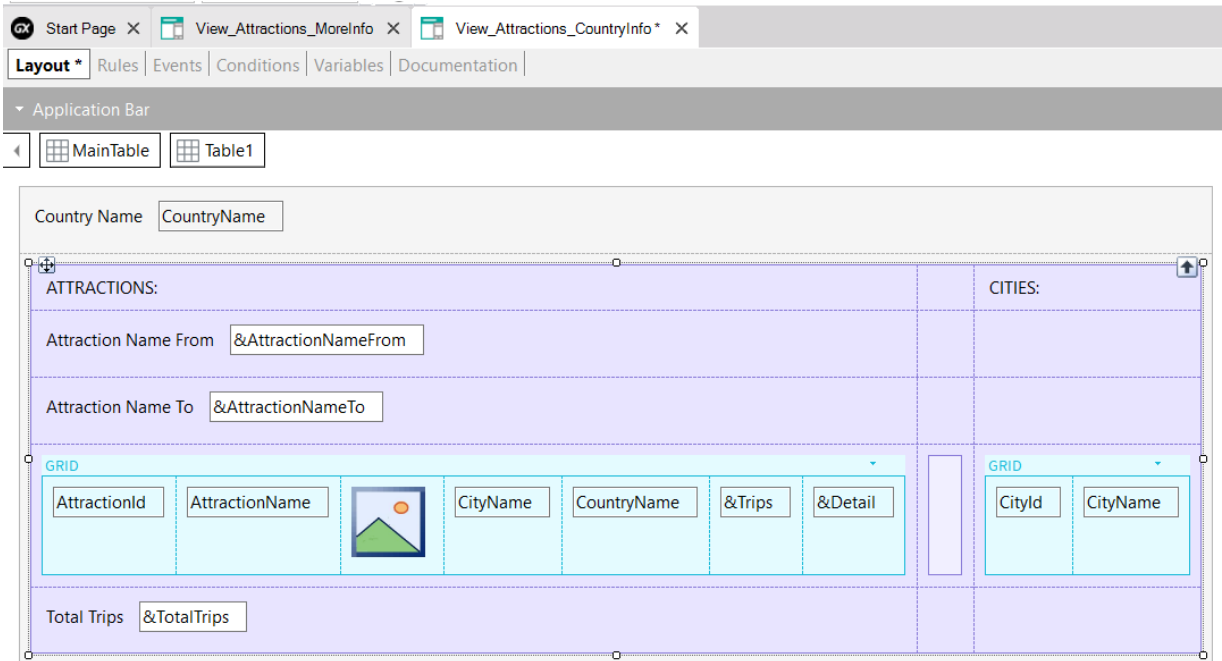


Panel with multiple grids

GeneXus™

We have seen how to use a grid in a panel object. We will now see the considerations we need to have when adding more than one grid to the panel. This will give us the knowledge necessary to continue the development of our travel agency application.

Panel object with more than one grid



We will build a panel object that shows the attractions and cities of a country received by parameter.

For that, we make a Save As of the View_Attractions_MoreInfo panel and name it View_Attractions_CountryInfo. Then we remove its main object setting, because the goal is to have it called from the grid of attractions when the country's name is clicked on.

In the form, we are going to remove the titles we had, since we will show titles by section and in the grid we are going to add a &Detail variable of Character type. This variable will be used later to invoke a panel that shows the details of each attraction.

In the events, we delete the lines that assigned the text to the titles.

In the form, we add the CountryName attribute and delete the &CountryId variable of Dynamic combo type, since we will not choose a country but receive it by parameter. We remove the variable from the conditions. We delete the ControlValueChanged event associated with the variable we removed, and the where clause that used it; in the Parm rule, we change the &CountryId variable to the CountryId attribute.

To group all the data of the attractions and then the data of the cities, we insert a Table control from the toolbar and place the variables AttractionNameFrom, AttractionNameTo, the grid and the variable TotalTrips inside the table. To the right we insert another table as a separator and then we insert a grid to show the cities of the country, with the attributes CityId and CityName.

We add titles to the sections, placing a textblock "ATTRACTIONS:" above the attractions section and "CITIES:" above the cities section.

Changing the style of the rows and columns of a table

The screenshot shows the GeneXus IDE interface. The main workspace displays a form with a table. The table has columns for 'AttractionId', 'AttractionName', a landscape image, 'CityName', and 'CountryName'. A 'Columns Style' dialog box is open, showing the following configuration:

Colu.	Width
1	50%
2	25%
3	25%

The dialog also shows the 'Unit' set to 'Percentage' and the 'Value' field set to '25'. The 'Properties' panel on the right shows the 'Table1' control with the following 'Appearance' properties:

Property	Value
Columns Style	33%;33%;34%
Rows Style	20%;20%;20%;20%;20%
Width	100%
Height	100%
Auto Grow	True
Class	Table
Background	(none)
Visible	True
Invisible Mode	Keep Space
Enabled	True

To define how we want the contents of a table to be shown, we must change the size of the table rows or columns. For example, we want the grids to have enough space in the corresponding row to be displayed correctly and to assign more space to the column of the table that shows the content of the attractions, because we have more things to show than for the cities.

For that we have the Columns Style and Rows Style properties of the table. First, we will change the columns that we know are 3, the column containing the attractions' data, the column with the separating table, and the column containing the cities' data. If we select Table1 and click on the Column Style property, by default each column takes up 33% of the available space.

The possible values to be assigned to the columns' width are a percentage of the total table space or in Device Independent Pixels (DIPs). This measure allows assigning units that are an abstraction of a pixel, don't depend on the platform, and will later be converted to real pixels when the application is executed. The number of pixels that each DIP will take up depends on the dimensions of the screen. This allows us to scale to different screen sizes using standard sizes.

Let's assign 50% of the available space to the first column, 25% to the second column, and 25% to the third column. This space is what remains after having assigned the columns with DIPs; that is, the percentages are relative to the value that results from subtracting the fixed values (in DIPs) from the total width.

Changing the style of the rows and columns of a table

The screenshot displays the GeneXus IDE interface. On the left, a design canvas shows a table with five rows. The first three rows and the fifth row contain text input fields, while the fourth row contains a grid with five columns: 'AttractionId', 'AttractionName', an image icon, 'CityName', and 'CountryName'. A 'Total Trips' field is located below the table. A 'Rows Style' dialog box is open, showing a table of row heights:

Row	Height
1	pd
2	pd
3	pd
4	100%
5	pd

The dialog also shows 'Unit' set to 'Platform Default' and 'Value' set to 20. On the right, the 'Properties' panel for 'Table1' is visible, showing the 'Rows Style' property set to '20%;20%;20%;20%;20%'.

Now we change the height of the rows. We click on the Rows Style property and see that by default all the rows have a 20% assigned, except the fourth row which is where the grids are located.

Here we can assign the values using percentages, DIPs and Platform Default. This last value corresponds to: "Using the best value depending on the platform and the context;" that is, it varies between platforms and for the same platform, it also depends on the cell's content.

We are going to assign that value to all the rows except row 4 where the grids are, which we set to occupy the 100% that remains available.

Invocation to the detail panel

The screenshot shows the GeneXus IDE interface. At the top, there are tabs for 'Start Page', 'View_Attractions_MoreInfo', 'View_Attractions_CountryInfo', and 'Navigation View'. Below the tabs is a menu bar with 'Layout', 'Rules', 'Events', 'Conditions', 'Variables', and 'Documentation'. The 'Application Bar' contains buttons for 'MainTable', 'Grid1', 'Grid1Table', and 'CountryId'. The main workspace displays a form layout with several fields: 'TextblockTitleLine1', 'TextblockTitleLine2', a 'Country' dropdown menu with '&CountryId', 'Attraction Name From' with '&AttractionNameFrom', 'Attraction Name To' with '&AttractionNameTo', a 'GRID' containing columns for 'AttractionId', 'AttractionName', a landscape image, 'CityName', 'CountryId', 'CountryName', and '&Trips', and 'Total Trips' with '&TotalTrips'. An arrow points from the 'CountryName' field in the grid to an event configuration window on the right. The event configuration shows 'Event CountryName.Tap' and 'View_Attractions_CountryInfo(CountryId, "", "")' with 'Endevent' below it.

If we right-click to see the navigation of the object we built, the Output window shows an error; if we look at the error in the navigation list, it says that the Load event cannot be programmed if we have multiple grids.

If we go to the events, we see that we still have the Load event programmed as we had it in the original object. Here we don't consider what we said before about the convenience of using the Load event of the grid and not the generic one to avoid that situation.

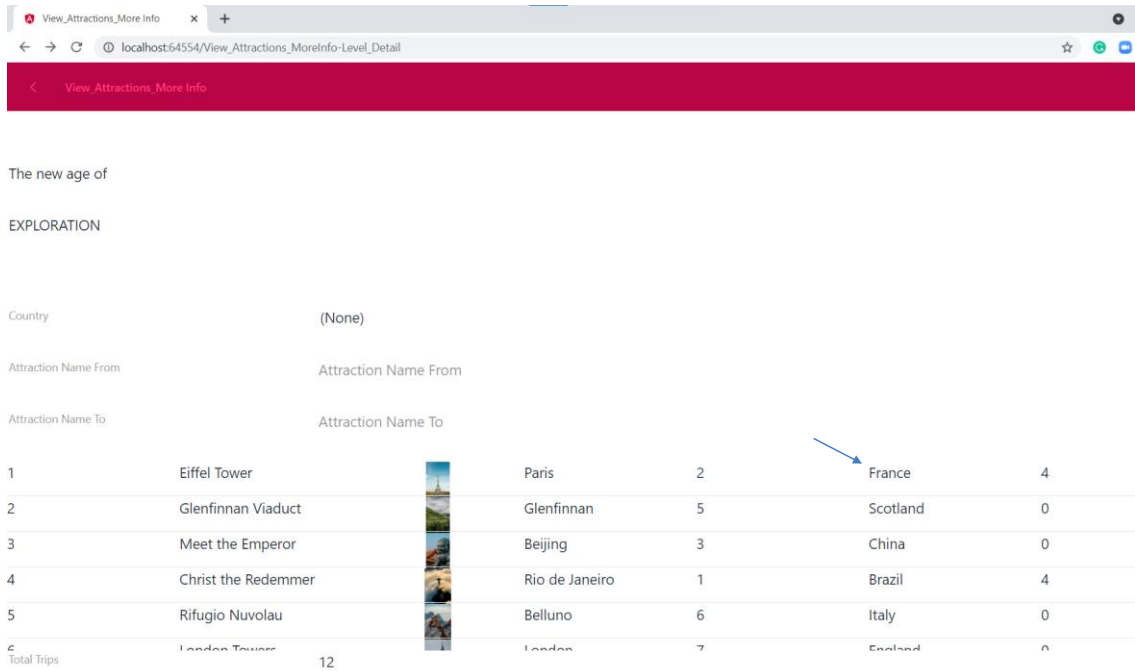
So, we add Grid1 to the Load. If we select View Navigation, we see that the problem has been solved.

Before running, we go to the [View_Attractions_MoreInfo](#) panel and add the CountryId attribute required to go to the information panel of a country, when we click on the name of the country in the grid.

Next, in the CountryName attribute we add a Tap event where we write the invocation to the [View_Attractions_CountryInfo](#) panel, passing it the CountryId as parameter.

We run it to see all this.

Example at runtime



View_Attractions_More Info

localhost:64554/View_Attractions_MoreInfo-Level_Detail

< View_Attractions_More Info







The new age of

EXPLORATION

Country (None)

Attraction Name From Attraction Name From

Attraction Name To Attraction Name To

1	Eiffel Tower		Paris	2	France	4
2	Glenfinnan Viaduct		Glenfinnan	5	Scotland	0
3	Meet the Emperor		Beijing	3	China	0
4	Christ the Redemmer		Rio de Janeiro	1	Brazil	4
5	Rifugio Nuvolau		Belluno	6	Italy	0
6	London Towers		London	7	England	0
Total Trips		12				

If we click on France...

Example at runtime




The screenshot shows a web browser window with the URL `localhost:64554/app/View_Attractions_CountryInfo-Level_Detail?countryid=2&attractionnamefrom=&attractionnameto=`. The page title is "View Attractions, Country Info". The main content area shows the following information:

Country Name: France

ATTRACTIONS:

Attraction Name From: Attraction Name From

Attraction Name To: Attraction Name To

1	Eiffel Tower		Paris	France	4	Details
7	Louvre		Paris	France	0	Details
11	Matisse ...		Nice	France	4	Details

CITIES:

1	Paris
2	Nice

Total Trips: 8

The information of the country France is opened, showing the attractions with the total of trips of each attraction and the total of trips to France. To the right are the cities of that country.

Navigation list of the new panel

The screenshot displays the GeneXus IDE interface for configuring a new panel. On the left, a navigation tree shows the project structure with 'View_Attractions_CountryInfo' expanded to show 'Level_Detail_Grid1', 'Level_Detail_Grid2', and 'Level_Detail'. The main area is divided into several sections:

- Properties:**
 - Name:** View_Attractions_CountryInfo_Level_Detail_Grid1
 - Description:** View_Attractions_CountryInfo_Level_Detail_Grid1
 - Output Devices:** None
 - Environment:** C# | Default (C#)
 - Spec. Version:** 17_0_7-154604
 - Form Class:** HTML
 - Program Name:** View_Attractions_CountryInfo
 - Parameters:** in: CountryId, in: &AttractionNameTo, in: &AttractionNameFrom, in: ⋆ out: View_Attractions_CountryInfo
- LEVELS:**
 - For Each Attraction (Line: 5)**
 - Order:** CountryId
 - Index:** IATTRACTION1
 - Navigation filters:** Start from: CountryId = @CountryId; Loop while: CountryId = @CountryId
 - Constraints:** AttractionName >= &AttractionNameFrom WHEN not &AttractionNameFrom.isempty(); AttractionName <= &AttractionNameTo WHEN not &AttractionNameTo.isempty()
 - Join location:** Server
 - Optimizations:** Server Paging
- Navigation List (highlighted in orange):**
 - Attraction (AttractionId)
 - Country (CountryId)
 - City (CountryId, CityId)
 - count(TripDate) navigation
 - TripAttraction (AttractionId)
 - Trip (TripId)

If we go to the navigation list, we see that now there is a Level_Detail entry for Grid1 and another one for Grid2.

For Grid1 we see the access to the Attraction table and the navigation of the formula that we saw when we implemented the panel

[View_Attractions_MoreInfo](#).

Navigation list of the new panel

Pattern:

- View_Attractions_CountryInfo
 - Level_Detail_Grid1
 - Level_Detail_Grid2**
 - Level_Detail


Data Provider View_Attractions_CountryInfo_Level_Detail_Grid2 Navigation Report

Name:	View_Attractions_CountryInfo_Level_Detail_Grid2	Environment:	Default (C#)
Description:	View_Attractions_CountryInfo_Level_Detail_Grid2	Spec. Version:	17_0_7-154604
Output Devices:	None	Form Class:	HTML
Parameters:		Program Name:	View_Attractions_CountryInfo_Level_Detail_Grid2
			in: CountryId, in: &AttractionNameTo, in: &AttractionNameTo, in: &start, in: &end, out: View_Attractions_CountryInfo_Level_Detail_Grid2

LEVELS

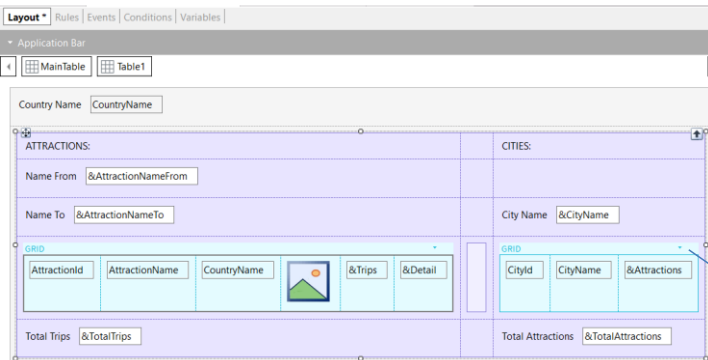
For Each City (Line: 4)

Order:	CountryId	
	Index: ICITY	
Navigation filters:	Start from:	CountryId = @CountryId
	Loop while:	CountryId = @CountryId
Optimizations:	Server Paging	

=City (CountryId, CityId)

If we select the node corresponding to Grid2, we see that the grid is accessing the City table to show the cities, filtering by CountryId, the attribute received in the Parm rule.

We add a filter by city and totals by city and country.



```

1 Event Start
2   &Detail = "Details"
3 Endevent
4
5 Event Grid1.Load
6   &Trips = Count(TripDate)
7 Endevent
8
9 Event Grid2.Load
10  &Attractions = Count(AttractionName)
11 Endevent
12
13 Event Grid1.Refresh
14  &TotalTrips = 0
15  For each Trip.Attraction
16    Where AttractionName >= &AttractionNameFrom when not &AttractionNameFrom.IsEmpty()
17    Where AttractionName <= &AttractionNameTo when not &AttractionNameTo.IsEmpty()
18    &TotalTrips += 1
19  Endfor
20 Endevent
21
22 Event Grid2.Refresh
23  &TotalAttractions = 0
24  For each Attraction
25    &TotalAttractions += 1
26  Endfor
27 Endevent

```

In a similar way to the totals shown for the attractions and the filter by name, we will show the number of attractions of each city and in the country, and we will add a filter by city name.

In the Variables tab, we create an `&Attractions` variable, another one called `&TotalAttractions` and a third one called `&CityName`.

Now we add the `&Attractions` variable to the grid by setting its Label Position property to None. Then we add `&TotalAttractions` below the grid and the filter variable `&CityName` above the city grid. In the grid we add the necessary condition for the filter.

Before that, we take the opportunity to assign the Base Transaction property in **City**. Here we click on Conditions and write the filter condition using the like operator, since we will not filter by name range, but by a similar name.

Then we add the `Grid2.Load` event where we load the `&attractions` variable with a Count formula with the `AttractionName` attribute. Since the base table of the grid is **City**, the formula will only count the attractions of the city corresponding to each line.

To calculate the total number of attractions, for the same reasons as when we calculated the total number of trips, we write the `Grid2.Refresh` event, in which we programmed a For Each command on the Attractions table to count the attractions, which will be filtered by the attribute of the country identifier received by parameter.

We calculate it every time a line is going to be loaded; that is, in the `Grid2 Load` event.

Navigation list of the panel with the new changes

Pattern:

Parameters: in: C
&Att
out:
View

LEVELS

For Each Attraction (Line: 5)

Order: [CountryId](#)
Index: IATTRACTION1

Navigation filters: Start from: [CountryId = @CountryId](#)
Loop while: [CountryId = @CountryId](#)

Optimizations: count(*)

- [Attraction \(AttractionId \)](#)

For Each City (Line: 11)

Order: [CountryId](#)
Index: ICITY

Navigation filters: Start from: [CountryId = @CountryId](#)
Loop while: [CountryId = @CountryId](#)

Constraints: [CityName](#) like &CityName WHEN not &CityName. isempty() ←

Join location: Server

Optimizations: Server Paging

- [City \(CountryId, CityId \)](#)
 - [count\(AttractionName \) navigation](#)
 - [Attraction \(CountryId, CityId \)](#)

If we now see the panel navigation list, in the node `Level_Detail_Grid2` we see the navigation of the For Each command to the Attraction table, and below it the navigation of the Count formula on the Attraction table.

And here we see the filter we added by **CityName**.

Viewing totals by city and country







View_Attractions_More Info

The new age of
EXPLORATION

Country (None)

Attraction Name From Attraction Name From

Attraction Name To Attraction Name To

1	Eiffel Tower		Paris	2	France	4
2	Glenfinnan Viaduct		Glenfinnan	5	Scotland	0
3	Meet the Emperor		Beijing	3	China	0
4	Christ the Redemmer		Rio de Janeiro	1	Brazil	4
5	Rifugio Nuvolau		Belluno	6	Italy	0
6	London Towers		London	7	England	0
Total Trips		12				

Let's run our main object, [View_Attractions_MoreInfo](#), to see what we did.

We click on France again...

Viewing totals by city and country

View_Attractions_More Info x +

localhost:52618/app/View_Attractions_CountryInfo-Level_Detail.countryid=2,attractionnamefrom=-attractionname=




View_Attractions_Country Info

Country Name France

ATTRACTIONS:

Attraction Name From Attraction Name From

Attraction Name To Attraction Name To

1	Eiffel Tower		Paris	France	4	Details
7	Louvre		Paris	France	0	Details
11	Matisse ...		Nice	France	4	Details

CITIES:

City Name	City Name	
1	Paris	2
2	Nice	1

Total Trips 8

Total Attracti... 3

Now we can see the total of attractions of each city in France and the total of attractions of the country France.

Order of execution of events



We saw that because we had more than one grid in the panel, we had to use the Refresh and Load event of each grid.

But after adding these events, we wonder what the triggering order of these events will be in relation to the events of the panel object.

When the panel object is executed for the first time, the events will be triggered in the following order:

First, the Start event, only once (If it is necessary to go to the Server, as in this example).

Then the generic Refresh event; that is, the panel's own Refresh event.

Finally, the Refresh of the first grid. If it has a base table, the Load event of the grid will be executed as many times as records are retrieved from the database, filtering the corresponding records. If it doesn't have a base table, the Load event of the grid is executed only once, and if it is a grid based on an SDT, the Load event is not executed.

And then the same with the Refresh and Load events of the second grid.

What do you want to refresh?



```

Start
Refresh
Grid1.Refresh      Grid2.Refresh
Grid1.Load         Grid2.Load
  
```

```

Event 'User-event'
...
Form.Refresh
endevent
  
```

```

Event 'User-event'
...
Refresh
endevent
  
```

```

Event 'User-event'
...
Grid1.Refresh()
endevent
  
```

Since there is more than one grid, the Refresh command must also be specialized to indicate which grid you want to refresh.

The generic Refresh command (the one we had seen when we used it in the View_Attractions_MoreInfo panel) causes the generic Refresh to be executed, as well as the Refresh and Load event of each grid (that is, everything but Start).

And now we also have the Refresh method of a grid, which will refresh only the grid; that is to say, run the grid Refresh and Load (n times, once, or never) depending on whether the grid has a base table, or if it's a grid of a collection SDT variable, respectively.

In this video, we saw how we can work with multiple grids in a Panel object, in this case with parallel grids and the considerations we need to have when invoking the events of each grid.

GeneXus™

training.genexus.com

wiki.genexus.com

training.genexus.com/certifications