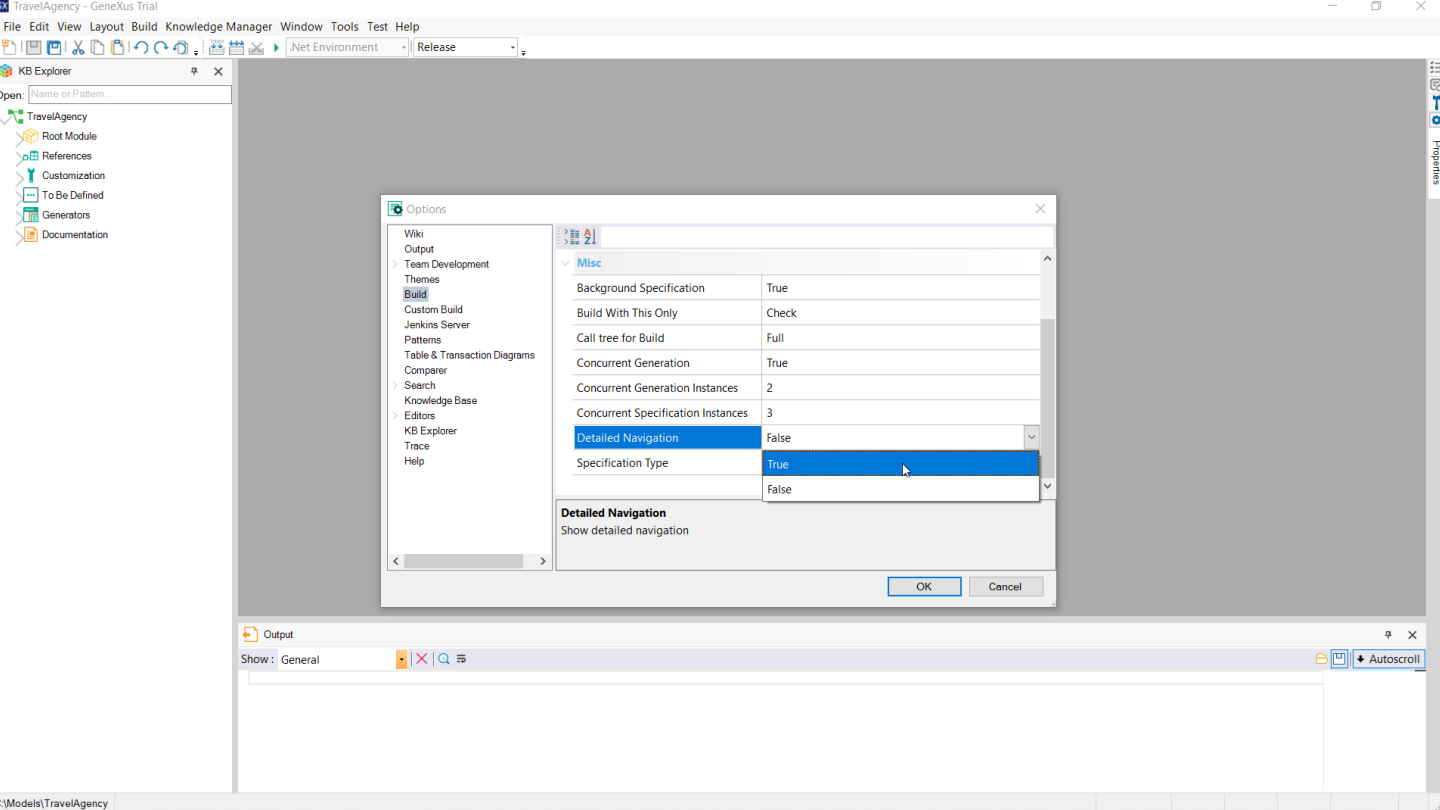
After Complete Rules
prc.CheckReceiptOfPayment Call(CustomerId, .. InvoiceId) IF insert

```

If you want to see in more detail the order of the evaluations triggered by GeneXus, you can use the detailed navigation list.

Here is the difference between the two... let's see, for example, that in the detailed navigation we are shown the rules and the moments when they will be triggered, which doesn't happen in the other case.

Detailed navigation can be useful in cases where we need to understand exactly where a formula or rule is being triggered, but it usually takes longer to specify, so we often don't need it.



To enable it, go to the menu options Tools > Options, and in the Build category activate the **Detailed Navigation** property.

GeneXus™

training.genexus.com
wiki.genexus.com
training.genexus.com/certifications