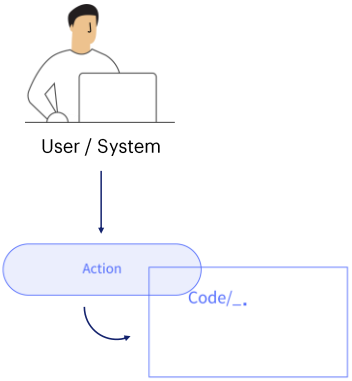
More about Events in Transactions

GeneXus™

Event: Concept



We have learned before that the Transaction object has events that can be programmed to control and define its behavior.

As a review, let's remember the events that are available:

Events in Transactions: Start event


```
Event Start
     /* Generated by Work With Pattern [Start] - Do not change */
     [web]
     {
     If not IsAuthorized(&PgmName)
          NotAuthorized(&PgmName)
     Endif

     &TrnContext.FromXml(&WebSession.Get(!"TrnContext"))
     &Insert_CountryId.SetEmpty()
     &Insert_CityId.SetEmpty()
     &Insert_CategoryId.SetEmpty()

     }
     /* Generated by Work With Pattern [End] - Do not change */
Endevent
```

The Start event defines an action to be executed when you open and start working with the transaction.

Events in Transactions : TrackContext event


```
Event TrackContext(&AttractionId)
     WCDetails.Object = AttractionsDetail.Create(&AttractionId)
Endevent
```

If the value of the &AttractionId variable is changed, this event will be triggered, which will cause the component on the screen to be reloaded from the Web Component AttractionDetail with the new value of the variable.

The Track Context event allows you to schedule what action to take when a change is made in the context of the transaction execution.

What do we mean by context? It is the context in relation to a screen, to a form.

When we move within a screen, we are changing the context of attributes and variables. This information about the context change is fundamental to define an action to be taken.

Events in Transactions : OnMessage event


```
Event OnMessage(&NotificationInfo)
  for each line
    if (&NotificationInfo.Id=&postid.ToString())
     //processs the notification data
    endif
  endfor
EndEvent
```

The OnMessage event, which is related to web notifications.

Events in Transactions : Insert, Update, Delete events


```
Event Insert(&Messages)
     If DocumentType = Type.Invoice
          &Invoice = New()
          &Invoice.InvoiceId = DocumentId
          &Invoice.InvoiceDate = DocumentDate
          &Invoice.InvoiceTotal = DocumentAmount
          &Invoice.Insert()
          &Messages = &Invoice.GetMessages()
     else
          &Receipt = New()
          &Receipt.ReceiptId = DocumentId
          &Receipt.ReceiptDate = DocumentDate
          &Receipt.ReceiptTotal = DocumentAmount
          &Receipt.Insert()
          &Messages = &Receipt.GetMessages()
     endif
Endevent
```

The Insert, Update, and Delete events which, as we have seen, apply to the specific case of dynamic transactions and allow programming its behavior so that they are updatable.

Events in Transactions : Exit event


```
Event Exit
     //Process code
Endevent
```

The Exit event allows programming the actions to be executed when the transaction is already closed.

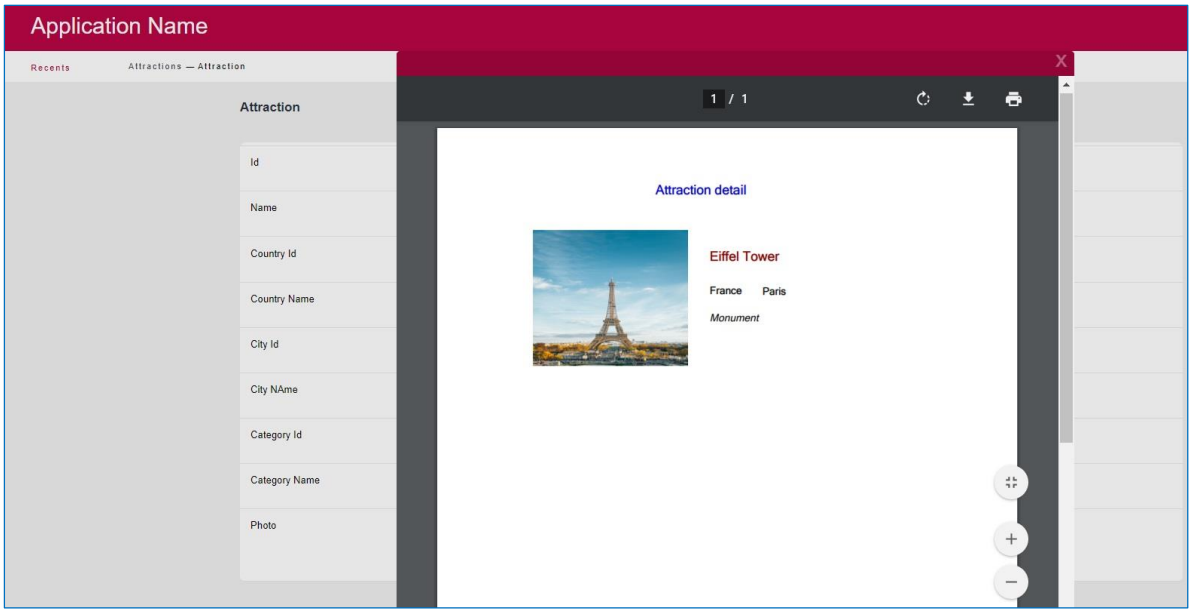
Events in Transactions : After Trn event


```
Event After Trn
      /* Generated by Work With Pattern [Start] - Do not change */
      [web]
      {
      If (&Mode = TrnMode.Delete and not &TrnContext.CallerOnDelete)
           WWAttraction()
      Endif

      Return
      }
      /* Generated by Work With Pattern [End] - Do not change */
EndEvent
```

And the AfterTrn event, which is triggered after the Commit has been executed. Looking back at what has already been discussed about rule triggering moments, this AfterTrn event is executed just like the triggering moment on AfterComplete.

New requirement...



. We are now going to meet a new requirement made by the Travel Agency.

We are requested that every time you work with the registration of a tourist attraction, and after the Commit, a popup window be shown with a PDF with the detail of that attraction.

New requirement ...

```
Event After Trn
     AttractionDetail.Popup(AttractionId)

     /* Generated by Work With Pattern [Start] - Do not change */
     [web]
     {
     If (&Mode = TrnMode.Delete and not &TrnContext.CallerOnDelete)
          WWAttraction()
     Endif

     Return
     }
     /* Generated by Work With Pattern [End] - Do not change */
EndEvent
```

We have already defined the PDF list named AttractionDetail, which receives by parameter the identifier of an attraction, and shows its detail. The question to answer now is: Where do we declare the call to this procedure to achieve the required functionality?

If we need to call it after the Commit, then we may consider, for example, declaring a rule in the Attraction transaction and conditioning it to the moment on AfterComplete. However, the requirement is to show the PDF list in a popup window and this popup method is not available in the rules of a transaction.

Therefore, we can resort to the AfterTrn event.

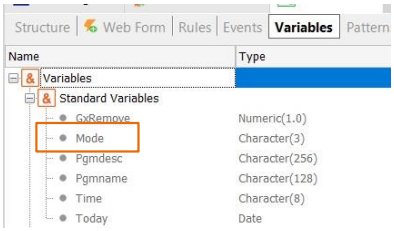
Since we have applied the Work With pattern, we already know that GeneXus automatically adds code in this Attraction transaction and that, in particular, it adds the return command in this AfterTrn event.

Therefore, we must call the list before this Return command is executed, and outside the marks of the code automatically generated as a result of applying the pattern.

New requirement ...

```
Event After Trn
    If &Mode = TrnMode.Insert
        AttractionDetail.Popup(AttractionId)
    endif
    /* Generated by Work With Pattern [Start] - Do not change */
    [web]
    {
    If (&Mode = TrnMode.Delete and not &TrnContext.CallerOnDelete)
        WWAttraction()
    Endif

    Return
    }
    /* Generated by Work With Pattern [End] - Do not change */
EndEvent
```



Note that we can condition the call to this PDF depending on the execution mode of the transaction.

If we want the detail to be shown only when a new attraction is being inserted, and not when it is being modified or deleted, then we can write something like this...

Remember that the &Mode variable is a system variable that saves the mode in which the transaction is being executed. The value of that variable is received in the Parm rule automatically declared when the Work With pattern was applied, and depends on the action selected by the user in the main screen.

Events in Transactions - Restrictions

- Unable to make updates to the database

  It is not possible to directly assign a value to an attribute, but it is possible to call a procedure to perform the necessary update.

- Commit command

```
Event Insert
     If DocumentType = Type.Invoice
          &Invoice = New()
          &Invoice.InvoiceId = DocumentId
          &Invoice.InvoiceDate = DocumentDate
          &Invoice.InvoiceTotal = DocumentAmount
          &Invoice.Insert()
     else
          &Receipt = New()
          &Receipt.ReceiptId = DocumentId
          &Receipt.ReceiptDate = DocumentDate
          &Receipt.ReceiptTotal = DocumentAmount
          &Receipt.Insert()
     endif
Endevent
```

Finally, it is worth mentioning that in transaction events it is not possible to make updates to the database directly, although it is possible to work with business components.

When we work with business components, as in the case of events associated with dynamic transactions, we do not declare the commit command because we are in the context of a transaction, and therefore the Commit on exit property is set to True.

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